

UNCLASSIFIED

Report Number: I331-010R-2004

Guide to Microsoft .NET Framework Security

Systems and Network Attack Center



Updated: September 22, 2004
Version 1.4

National Security Agency
9800 Savage Road
Ft. George G. Meade, MD 20755-6704

securew2k@dewnet.ncsc.mil
w2kguides@nsa.gov
snacguides@nsa.gov

UNCLASSIFIED

Warnings

- Do not attempt to implement any of the settings in this guide without first testing in a non-operational environment.
- This document is only a guide containing recommended security settings. It is not meant to replace well-structured policy or sound judgment. Furthermore this guide does not address site-specific configuration issues. Care must be taken when implementing this guide to address local operational and policy concerns.
- The security recommendations described in this document only apply to Microsoft .NET Framework installed on Microsoft Windows 2000, Windows XP, or Windows Server 2003 systems. This document does not apply to the .NET Compact Framework.
- This document applies only to Microsoft .NET Framework version 1.0 and 1.1. See Microsoft Corporation at www.microsoft.com for the latest updates and versions of the .NET Framework.
- Citation of a work in this document does not imply endorsement by the National Security Agency of the content of such a work, including its accuracy, applicability, or suitability for use in operational environments. This document does not provide a comprehensive bibliography of resources.
- Reference to an information security standard or guideline does not imply a claim that the .NET Framework is in conformance to that standard or guideline. In particular, this guide makes no claim about the conformance of the .NET Framework to Standard ECMA-335, nor does this guide apply to any other implementation of Standard ECMA-335.
- This guide makes reference to fictitious Web sites and domain names in its examples. These names are for demonstration purposes only and do not endorse any actual Web sites or domain names.

Trademark Information

- Microsoft, MS-DOS, Windows, Windows 2000, Windows XP, and .NET are either registered trademarks or trademarks of Microsoft Corporation in the U.S.A. and other countries. All other names are registered trademarks or trademarks of their respective companies.
- Some parts of this document use Microsoft copyrighted materials with their permission.

Table of Contents

Warnings	ii
Trademark Information	ii
Table of Contents	iii
Table of Figures	xv
Table of Tables	xviii
Introduction	xix
Purpose.....	xix
Assumptions.....	xix
Conventions Used In This Guide.....	xx
Document Summary	xx
.NET Framework Overview	1
What Is the .NET Framework?	1
Security Principles and the .NET Framework	2
Accountability	3
Availability.....	3
Confidentiality.....	4
Integrity	5
The .NET Framework Security Model	5
.NET Framework Security Model Components.....	6
Mediated Access to Resources.....	7
Walking the Call Stack	7
Features of the .NET Framework Security Model	9
Assemblies	9
Evidence-Based Access Control.....	10
Types of Assembly Evidence	10
Intrinsic Assembly Evidence	10
Location Assembly Evidence	11
.NET Framework Protected Resources.....	12
Protected Local Machine Resources	13
File System	13
<i>File IO</i>	14

<i>File Dialog</i>	14
<i>Isolated Storage File</i>	15
Local Assembly Data Stores	17
Local Application Data Stores	17
Roaming Assembly and Application Data Stores	19
User Interface Elements	19
<i>Windowing</i>	19
<i>Clipboard</i>	20
Reflection	21
Protected Network Resources	22
Printers	22
Domain Name System (DNS)	22
Network Sockets.....	23
Web Access	23
Message Queues	24
Windows Services	25
Databases.....	25
<i>Version 1.0 of the .NET Framework</i>	26
<i>Version 1.1 of the .NET Framework</i>	27
<i>OLE DB</i>	31
<i>ODBC</i>	32
<i>Oracle Client</i>	32
<i>SQL Client</i>	32
<i>Summary of Trust Requirements</i>	33
Protected Administrative Resources	33
Security Settings.....	34
<i>Runtime Environment</i>	34
Extend Infrastructure	34
Enable Remoting Configuration	35
Enable Serialization Formatter.....	36
Enable Thread Control.....	36
Allow Principal Control.....	37
Create and Control Application Domains	37
<i>Execution</i>	37
Enable Assembly Execution	38

Skip Verification.....	39
Allow Calls to Unmanaged Assemblies	39
<i>CAS Policy</i>	40
Allow Policy Control.....	40
Allow Domain Policy Control	40
Allow Evidence Control	41
Assert Any Permission That Has Been Granted	41
Performance Counters.....	42
Environment.....	44
Event Logs	48
Registry	49
Directory Services.....	49
Summary	50
Recommendations in This Section.....	51
CAS Policy.....	54
Code Groups and Membership Conditions	55
Union Code Groups and First Match Code Groups.....	58
File Code Groups and Net Code Groups	59
<i>Copying File Code Groups and Net Code Groups</i>	60
Named Permission Sets	61
Policy Levels	65
Permission Resolution.....	66
Level Permission Set	66
Allowed Permission Set.....	67
Granted Permission Set.....	67
<i>Assembly Permission Requests</i>	68
<i>Computation of the Granted Permission Set</i>	68
Code Group Attributes.....	69
<i>Exclusive</i>	69
<i>Level Final</i>	69
Policy Enforcement	70
Summary	71
Recommendations in This Section.....	72
Deploying .NET Framework CAS Policy Using Group Policy.....	73
Deployment Options	73

Creating a Windows Installer Package for CAS Policy Deployment.....	74
Deploying CAS Policy Using Group Policy Objects	76
Creating or Selecting a GPO Using the Active Directory Users and Computers Console	76
Creating or Selecting a GPO Using the Group Policy Management Console	78
Adding an Installer package to a GPO Using the Group Policy Object Editor	78
Deployment Modes of Group Policy	80
Synchronous and Asynchronous Modes	80
Foreground Policy Application	81
<i>Fast Logon Optimization</i>	81
Disabling Fast Logon Optimization	81
Background Policy Application	81
Summary	82
Group Policy Processing.....	82
Group Policy Processing Precedence.....	82
Precedence of Active Directory Containers	82
<i>Group Policy Processing Example 1</i>	82
Precedence of Linked GPOs.....	83
<i>Group Policy Processing Example 2</i>	84
Precedence of Software Installation Packages	84
<i>Group Policy Processing Example 3</i>	85
Loopback Processing	85
Forcing Policy Deployment	86
Forcing Policy Deployment Locally	86
Forcing Policy Deployment Remotely	86
Uninstalling CAS Policy.....	88
Summary	88
Recommendations in This Section.....	88
URL Security Zones and the .NET Framework Zone Membership Condition.....	91
URL Security Zone Settings	93
Internet Explorer Enhanced Security Configuration.....	94
URL to Zone Mappings	94
Summary	96
Recommendations in This Section.....	96
Cryptographic Localization in the .NET Framework.....	97

The <cryptoClass> element.....	108
The <nameEntry> element.....	108
The <oidEntry> element.....	109
Cryptographic Localization Examples.....	111
Cryptographic Localization Example 1.....	111
Cryptographic Localization Example 2.....	112
Cryptographic Localization Example 3.....	113
Summary.....	113
Recommendations in This Section.....	114
Administrative Tasks and Tools.....	115
Security-Related Administrative Tasks Summary.....	115
Administrative Tools Summary.....	117
Task Descriptions.....	118
General .NET Framework Tasks.....	118
List the .NET Framework versions installed.....	118
Global Assembly Cache tasks.....	118
Enable or disable the Assembly Cache Viewer.....	118
View cache contents.....	119
Add an assembly to the GAC, Delete an assembly from the GAC, View properties of an assembly installed in the GAC.....	119
View or modify GAC properties, Clear the Download Cache.....	119
Isolated Storage tasks.....	120
List Isolated Storage File stores (including roaming) for the current user, Remove all Isolated Storage File stores (including roaming) for the current user.....	120
Code Access Security policy tasks.....	120
Migrate CAS policy from one .NET Framework version to another.....	120
Create a CAS policy deployment package.....	121
Enable or disable CAS policy.....	121
Enable or disable Execution permission checking.....	121
Build a CAS policy cache file.....	121
Reset all CAS policy levels to default settings.....	121
Recover the previous settings for a CAS policy level.....	121
View Code Groups, Add or remove a Code Group, Rename a Code Group, Set or clear the Exclusive or Level Final attribute of a Code Group, Change a Code Group's Membership Condition, Change a Code Group's associated Named Permission Set, View	

Named Permission Sets, Add or remove a Named Permission Set, Modify a Named Permission Set	122
View Policy Assemblies, Enroll or withdraw a Policy Assembly	122
List Code Groups to which an assembly belongs, View an assembly’s Allowed Permission Set	123
Adjust the Allowed Permission Set for an assembly, Create a tailored Code Group, Use the Trust an Assembly Wizard	123
<i>Creating a Tailored Code Group</i>	124
System security tasks	126
View publisher certificate verification settings, Adjust publisher certificate verification settings.....	126
<i>Setting 1: Trust Test Root Certificates</i>	127
<i>Setting 2: Check for Expired Certificates</i>	127
<i>Setting 3: Check for Revoked Certificates</i>	128
<i>Settings 4-7: Automatically Trust Certificates Whose Revocation Status Cannot Be Determined</i>	128
<i>Setting 8: Invalidate Version 1 Signed Objects</i>	129
<i>Setting 9: Check for Revoked Timestamp Signer Certificate</i>	129
<i>Setting 10: Only Trust Items Found in the Personal Trust Database</i>	130
Set the CSP used by the CLR when strong-naming assemblies.....	130
Assembly tasks	130
Validate and verify an assembly.....	130
View an assembly’s strong name, View the public key token corresponding to a public key	132
Strong name an assembly	132
Verify an assembly’s strong name	132
Enroll an assembly for strong name simulation, Withdraw an assembly from strong name simulation, List assemblies enrolled for strong name simulation	133
View an assembly’s publisher certificate	133
Verify the trust associated with an assembly’s Authenticode digital signature	134
View an assembly’s permission requests and declarative permission constraints	134
View the list of configured assemblies, Configure an assembly, Delete the configuration information for an assembly	135
<i>Introduction to Binding Policy</i>	135
<i>Application Policy</i>	136
<i>Publisher Policy</i>	136
<i>Machine Policy</i>	137

<i>Summary</i>	137
Application configuration tasks.....	138
Add an application to be configured, Configure application properties	138
View assembly dependencies for an application, View list of assemblies configured for an application, Configure an assembly for an application, Fix an application (roll back application Binding Policy)	138
Configure Remoting Services for an application.....	138
Summary.....	138
Recommendations in This Section	138
Administrative Tools Reference	141
caspol.exe – .NET Framework Code Access Security Policy Tool	141
Syntax.....	141
Tasks.....	142
Enable or disable CAS policy	143
Enable or disable Execution permission checking.....	143
Build a CAS policy cache file.....	143
Reset all CAS policy levels to default settings	144
Recover the previous settings for a CAS policy level	144
View Code Groups.....	144
Add or remove a Code Group.....	145
Rename a Code Group.....	146
Set or clear the Exclusive or Level Final attribute of a Code Group	146
Change a Code Group’s Membership Condition.....	147
Change a Code Group’s associated Named Permission Set	147
View Named Permission Sets.....	148
Add or remove a Named Permission Set	148
Modify a Named Permission Set	148
View Policy Assemblies	149
Enroll or withdraw a Policy Assembly	149
List Code Groups to which an assembly belongs	149
View an assembly’s Allowed Permission Set.....	150
Create a tailored Code Group	150
certmgr.exe – Microsoft Certificate Manager Tool.....	150
Syntax.....	151
Tasks.....	151

View an assembly’s publisher certificate	151
chktrust.exe – Microsoft Authenticode Signature Verification Tool.....	151
Syntax	152
Tasks	152
Verify the trust associated with an assembly’s Authenticode digital signature	152
explorer.exe/shfusion.dll – Windows Explorer/ Assembly Cache Viewer	152
Tasks	153
View cache contents	153
Add an assembly to the GAC	154
Delete an assembly from the GAC	155
View properties of an assembly installed in the GAC.....	155
View or modify GAC properties	156
Reset all CAS policy levels to default settings.....	157
gacutil.exe – .NET Global Assembly Cache Utility	158
Syntax	158
Tasks	158
View cache contents	158
Add an assembly to the GAC	159
Delete an assembly from the GAC	161
<i>Example</i>	162
Clear the Download Cache.....	163
migpol.exe – CAS Policy Migration Tool	163
Syntax	163
Tasks	164
List the .NET Framework versions installed	164
Migrate CAS policy from one .NET Framework version to another	164
mscorcfg.msc – .NET Framework Configuration Tool	165
permview.exe – .NET Framework Permission Request Viewer	165
Syntax	166
Tasks	166
View an assembly’s permission requests and declarative permission constraints	166
peverify.exe – .NET Framework PE Verifier	166
Syntax	167
Tasks	167

Validate and verify an assembly	167
regedit.exe – Registry Editor.....	167
Syntax.....	168
Tasks.....	168
Change a registry setting from the console.....	168
Enable or disable the Assembly Cache Viewer	168
secutil.exe – Microsoft .NET Framework Security Utility	168
Syntax.....	168
Tasks.....	169
View an assembly’s strong name.....	169
View an assembly’s publisher certificate	169
setreg.exe – Software Publishing State Tool.....	170
Syntax.....	170
Tasks.....	170
View publisher certificate verification settings	170
Adjust publisher certificate verification settings	170
sn.exe – .NET Framework Strong Name Utility	171
Syntax.....	171
Tasks.....	171
Enroll an assembly for strong name simulation.....	172
Withdraw an assembly from strong name simulation.....	172
List assemblies enrolled for strong name simulation.....	173
Verify an assembly’s strong name	173
View the public key token corresponding to the public key in an assembly’s manifest.....	173
Strong name an assembly	173
Set the CSP used by the CLR when strong-naming assemblies	174
storeadm.exe – .NET Framework Isolated Storage Tool.....	174
Syntax.....	174
Tasks.....	174
List all local or roaming data stores associated with the current user.....	175
Remove all local or roaming data stores associated with the current user.....	175
Summary.....	175
Recommendations in This Section	175
mscorcfg.msc – The .NET Framework Configuration Tool.....	177

	Assembly Cache.....	179
	Assembly Cache Tasks	179
	View cache contents.....	179
	Add an assembly to the GAC	181
	Delete an assembly from the GAC.....	181
	View properties of an assembly installed in the GAC.....	182
	Configured Assemblies.....	182
	Configured Assemblies Tasks.....	183
	View the list of configured assemblies.....	183
	Configure an assembly	183
	<i>Using the Properties Dialog Box</i>	185
	Delete the configuration information for an assembly	186
	Remoting Services	187
	<i>Terminology</i>	187
	Remoting Services Tasks	188
	Remoting Services Configuration Files	188
	The <application> element.....	190
	Well-known (Service-activated) objects	191
	Client-activated objects	191
	The <client> element	191
	The <service> element.....	193
	Remoting Services Configuration Example 1	194
	Remoting Services Configuration Example 2	195
	<i>Service</i>	195
	<i>Client</i>	195
	Remoting Services Configuration Example 3	196
	<i>Service</i>	196
	<i>Client</i>	196
	The <channel> element.....	196
	The <clientProviders> and <serverProviders> elements.....	199
	The <provider> and <formatter> elements	200
	<i>includeVersions</i>	201
	<i>strictBinding</i>	201

<i>typeFilterLevel</i>	201
Default channels in the Remoting Services configuration	202
The <lifetime> element	203
leaseTime	203
leaseManagerPollTime	203
renewOnCallTime	204
sponsorshipTimeout	204
The <soapInterop> element	204
interopXmlElement	205
interopXmlType	205
preLoad	205
 Runtime Security Policy	206
Runtime Security Policy tasks	207
Create a CAS policy deployment package	208
Reset all CAS policy levels to default settings	209
View Code Groups	209
Add or remove a Code Group	209
<i>Interactively defining a new Code Group</i>	210
<i>Importing a Code Group definition from an XML file</i>	211
Rename a Code Group	212
Set or clear the Exclusive or Level Final attribute of a Code Group	212
Change a Code Group's Membership Condition	214
Change a Code Group's associated Named Permission Set	214
Adjust Zone Security	215
View Named Permission Sets	217
Add or remove a Named Permission Set	217
<i>Interactively defining a new Named Permission Set</i>	218
<i>Importing a Named Permission Set definition from an XML file</i>	220
Modify a Named Permission Set	221
View Policy Assemblies	222
Enroll or withdraw a Policy Assembly	222
List Code Groups to which an assembly belongs	223
View an assembly's Allowed Permission Set	224
Create a tailored Code Group	224

Use the Trust an Assembly Wizard	225
 Applications	227
Applications Tasks	227
Add an application to be configured	228
Configure application properties	228
<i>Garbage Collection Concurrency</i>	229
<i>Publisher Policy Safe Mode</i>	229
<i>Probing Path</i>	229
View assembly dependencies for an application	230
View list of assemblies configured for an application	230
Configure an assembly for an application	231
<i>Using the Properties Dialog Box</i>	232
Fix an application (roll back application Binding Policy)	233
Configure Remoting Services for an application	235
Summary	236
Recommendations in This Section	236
Summary of Recommendations and Checklist	237
Summary of Recommendations	237
CAS Policy Checklist	242
First Steps to Configuring CAS Policy	243
Know how you will support the policy	243
Know what the policy needs to do for you	244
<i>Know the code</i>	244
<i>Know the environment</i>	245
<i>Know the resources</i>	245
CAS Policy Creation	246
General Guidelines	246
Policy Refinement Process	246
CAS Policy Review	248
Host Policy Review	248
Security Architecture Review	249
Summary	249
Works Cited	251

Table of Figures

Figure 1. Components of .NET Framework Execution on a Windows Platform.....	2
Figure 2. Composite Policy Enforcement Using the .NET Framework.....	6
Figure 3. Evidence-Based Access Control.....	10
Figure 4. Local Assembly Data Stores.....	17
Figure 5. Local Application Data Stores.....	18
Figure 6. Local Data Stores for a Managed Application that Directly Uses Isolated Storage.	18
Figure 7. Roaming Assembly and Application Data Stores.....	19
Figure 8. XML Structure of a Database Permission in Version 1.0.....	26
Figure 9. XML Structure of a Database Permission in Version 1.1.....	27
Figure 10. XML Structure of an OLE DB Data Provider <keyword> Element.....	31
Figure 11. Properties Button in perfmon.exe Window.....	43
Figure 12. System Monitor Properties Dialog Box.....	43
Figure 13. Add Counters Dialog Box.....	44
Figure 14. Default Code Groups at the Machine Level in Version 1.1 of the .NET Framework.....	58
Figure 15. Custom Code Group Tab for Intranet_Same_Site_Access.....	60
Figure 16. Create Code Group Dialog Box.....	61
Figure 17. Assigning Assembly Permissions.....	65
Figure 18. Creation of a Level Permission Set.....	66
Figure 19. Creation of the Allowed Permission Set.....	67
Figure 20. The Stack Walk.....	71
Figure 21. Deployment Package Wizard.....	75
Figure 22. Active Directory Users and Computers Microsoft Management Console Snap-in.	77
Figure 23. Group Policy Objects Linked to a Container.....	77
Figure 24. Displaying GPOs Linked to an Organizational Unit with the Group Policy Management Console.....	78
Figure 25. Viewing Software Installation Packages in the Group Policy Object Editor.....	79
Figure 26. Deploy Software Dialog Box.....	79

Figure 27. Displaying the Order of Precedence of GPOs Linked to an Organizational Unit with the Group Policy Management Console.	83
Figure 28. Custom Software Installation Item Names for CAS Policy Installer Packages. ..	84
Figure 29. Setting Loopback Processing Mode.	86
Figure 30. Forcing Software Installation for Unchanged GPOs.	87
Figure 31. XML Structure of the Cryptography Settings in <code>machine.config</code>	107
Figure 32. XML Structure of the <code><cryptographySettings></code> Element.	107
Figure 33. Cryptographic Localization Example 1.	112
Figure 34. Cryptographic Localization Example 2-A.	112
Figure 35. Cryptographic Localization Example 2-B.	113
Figure 36. Cryptographic Localization Example 3.	113
Figure 37. Relationship between validation and verification.	131
Figure 38. Viewing the GAC and Zap Cache in Windows Explorer using the Assembly Cache Viewer.	154
Figure 39. Assembly Properties Dialog Box.	155
Figure 40. Cache Properties Dialog Box.	157
Figure 41. The .NET Framework Configuration Console (<code>mscorcfg.msc</code>).	177
Figure 42. GAC Display in <code>mscorcfg.msc</code>	181
Figure 43. Adding a Configured Assembly.	184
Figure 44. Assembly Properties Dialog Box – General Tab.	185
Figure 45. Assembly Properties Dialog Box – Binding Policy Tab.	185
Figure 46. Assembly Properties Dialog Box – Codebases Tab.	186
Figure 47. XML Structure of the Remoting Services Configuration.	189
Figure 48. XML Structure of the <code><application></code> Element.	191
Figure 49. XML Structure of the <code><client></code> Element.	192
Figure 50. Remoting Services Properties Dialog Box – Remote Applications Tab.	193
Figure 51. XML Structure of the <code><service></code> Element.	193
Figure 52. Remoting Services Properties Dialog Box – Exposed Types Tab.	194
Figure 53. Remoting Services Configuration Example 2.	195
Figure 54. Remoting Services Configuration Example 3.	196
Figure 55. Remoting Services Properties Dialog Box – Channels Tab.	198
Figure 56. XML Structure of the <code><channel></code> Element.	198
Figure 57. XML Structure of the <code><provider></code> and <code><formatter></code> Elements.	200

Figure 58. Default Remoting Services Channels.....	202
Figure 59. Remoting Services Properties Dialog Box – Channels Tab.....	203
Figure 60. XML Structure of the <lifetime> Element.....	203
Figure 61. XML Structure of the <soapInterop> Element.....	205
Figure 62. Runtime Security Policy Node.....	207
Figure 63. Create Code Group Dialog Box.....	210
Figure 64. Example of a Membership Condition To Be Imported.....	210
Figure 65. Example of a Code Group To Be Imported.....	211
Figure 66. Code Group Properties Dialog Box: General Tab.....	213
Figure 67. Code Group Properties Dialog Box: Membership Condition Tab.....	214
Figure 68. Code Group Properties Dialog Box: Permission Set Tab.....	215
Figure 69. Security Adjustment Wizard.....	216
Figure 70. Adjusting the Security Level for Each Zone.....	217
Figure 71. Create Permission Set Dialog Box.....	218
Figure 72. Assign Permissions to a Named Permission Set.....	219
Figure 73. Example of a Permission To Be Imported.....	220
Figure 74. Example of a Named Permission Set To Be Imported.....	221
Figure 75. Evaluate an Assembly Dialog Box.....	223
Figure 76. Trust an Assembly Wizard.....	226
Figure 77. Membership Condition Selection in the Trust an Assembly Wizard.....	226
Figure 78. Named Permission Set Selection in the Trust an Assembly Wizard.....	227
Figure 79. Application Properties Dialog Box.....	229
Figure 80. Configure an Assembly for an Application Dialog Box.....	231
Figure 81. Enable Publisher Policy Check Box.....	233
Figure 82. ConfigWizards Dialog Box.....	233
Figure 83. .NET Application Restore Dialog Box – Managed Applications.....	234
Figure 84. .NET Application Restore Dialog Box – Restore Points.....	234
Figure 85. Example Relationship Between Functions and Resources.....	247

Table of Tables

Table 1. Combined Effect of <add> Element Attributes.....	30
Table 2. Summary of Trust Requirements for Data Providers.....	33
Table 3. Information Protected by the Environment Permission.....	47
Table 4. Default Named Permission Sets in the .NET Framework Version 1.1.....	64
Table 5. .NET Framework Zone Membership Conditions and URL Security Zones.....	91
Table 6. Default Code Groups and Named Permission Sets for Zone Membership Conditions.....	92
Table 7. Default Permissions for Named Permission Sets Associated with Zone Code Groups.....	92
Table 8. URL Security Zone Settings That Impact Managed Code Execution.....	93
Table 9. URL Security Zone Registry Keys.....	95
Table 10. Types of cryptographic services in the .NET Framework version 1.1.....	98
Table 11. Recommended default cryptographic algorithms.....	99
Table 12. Named cryptographic services and their default behavior in the .NET Framework version 1.1.....	105
Table 13. Default Friendly Names for OIDs in the .NET Framework version 1.1.....	110
Table 14. Security-Related Administrative Tasks.....	117
Table 15. Administrative Tools.....	118
Table 16. Discriminating Power of Membership Conditions.....	126
Table 17. Software Publishing State Settings.....	126
Table 18. <code>caspol.exe</code> Membership Condition Arguments.....	145
Table 19. Assembly Properties Viewable Through the Assembly Cache Viewer.....	156
Table 20. XML Files for .NET Framework CAS Policy.....	157
Table 21. <code>gacutil.exe</code> Reference Data Values.....	160
Table 22. <code>setreg.exe</code> Command Examples.....	171
Table 23. Formatter Sink Attributes.....	201
Table 24. CAS Policy File Locations.....	206
Table 25. Named Permission Sets Associated with “Levels of Trust.”.....	217
Table 26. Summary of Recommendations.....	242

Introduction

Purpose

The purpose of this document is to inform administrators responsible for systems and network security about the configurable security features available in the .NET Framework. To place some of the configuration options in context, a brief introduction to the .NET Framework security model and its components is provided. For further information about security in the .NET Framework, many resources are available; for example, see [Microsoft, MSDN], [Microsoft, .NET Framework], [LaMacchia, et al., 2002], or [Watkins and Lange, 2002].

The security features of the .NET Framework are highly extensible and configurable. While this document describes some of the default settings, it cannot address all possible circumstances or scenarios administrators may experience. This guide is intended to assist the administrator in exercising discriminating judgment in the configuration of the .NET Framework in response to variations in organizational security policies and operational environments.

This guide does not address Microsoft Windows operating system security issues that are not specifically related to the .NET Framework.

Assumptions

The following assumptions have been made about the environment in which the recommendations included in this guide will be implemented.

- The network consists only of computers running Microsoft Windows 2000 Server, Windows 2000 Professional, Windows XP Professional, and Windows Server 2003.
- All disk volumes are formatted using NTFS (NT File System).
- Domain Controllers are Windows 2000 Server or Windows Server 2003 running Active Directory.
- The latest Windows service packs and hot fixes have been installed.
- All operating systems are configured using their respective default NSA security templates. NSA security templates are available at <http://www.nsa.gov>.
- Users of this guide are familiar with the Windows 2000 and Windows XP operating systems, and have system administration skills sufficient to manage host-based and network security.

Conventions Used In This Guide

- URLs, UNC paths, file names, file system paths, text and XML examples are presented in a fixed-pitch typeface for enhanced readability. Examples:

```
http://www.nsa.gov
```

```
\\server1\myshare
```

```
c:\winnt\microsoft.net\Framework\v1.1.4322\mscorlib.dll.
```

- When a URL, UNC path, file name, file system path, text or XML example is too long to fit on a single line, the symbol (↵) will be used at the end of a line to indicate that a line break has been inserted to accommodate the text to the page width, but the next line is actually a continuation of the current line. Example:

```
http://msdn.microsoft.com/library/default.asp?url=/library/↵  
en-us/dnnetsec/html/netframesecover.asp
```

- When examples include placeholders for user-supplied text, the placeholder will be surrounded by braces. For example, in the following XML, both the label and value of an XML attribute are supplied by the user. “short name” and “fully-qualified name” are descriptive placeholders that should be replaced by the user with context-dependent values.

```
<cryptoClass {short name}="{fully-qualified name}"/>
```

Document Summary

This document contains the following chapters:

- Chapter 1, .NET Framework Overview, provides a high level description of the .NET Framework.
- Chapter 2, Features of the .NET Framework Security Model, provides a description of the components of the .NET Framework access control system, and how the administratively configured policy gets applied and enforced.
- Chapter 3, Deploying .NET Framework CAS Policy Using Group Policy, discusses network deployment issues and explains how to use Group Policy to deploy policy configuration files.
- Chapter 4, URL Security Zones and the .NET Framework Zone Membership Condition, discusses the interaction between the .NET Framework and the Windows URL Security Zones settings.
- Chapter 5, Cryptographic Localization in the .NET Framework, provides a discussion of the configurable settings for the use of cryptographic algorithms in managed code.

- Chapter 6, Tasks and Tools, describes common .NET Framework administrative tasks.
- Appendix A, Administrative Tools Reference, describes various tools and how to use them to perform common administrative tasks.
- Appendix B, `mscorcfg.msc` – The .NET Framework Configuration Tool, describes the most full-featured administrative tool, `mscorcfg.msc`, and how to use it to perform common administrative tasks.
- Appendix C, Summary of Recommendations and Checklist, lists all the recommendations contained in this document. A checklist is also provided that lists steps that should be performed when administering the .NET Framework as a security control.
- Appendix D, Works Cited, contains bibliographic entries for works cited in this document.

This page has been intentionally left blank.

.NET Framework Overview

What Is the .NET Framework?

The .NET Framework is a virtual execution system that supports platform independent, distributed applications. From a security perspective, it provides isolation for executing software to protect code from corruption and offers granular access control to resources to protect information from exploitation by local and distributed applications.

The .NET Framework is Microsoft's implementation of Standard ECMA-335, Common Language Infrastructure (CLI) [ECMA-335, 2002]. Any CLI implementation will support the creation and execution of platform-independent code; however, the .NET Framework leverages the diverse set of operating system and network resources available on Windows platforms to support the development of applications that take advantage of the security features of a CLI and are also fully integrated into the Windows environment.

The .NET Framework is implemented in the context of an operating system process, and includes the Common Language Runtime (CLR) and supporting software libraries. A Windows process that loads the CLR and creates the execution environment in which .NET applications run is called a Runtime Host. Within the Runtime Host process, the CLR and its libraries work together in conjunction with administratively configured policy data to provide isolated execution environments called application domains which manage and execute platform-independent code compiled with a .NET Framework-enabled compiler. Code management includes security policy enforcement, resource allocation, and tightly controlled separation of resource usage to prevent inadvertent or malicious data corruption. Programs compiled into platform-independent code that is subject to the constraints of the .NET Framework are called managed code and are implemented in specially formatted executable files called assemblies. Some of the internals of the CLR itself are managed code and most of the interaction with platform-specific system resources is provided through a set of managed software libraries distributed with the .NET Framework that developers can link to their own programs.

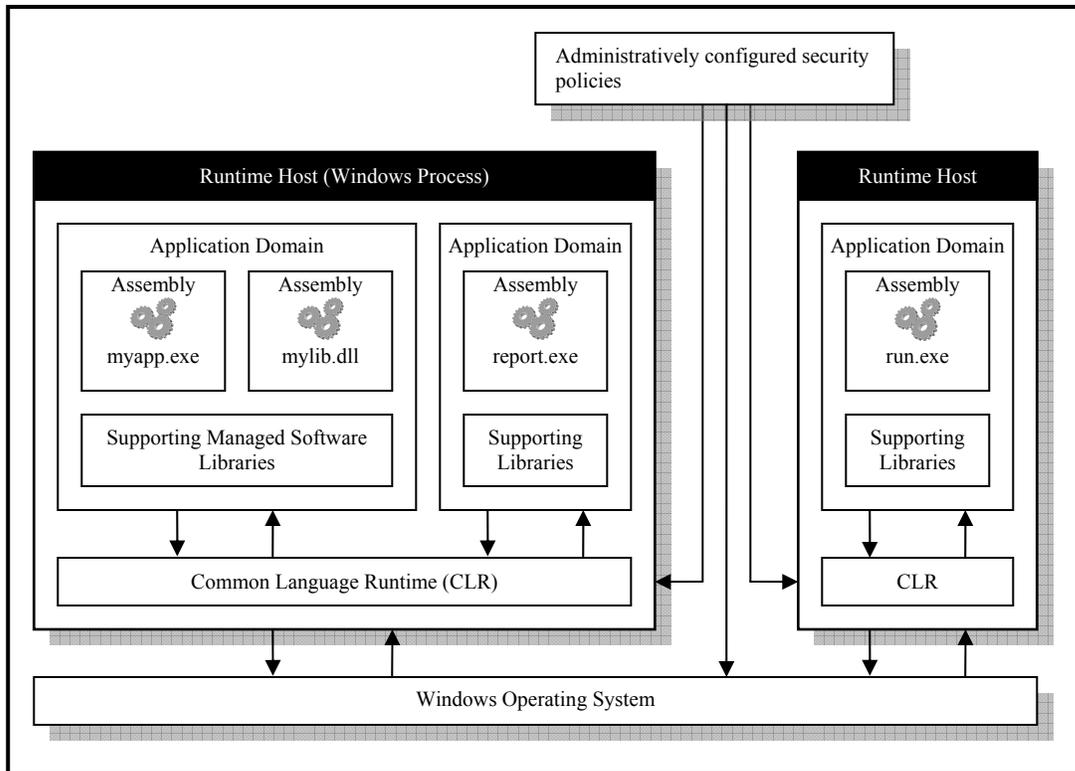


Figure 1. Components of .NET Framework Execution on a Windows Platform.

Security in this architecture is provided by the operating system and the CLR enforcing a composite security policy. Although the execution environment of the .NET Framework is layered on top of the operating system security mechanisms, it is administered separately. Figure 1 illustrates the security components used in the .NET Framework. Separation is enforced by the operating system at the level of a Windows process and by the .NET Framework at the level of an application domain. Access control is enforced by the CLR and, ultimately, by the operating system, in keeping with each layer's security policies and configurations.

The remainder of this chapter will introduce the features of the .NET Framework security model and how they are designed to work together to create a configurable execution environment that provides support for key security principles.

Security Principles and the .NET Framework

When deploying an IT system, system designers and administrators generally consider a number of factors to ensure smooth operation. These typically include functional and information flow requirements, ease of use and scalability requirements, security requirements, and cost. When considering security requirements, designers and administrators should understand the composite security policy being enforced by the whole system as well as the role each individual component plays in that overall enforcement. Numerous guidelines offer insight on how to implement general security planning and

address general security architectural issues. See, for example, [BS/ISO/IEC 17799, 2000], [Fraser (RFC 2196), 1997], [GAO/AIMD-12.19.6, 1999], [GASSP, 1999], [Swanson and Guttman (NIST SP 800-14), 1996], [Swanson (NIST SP 800-18), 1998], and [Weise and Martin, 2001].

Developers and administrators can use the security features of the .NET Framework to support security principles such as Accountability, Availability, Confidentiality, and Integrity. However, the .NET Framework does not explicitly support all these properties equally and it must be carefully integrated with the underlying operating system security policy to ensure that a sound composite policy is implemented. The rest of this section describes these various policies at a high level and offers some insight on the role the .NET Framework could play in a composite architecture and what the administrator should consider in the context of its integration.

Accountability

Accountability is a property that enables activities on a system or network to be traced to individuals or entities that may then be held responsible for their actions. An important goal for a composite accountability policy is for the administrator to be able to review a security log, discover actions that took place in a given time frame, and identify the individual or entities responsible for those actions. Traditionally, achieving accountability requires four components: identification and authentication (I/A), auditing, and non-repudiation.

The .NET Framework execution environment itself does not implement user I/A, user auditing, or user non-repudiation. Users or code do not have to “log on” to the .NET Framework. However, the .NET Framework provides supporting libraries that applications can use to implement their own I/A, audit, and non-repudiation mechanisms and use the built-in capabilities of the Windows operating system.

In addition, the access control system of the .NET Framework is based on the classification of code in terms of characteristics (“evidence”) that form a type of credential that managed code presents before being granted access to resources. In some cases, these credentials strongly identify a particular assembly (cryptographic hash) or the software developer who created the assembly (software publisher’s digital certificate). Unfortunately, there is no robust built-in auditing capability that would record noteworthy events and the associated code identity information.

Availability

Availability is a property that ensures that resources and services are available for use when they are needed. Availability is typically achieved through redundancy with isolation, quotas on resource usage, or finally on fault-tolerant code or equipment.

The availability of the .NET Framework itself is based on built-in Windows separation features. Since the .NET Framework is loaded into a Windows process, the Windows protected mode of execution ensures that one instance of the execution environment may fail without affecting other instances.

The availability of information and services provided by managed code that runs in the context of the .NET Framework is facilitated in three ways. First, the .NET Framework enables a distributed application architecture which could be used to implement redundancy with respect to application components. Second, the stability and fault tolerance of managed code execution is enhanced by the .NET Framework's strict control of resource usage and support for exception handling. Finally, the access control mechanism can be used to deny resources to some managed code. Unlike a resource quota mechanism, which supports availability by constraining code to an appropriate use of resources, a fine-grained access control mechanism supports availability by constraining code to the use of appropriate resources.

Confidentiality

Confidentiality is a property that offers assurance that information is shared only among authorized entities. It typically consists of I/A, access control, and cryptography.

I/A and access control in the .NET Framework are discussed under Availability above, but have some further implications for confidentiality. Confidentiality is served by an access control mechanism that protects file contents and properties, configuration data, sensitive system information, direct access to memory contents, and other resources from unauthorized disclosure. The .NET Framework's evidence-based access control is a departure from general purpose operating system controls in that authorization is not necessarily tied to an authenticated identity or role, but is based on one or more indicators of trustworthiness. The administrator classifies code in advance based on types of evidence (defines Code Groups), and authorizes access to resources based on those classifications (associates to each Code Group a Named Permission Set that lists resource access permissions). The CLR then enforces this policy for each assembly while it is executing. This mechanism is called Code Access Security (CAS) and will be discussed in more detail in chapter 2. Note that access to resources will ultimately be subject to the access control mechanism of the Windows operating system, which is based on the identity of the user account associated with the Windows process hosting the .NET Framework.

The .NET Framework does not use cryptography to protect its internal data, but relies on its code separation enforcement to prevent managed code from accessing or modifying the state of the execution environment itself in unauthorized ways. It does provide cryptographic services to assemblies through its managed libraries. These services are fully extensible and configurable by an administrator if additional or substitute cryptographic modules are required. For protecting U.S. Government information, only approved cryptographic modules should be configured. Appropriate government authorities should be consulted for guidance in cryptographic module deployment. The National Institute of Standards and Technology is responsible for cryptographic policy for all Federal information systems except national security systems. The National Security Agency is responsible for cryptographic policy for systems processing national security information. See [FISMA, 2002] and [Barker (NIST SP 800-59), 2003] for more details. Appendix D contains full references to works cited in this document.

Integrity

Integrity is a property that offers assurance that information is accurate, reliable, and consistent. To maintain information integrity, a system must prevent, detect, and/or correct the unauthorized or inadvertent modification of data and executables. The code separation mechanism of the .NET Framework prevents one assembly from directly modifying another assembly's internal state. In addition, cryptographic hashes and digital signatures are an integral part of the .NET Framework's CAS system. Hash values show whether an assembly has been corrupted or modified since it was created. Digital signatures may also identify the creator of the assembly. The CLR relies on these technologies to determine whether an assembly is allowed to execute and what resources it may access. This protection extends to all the managed software components of the .NET Framework itself. Managed code can use these services as well to maintain the integrity of their own data over time.

The .NET Framework Security Model

This section, which introduces the security model and its components, will lay the groundwork for the discussion of the configuration and management of the security properties of the .NET Framework in later chapters. The security model employed by the .NET Framework has some significant differences from traditional operating system security. In traditional operating system security models, programs run in the context of a user account, and are authorized to access resources based on the access rights defined for that user account. When a user account is granted access to a resource, that authorization is extended to all software the user executes. The basic premise of such a model is that the security mechanisms exist to protect the system from discretionary actions by users. One side effect of such an environment is that malicious code can cause greater damage when executed by an administrator than when executed by a non-administrator, since all software requests for resources are assumed to be at the discretion and authorization of the user.

The .NET Framework security model allows permissions to be granted to executable code based on characteristics of the code itself. This is achieved by the CLR working to enforce permissions granted by an administratively configured CAS policy. These permissions or their absence will be enforced regardless of the user account in which the code is running. For example, if an application is not granted permission to write to a file by the .NET Framework CAS policy, the application will be unable to write to the file even if run by a user who has Windows write access to the file. Conversely, if an application is granted permission to access a resource by CAS policy, but the user who runs the code is denied access by Windows security, any attempted access will fail. The basic premise of the .NET Framework security model is that the security mechanisms exist to protect the user from discretionary actions by software.

Figure 2 illustrates this concept. A user executes the managed application `myapp.exe` in the context of an operating system process. When `myapp.exe` attempts to access a protected resource, the CLR checks the access permissions of `myapp.exe` based on the .NET Framework CAS policy (steps 1 and 2). Since the assembly is granted access to the resource

by CAS policy, the CLR passes the access request through to the Windows operating system. The operating system then checks the access permissions of the user based on the security credentials associated with the Windows process that is hosting the managed code (steps 3 and 4). Since the user does not have permission to access the resource, access is denied.

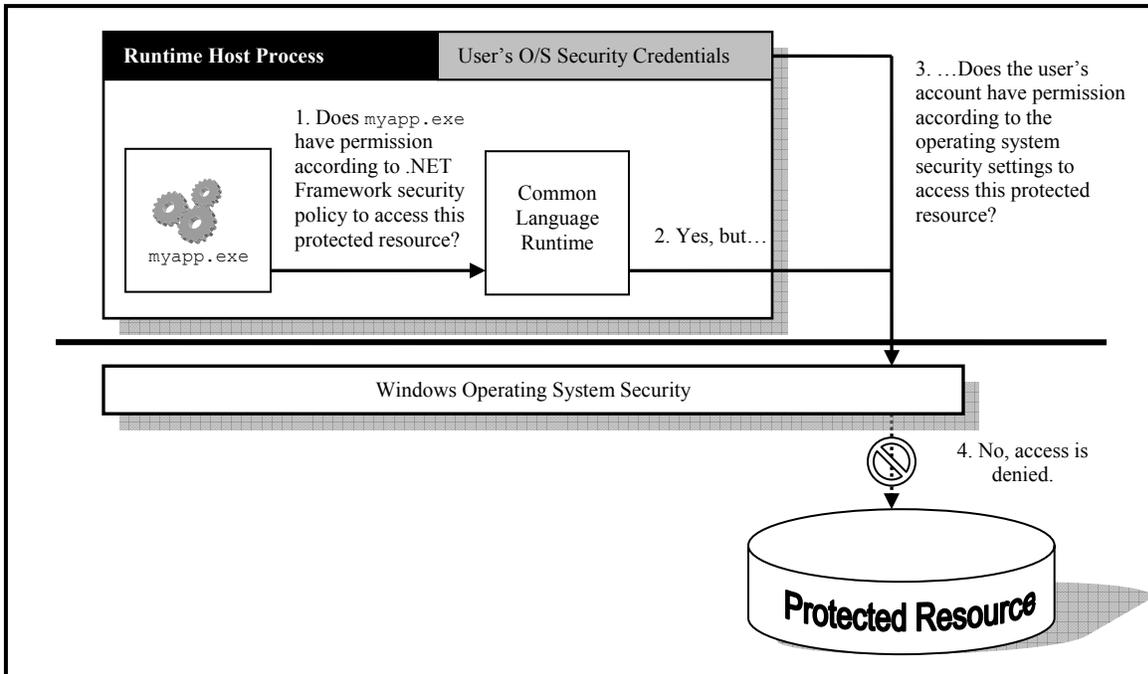


Figure 2. Composite Policy Enforcement Using the .NET Framework.

.NET Framework Security Model Components

The .NET Framework security model consists of the following components:

- Assemblies (managed executables). Assemblies are the unit of access control. A .NET Framework application may include a number of different assemblies, each of which may be granted different access rights.
- Protected resources that assemblies may want to use. These predefined sets of resources determine the access rights that may be granted to an assembly.
- An administratively controlled Code Access Security (CAS) policy. This policy defines administratively-created rules that determine which assemblies or groups of assemblies have which sets of access permissions. Additional fixed rules determine how permissions are combined when an assembly is a member of multiple groups.
- An access monitor internal to the CLR. This CLR component serves as an oracle that software can query to determine access rights to resources. The access monitor determines access rights by comparing the requested access type and resource identity against the permissions associated with an assembly.

CAS policy is applied whenever an assembly is loaded to associate access permissions with the assembly. During execution, the CLR's access monitor can perform access rights checks on behalf of executing code. Note that software that desires to use a resource can decide not to check access rights through the access monitor, and if it does query for access rights, can act on the result of the query, or choose to ignore it. Security in such a scheme is provided by mediated access to resources and "walking" the call stack.

Mediated Access to Resources

The CLR's software libraries function as highly privileged components of the .NET Framework's security mechanisms. While user code can avoid or ignore the access monitor, any attempt to access resources must pass through the supporting libraries, which are trusted (based on trust in Microsoft's software development and distribution processes) to query for the appropriate access rights and to not disregard the query results when attempting to use resources.

Note that an administrator may grant an assembly the right to directly access resources by invoking native code that is not subject to CAS policy. In such a case, the assembly is being treated as a privileged extension to the CLR.

Walking the Call Stack

When the access rights of software are checked, all of the assemblies in the chain of execution, starting from the initial application to the assembly performing the access rights query, are checked for the appropriate access permission. This process is known as walking the call stack. If any assembly in the chain is not authorized to access the requested resource, the access check will fail. Privileged code such as the CLR's libraries will be granted access to all resources, but they will not be able to access resources on behalf of less privileged code.

In many cases, the CLR will need to access protected resources to store internal state information or perform its functions, but these resources are not exposed to other code. The CLR's libraries can make a special request for access to these resources even though less trusted code is part of the call stack. This bypassing of the stack walk is a privileged operation that is typically granted only to CLR libraries or other code of trusted origin.

The next chapter will discuss the association of permissions with code by introducing some of the details of the assemblies, resources, and policy components of the .NET Framework security model.

This page has been intentionally left blank.

Features of the .NET Framework Security Model

The administratively configurable features of the .NET Framework security model consist essentially of three components: assemblies, protected resources, and CAS policy that defines access control rules by which assemblies may be granted or denied access to those resources. The access control rules defined by the administrator in the CAS policy are based on the nature of the evidence presented by code and the nature of the access and resource types available through the CLR libraries. Therefore the following discussion of assemblies and protected resources is an important context for the final discussion of CAS policy.

Assemblies

An assembly is the basic unit of software deployment. A managed application executes in the context of a Windows process and may consist of multiple assemblies; an assembly, in turn, may consist of multiple executable files that have been linked together into a self-describing logical collection. The self-description information is called the assembly metadata and is contained in an assembly manifest.

The manifest contains references to all of the executable files and resources that constitute the assembly. In addition, each executable file in an assembly has a manifest that contains information that allows it to interact with other assembly components without relying on external configuration or installation data such as registry keys.

Assemblies are also the basic unit of software access control. CAS policy rules are applied to assemblies or groups of assemblies rather than to individual executable files (even when the assembly consists of exactly one executable file component).

Assemblies may be private or shared. A private assembly is deployed in the same directory as the application that uses it, is visible only to that application, and cannot be accessed by any other application or assembly outside its directory. A shared assembly is structurally identical to a private assembly, but may be used by other applications executing on the same computer. A shared assembly is exposed for general use by installation into a machine-wide repository called the Global Assembly Cache (GAC).

Evidence-Based Access Control

In much the same way that users must present evidence (i.e., identification and authentication credentials) to the operating system before they are granted access to system resources, assemblies present their attributes as evidence to the CLR so that it can have a basis upon which to determine assembly access to protected resources. In addition, the application domain into which the .NET Framework loads the assembly contributes some evidence about where and how the assembly was obtained. When an assembly is loaded for execution, the CLR assigns permissions to assemblies by weighing the presented evidence against an administratively defined CAS policy. These assigned permissions are then checked during code execution by the CLR's access monitor to resolve all queries for access rights to desired resources.

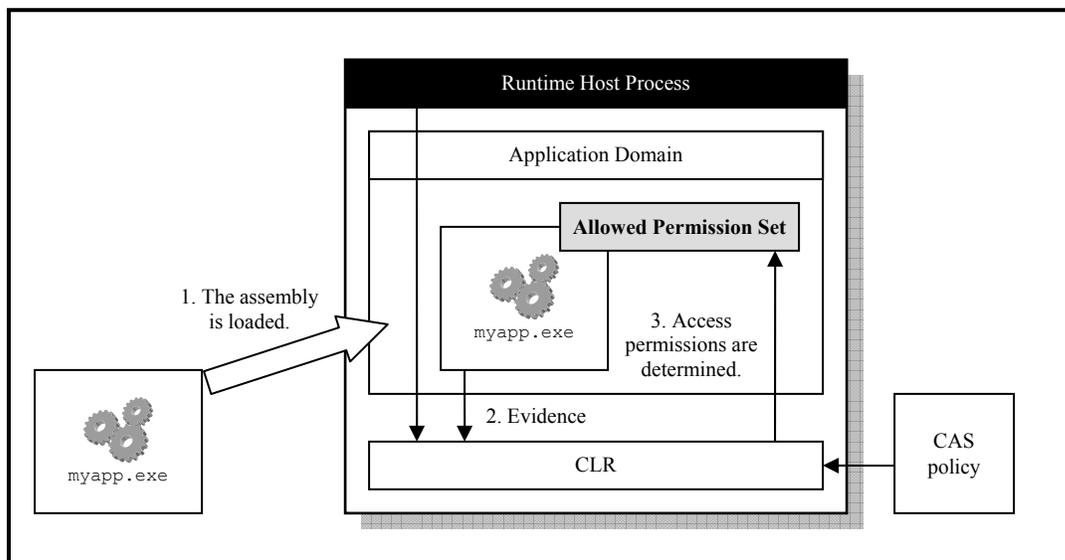


Figure 3. Evidence-Based Access Control.

Figure 3 illustrates the association of permissions with an assembly at load-time based on evidence. When the assembly is loaded by the CLR into an application domain (step 1), both the Runtime Host and the assembly itself present evidence to the CLR (step 2) that describes the origin and content of the assembly. From the CAS policy, the CLR determines the membership of the assembly in evidence-based groups and the sets of access permissions granted to group members (step 3). These permissions are then associated with the assembly for later use when checking access rights to specific protected resources.

Types of Assembly Evidence

Evidence used by the .NET Framework can be intrinsic or location evidence.

Intrinsic Assembly Evidence

Intrinsic evidence consists of properties that describe the assembly itself.

UNCLASSIFIED

- Hash: A cryptographic hash of an assembly. CAS policy supports both the MD5 and SHA-1 hash algorithms. An assembly can always present hash evidence, as a new hash value can be computed at will by the CLR.
- Publisher: The (successfully validated and verified) Authenticode digital signature of the assembly. This is a digital signature using a Software Publisher's digital certificate: it consists of a cryptographic hash of the assembly which is encoded using a vendor's private key. The corresponding public key, which is included in the assembly's manifest, is then used to decode the hash and check it against a newly computed hash of the assembly. If the encoded hash value and the newly computed hash value differ, then the assembly will not present any publisher evidence. In addition, if the digital certificate fails the Windows certificate verification process, no publisher evidence will be used.
- Strong Name: The strong name of an assembly is an extended cryptographic identity for a specific assembly distribution. It is designed to solve some of the conflict problems of referring to executables simply by filename. It precisely identifies an assembly down to the bit, so different release versions or even different language versions of the same assembly will have different strong names. The strong name consists of the assembly's filename, version, culture (language) information, a digital signature (encrypted hash value), and the public key corresponding to the private key used to digitally sign the assembly. The assembly will present strong name evidence if the digital signature is valid, i.e., the public key is used to decrypt the original hash value and it matches a new hash computation of the assembly.

Note that in order for a strong name to serve as a legitimate basis for granting access to resources, the public key must be associated with a known source. Simply having a valid digital signature does not imply that it was signed by a trustworthy party. Some method, such as a well-designed and maintained public key infrastructure, of associating the public key with a known source must be used to verify the assembly's origin. These methods are outside the scope of the .NET Framework execution environment.

Location Assembly Evidence

Location evidence consists of properties that describe how and from where the assembly was obtained. This evidence is provided to the CLR by the Runtime Host process.

- URL: The specific URL, UNC, or file system path from which the assembly originates. An assembly originating from `http://www.example.com/test/MyApp.exe` would present the full URL including the protocol (`http://`) as evidence. Assemblies referenced through UNC or file system paths would include the file protocol in their URL evidence, e.g., an assembly located at `c:\dev\myapp.exe` would present `file://c:\dev\myapp.exe` as URL evidence.

UNCLASSIFIED

- **Site:** The Website from which the assembly was obtained. Assemblies loaded from the local computer or the local intranet do not present site evidence. The Website is the domain name part of the URL. For example, an assembly originating from `http://research.example.com:8080/test` would present the site name `research.example.com` as evidence.

Note that the site evidence does not necessarily identify the developer of the code, as it may be obtained from a mirror site or other software distribution point. It merely identifies where the assembly was obtained this time.

- **Zone:** The Windows URL Security Zones (typically associated with Internet Explorer) including My Computer, Local Intranet, Trusted Sites, Internet, and Restricted Sites. Note that assemblies downloaded and cached by Internet Explorer are still associated with the Zone from which they were downloaded, even though they may be loaded from the cache on the local machine when executed at a later date.
- **Application Directory:** The base directory or URL associated with an application domain. This evidence is supplied by the application domain and can be used in conjunction with the assembly's URL evidence to define access control rules based on whether an assembly is considered a component of the current application or not (i.e., whether its URL evidence is a sub-path of the application domain's base directory or URL).

.NET Framework Protected Resources

The CLR provides access control over resources that software may need to use, but whose unrestricted use by malicious or unstable code may expose the user or system to risk. Permission by code to access these resources is provisionally granted based on the CAS policy defined using the administrative tools provided with the .NET Framework (i.e., `mscorcfg.msc` or `caspol.exe`). Permission is provisional because the CLR's access control mechanism is layered on top of the Windows operating system access control system. Access by managed code to a resource is subject to both CAS policy and the privilege level of the user account in which the managed code is executing. The effect of this layered approach is that code can be constrained on a case-by-case, user-independent basis using the fine-grained CAS policy resource permissions, allowing more code with richer features to be safely executed than ever before.

The resources that the .NET Framework security mechanisms protect fall into three basic categories:

- Protected Local Machine Resources
- Protected Network Resources
- Protected Administrative Resources

Protected Local Machine Resources are data or service resources available on the local machine. Protected Network Resources are those resources that are accessed on either local or remote hosts or provide networking services. Protected Administrative Resources are those resources that are used by programs to operate the system in a secure and predictable manner. These resources are typically managed by administrators responsible for maintaining the security and integrity of the system.

Each type of protected resource has an associated type of permission that can be granted to code. A permission generally consist of an identification of a specific instance of a protected resource and a type of allowed access to that resource. The way that resource instances are identified and the types of access that are possible vary from resource to resource. In addition to these resource-specific permission parameters, each type of protected resource also has an Unrestricted permission that allows full access to all instances of the resource. Many resources (for example, the file system) are also protected by operating system mechanisms, so permissions granted through CAS policy simply define the greatest possible access to resources available to code, that is, the access available when executing under the most privileged user account.

In addition to granting access to specific resources, the .NET Framework also supports a blanket grant of Unrestricted access to all protected resources. This is known as “Full Trust.” Any code that is not given this blanket permission is said to be “partially trusted,” even if the actual permissions granted consist of Unrestricted access to each individual permission.

Protected Local Machine Resources

Local Machine Resources are resources that are typically associated with a single machine:

- File System
- User Interface Elements
- Reflection

File System

The CLR offers assemblies persistent storage through use of the local file system. Access to persistent storage in a local file system is controlled through three basic permissions:

- File IO: permission to invoke general file system functions,
- File Dialog: permission to invoke file system dialog boxes,
- Isolated Storage File: permission to store data in a special file system-based virtual storage cache.

File IO

Assemblies may be granted access to named files, folders, or drives via the File IO permission. Access types include None, Read, Write, Append, Path Discovery, and AllAccess, which is equivalent to the combination of the Read, Write, Append, and Path Discovery access types to the specific named file system resource. In addition, the Unrestricted permission grants an assembly full access to all file system resources. Read access includes permission to determine whether a file exists and to read file metadata such as the file size, creation time, etc. Write access includes permission to delete or overwrite a file, but does not include permission to read from the file. Append access grants only the ability to add data to the end of a file – it also does not grant read access to the file. Path Discovery includes permission to determine the parent or the contents of a folder.

Permission to read or write to a file is checked only when the file access is first requested. Once a file has been opened by an assembly with Read access, the assembly may then allow other code to read from the file as well. Likewise, an assembly that has Write access to a file may allow code that doesn't have Write access to that file to write to it. Thus, granting Read, Write, or Append access to an assembly will potentially grant the same access to all other code with which the assembly can communicate. Granting File IO permission requires a degree of trust that the code will behave responsibly and not allow other code unauthorized access to file system resources.

In keeping with the principle of least privilege, File IO access should be granted only to the specific file system objects that are necessary for an assembly to function. The File Dialog permission (see below) may be used instead to grant access to a wide range of discretionary (user-specified) file system resources. The Isolated Storage File permission (see below) may be used instead to permit access to user or session-specific information such as customized settings or data needed to resume a previous state.



Recommendation: Only grant the File IO access permissions Read, Write, or Append to code that is trusted not to allow unauthorized access to file system resources. Grant File IO access to the most restrictive set of files and folders possible. Do not grant File IO access to file system roots or other broadly specified resources simply because they contain a few scattered files of interest. In many cases, the File Dialog or Isolated Storage File permissions are viable alternatives

File Dialog

Dialog boxes may be used by assemblies to obtain access to files and folders through interaction with a user. Since the file system resource is specified by a user, this is a safer way to provide file system access than by allowing direct access using the File IO permission. Moreover, the use of a file dialog box does not expose the identity (full path and file name) of the selected object(s) to the code invoking the dialog box, unless that code also has the AllAccess File IO permission to all the selected objects, so information about or derived from the structure of the file system cannot be obtained by partially trusted code.

Access through the File Dialog permission includes None, Open, Save, and OpenSave. By default, code from the Internet Zone has only Open permission, which allows read-only access to files specified by the user. Note that read access to the file specified by the dialog box is checked only when the file is opened. As with the File IO Read permission, any assembly that opens a file using a dialog box may allow other code to read from that file. Nevertheless, because code may not have access to the file identity, this is safer than the corresponding File IO permission, even though code must be trusted to behave responsibly.

Since the Save File dialog box does not allow append access, the File Dialog permission cannot be used by code that needs to incrementally write to a log file. In this case, a partially trusted assembly may either be granted File IO permission to a specific file, or the more restrictive Isolated Storage File permission (see below).



Recommendation: Grant the File Dialog permission to code that needs user-discretionary access to files and folders. Use the File Dialog permission to allow the user rather than partially trusted code to browse the file system to the desired items. Where Append access is necessary or direct file system access cannot be allowed, the Isolated Storage File permission may be a viable alternative.

Isolated Storage File

The Isolated Storage File facility provides a way for partially trusted assemblies to save information for later use without directly accessing local machine resources such as the file system or registry (similar to “cookies” used by Web applications). The “File” designation distinguishes this resource from other means of implementing Isolated Storage, for example, in an external storage network, in the registry, in a database, etc. Isolated Storage File is a protected resource in the .NET Framework that is implemented through a folder tree within each user’s local and/or roaming profile folder. There is no system-wide Isolated Storage File area. All Isolated Storage File data stores are associated with a user account and some form of identifying evidence (Publisher’s digital signature, Strong name, URL, Site, Zone) for an assembly and possibly also a managed application.

Access types for the Isolated Storage File permission include None, Assembly Isolation by User, Assembly Isolation by Roaming User, Domain Isolation by User, Domain Isolation by Roaming User, Administer Isolated Storage by User, and the Unrestricted permission.

When an assembly creates an Isolated Storage File data store, it is one of four types:

- A Local Assembly Data Store corresponding only to some identifying evidence of the assembly. This type of data store may be created by an assembly with the Assembly Isolation by User access type.
- A Local Application Data Store corresponding to some identifying evidence of the assembly and some identifying evidence of a managed application. This type of data store may be created by an assembly with the Domain Isolation by User access type.

- A Roaming Assembly Data Store corresponding only to some identifying evidence of the assembly. This type of data store may be created by an assembly with the Assembly Isolation by Roaming User access type.
- A Roaming Application Data Store corresponding to some identifying evidence of the assembly and some identifying evidence of a managed application. This type of data store may be created by an assembly with the Domain Isolation by Roaming User access type.

The Assembly Data Stores provide the least protection. A shared library that is granted this permission may read and write data to this storage area regardless of the application context in which it is executing. This permission is designed to allow access to data that is needed by many applications. Note that access will still be limited to the data stored by the shared library. The Application Data Stores provide greater protection. A shared library that is granted this permission may read and write data only to a storage area associated to this library running on behalf of a specific application. When the library is invoked by a different application, it will create a different Isolated Storage data store. This is useful when the assembly only processes data that is specific to an application. The Administer Isolated Storage by User permission allows browse access to all data stores created for the user in either the local or roaming profile, as well as read and write access to the assembly's own Assembly data stores. Browse access to a user's data stores permits the enumeration of the names of the directories and files contained in any data store for the current user.



Recommendation: Grant Administer Isolated Storage by User access only to highly trusted administrative tools. Grant Assembly Isolation by User/Roaming User access only to assemblies that need to use user-specific data applicable to many applications, and do not use application-specific data. Grant Domain Isolation by User/Roaming User access to all other assemblies. Note that this recommendation entails a separation of duties among assemblies: those that process data of common relevance to multiple applications should not also process application-specific data and vice versa.

In the current implementation of Isolated Storage File, the file system folders created by the Isolated Storage File libraries are often given names that are cryptographically derived from the appropriate identifying evidence of an assembly or managed application. This is a convenience that allows short (but cryptic) folder names to represent complex identity information. Because folders names are generated in this way, the Isolated Storage File file system should only be manipulated through the .NET Framework libraries (including the `storeadm.exe` tool, which uses the .NET Framework libraries). Attempting to directly modify the files and folders of the Isolated Storage File file system would probably lead to corrupt and unusable data. Moreover, the actual implementation of the Isolated Storage File data stores may change in future versions of the .NET Framework. Therefore, the discussion below of an assembly's available data stores is a simplified abstraction of the actual file system hierarchy employed by the .NET Framework libraries.

Local Assembly Data Stores

Local Assembly Data Stores (see Figure 4) are folders in the user’s local profile area that are each tied to some identifying evidence of an assembly. The Isolated Storage File permission “Assembly Isolation by User” is required to access this kind of data store. There will only be one data store of this type for a given form of identifying evidence, and all assemblies with the same evidence can share this data store. Thus, a strong-named assembly will use the same data store, whether it was loaded from the Global Assembly Cache on the local host or invoked through a Web URL. Two different assemblies (including assemblies with the same name but not strong-named) will only be able to share the data stores for identifying evidence that they have in common. An assembly loaded from the local host could create a data store based only on Zone evidence that would be shareable by any assembly in the MyComputer Zone. Similarly, an assembly executed from the Internet Zone with the required permission could store data in a Site-based data store that other such assemblies from the same Website could access.



Figure 4. Local Assembly Data Stores.

When the “Assembly Isolation by User” permission is granted to a library assembly, it can read information that it had previously stored, regardless of the managed application on whose behalf it had previously written the data. Thus, these data stores are useful for holding non-application-specific data that an assembly might need to handle for multiple applications. For example, an address book assembly might hold contact information here for a number of telecommunication applications, but not data for particular messages. Similarly, an auditing library might aggregate logging or performance data for multiple applications here, but not data for particular application events.

Local Application Data Stores

Local Application Data Stores (see Figure 5) are folders in the user’s local profile area that are each tied to some identifying evidence of an assembly and contained in a grouping tied to some identifying evidence of a managed application. The Isolated Storage File permission “Domain Isolation by User” is required to access this type of data store. These data stores are used to hold information that is specific to a managed application, even though it may be stored and retrieved through a shared library. An assembly may create multiple application-scope data stores, each contained in some managed application’s Isolated Storage File data store. Note that there is no application-wide data store; data stores must be tied to some

identity evidence of an assembly invoked by the application. If an application depends on two different managed libraries (say, `ShoppingBasketHelper.dll` and `WishListHelper.dll`) that both use Isolated Storage File by URL, there will be two application-scope data stores created, one by each library, and the libraries will not be able to access each other's data stores.

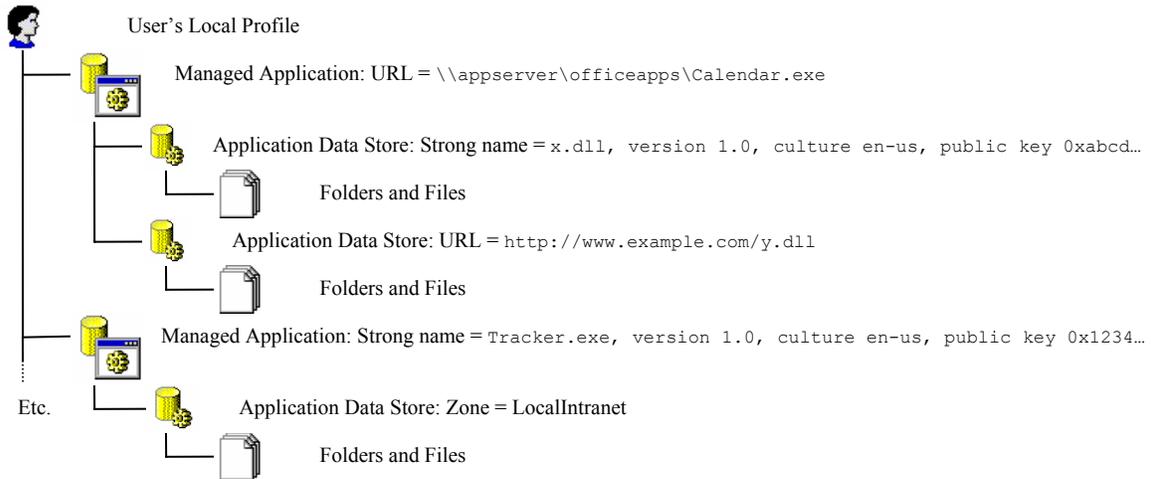


Figure 5. Local Application Data Stores.

Even when a managed application directly uses Isolated Storage File without a helper library, the assembly-scope and application-scope data stores are distinct. Figure 6 shows assembly and application-scope data stores for a managed application that directly accesses Isolated Storage. If the application is granted only the “Assembly Isolation by User” permission, only the assembly-scope data store will be accessible, and similarly for the “Domain Isolation by User” permission. If the application has unrestricted use of Isolated Storage or, equivalently, has the “Administer Isolated Storage by User” permission, both data stores will be accessible.

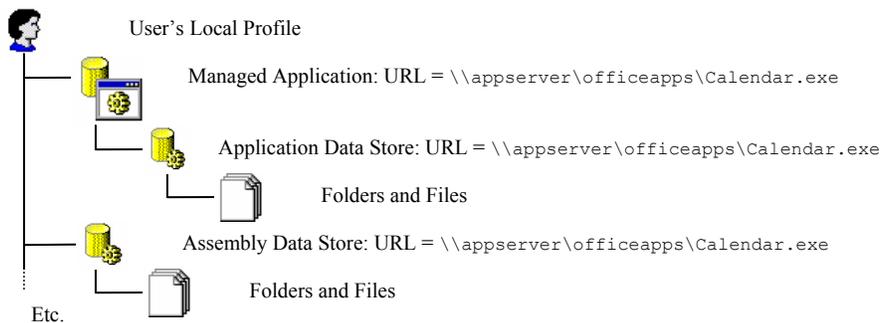


Figure 6. Local Data Stores for a Managed Application that Directly Uses Isolated Storage.

Roaming Assembly and Application Data Stores

The Roaming Data Stores (see Figure 7) differ from the Local Data Stores only in their location within the host or network. The Roaming Data Stores are located in the user's roaming profile, which is a combination of data in a central network location and a locally cached working copy which updates the remotely stored profile at logout. The Local Data Stores are only located in the user's profile directory on the local host. Local and Roaming Data Stores are completely independent of one another. The corresponding Isolated Storage File permissions that are required for access to the Roaming Data Stores are "Assembly Isolation by Roaming User" and "Domain Isolation by Roaming User." Note that there is no built-in facility to synchronize the Local and Roaming Data Stores.

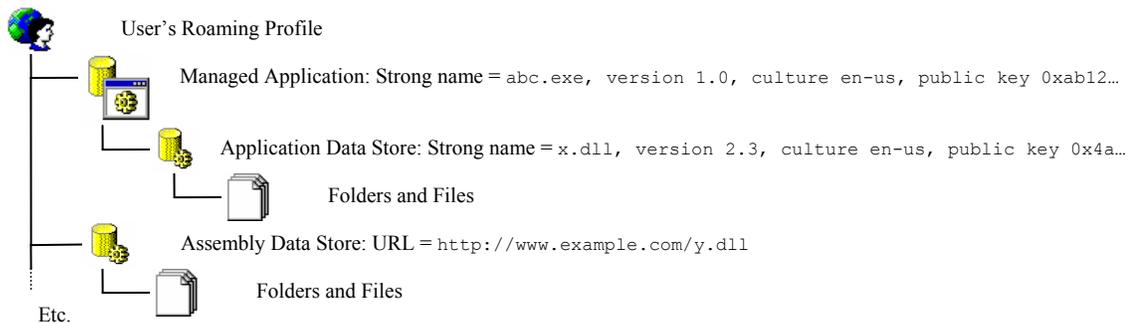


Figure 7. Roaming Assembly and Application Data Stores.

User Interface Elements

Users interact with applications through graphical interface elements that provide convenient and standardized means to supply input data or present output data. Because users may make security-relevant decisions or supply input data based on the state of the application as it is presented through the user interface elements, these items are protected by the CLR to prevent the user from being tricked into disclosing information. In addition, the transfer of data through the Windows system clipboards is protected. Access to these resources is controlled by two independent settings of the User Interface permission:

- **Windowing:** access to various types of window elements and associated events
- **Clipboard:** access to cut/paste operations between applications

Windowing

Control over windowing elements is granted using the four access types: No windows, Safe subwindows, Safe top-level windows, and All windows and events. A "safe" window has some unmodifiable features that prevent code from controlling the appearance of the window. A top-level window is a main window of an application. Its parent is the operating system desktop, and it may serve as a parent for other windows created by the application,

including message windows, dialog boxes, or forms. When an assembly runs in a browser window, all assembly windows are subwindows of the browser window.

The All windows and events permission allows an assembly to create and modify all aspects of windows that have been created by the assembly. This includes unrestricted access to all events and windows properties (such as location, shape, size, window titles, button-click events, etc.) for top-level windows and subwindows.

With the Safe top-level windows permission, all the windows associated with the application are “safe” windows as described above. The assembly may draw and have access to user input events within the windows, but is not able to change the visual layout of the window itself, such as its size, shape, opacity, and location.

The Safe subwindows permission further restricts window access, as an assembly with this permission may not have any access to top-level windows (including user input events), and is still prohibited from modifying the layout of subwindows.

The most restrictive access type is No Windows permission which denies the ability for an assembly to provide any windowing interface.



Recommendation: Code with limited trust should be granted at most Safe subwindows permission. Highly trusted code that accepts user authentication information or allows the user to authorize program actions through a graphical interface should be granted at most Safe top-level windows permission.

Clipboard

The Windows Clipboard is a familiar mechanism to users who routinely cut and paste information from one application to another. There are three types of clipboard access that can be granted by the .NET Administrator: No Clipboard, Own Clipboard and All Clipboard. The No Clipboard permission disallows assembly access to the system wide clipboard. The Own Clipboard permission allows an assembly to copy or cut information to the system wide clipboard, but does not allow the assembly to paste data from the clipboard except when invoked by Ctrl-v or similar user input. This permission prevents the unauthorized disclosure of clipboard data. The All Clipboard permission provides unrestricted access to the system clipboard, allowing the assembly potentially to exchange data with any other application on the local computer that also has clipboard access.



Recommendation: The clipboard is a convenience for users who wish to change the presentation context of data or reuse data without retyping it into another application. It is a “broadcast” channel in that most software can programmatically read the contents of the clipboard and write data to it whether initiated by user input or not. Nevertheless, software should use other means to communicate and reserve the clipboard for discretionary use by the user. Read access (i.e., through the All Clipboard permission) should be reserved for highly trusted code.

Reflection

Reflection refers to the capability of an assembly to discover information about itself or other assemblies while it is executing. Through reflection, an assembly can determine the best way to save its internal state for later resumption of processing, it can dynamically modify its behavior by detecting and responding to alternate forms of input data, and it can query the attributes and methods of other assemblies to find and invoke desired functions. The capability to use reflection is protected by three permissions:

- Type Information
- Member Access
- Reflection Emit

The Type Information permission allows an assembly to enumerate all the features and functions of code defined in a different assembly, even those that are part of the assembly's internal functional implementation and are not meant to be modified directly (its "private" details). Although obscuring internal details of an assembly is not a viable strategy to protect sensitive information, this permission supports a "need-to-know" policy on implementation details, while exposing the necessary features. Code such as highly trusted software engineering tools or perhaps some administrative tools may require this permission to function, but few, if any, other assemblies should need it.

The Member Access permission allows an assembly to access and invoke all the features and functions of code defined in a different assembly, even its internal features that are not intended to be publicly exposed. To invoke these functions, their names and invocation requirements must be known in advance. Since some functions designed for internal use are not subject to security checks for performance reasons, only highly trusted code should be granted the Member Access permission.

The Reflection Emit permission allows an assembly to create ("emit") executable code on the fly. This might be useful for code that must optimize or restrict its behavior based on alternate execution environments, or that must interpret or compile a programming language. The dynamically created assembly does not automatically inherit the evidence of the creating code. If the creating code has the highly privileged Allow Evidence Control permission, it can specify what evidence that the dynamic assembly will present. In this case, the CLR will use this evidence to grant permissions to the dynamic code based on CAS policy. If the creating code does not have the Allow Evidence Control permission, the dynamic assembly will simply be granted the same permissions as the creating assembly.



Recommendation: Grant the Type Information permission only to highly trusted code that requires access to implementation details—typically this is restricted to software engineering tools or software interoperability services. Grant the Member Access permission only to highly trusted code.

Protected Network Resources

- Printers
- Domain Name System (DNS)
- Network Sockets
- Web Access
- Message Queues
- Windows Services
- Databases

Printers

Printers have four levels of administratively controlled permissions: No Printing, Safe Printing, Default Printing and All Printing. The No Printing permission prevents any access to printers. The Safe Printing permission allows printing only to a destination selected by a user through the Windows Print dialog box. The Default Printing permission allows the user to select a printer destination from the dialog box, and also permits code to programmatically send a document to the default printer. The All Printing permission allows an assembly to programmatically send documents to any available printer without user interaction through the Windows Print dialog box. This may be useful for applications that present a highly customized printing interface, but assumes that code will not take unauthorized actions.



Recommendation: Grant All Printing permission only to highly trusted code.

Domain Name System (DNS)

The .NET Framework provides domain name resolution service through its managed libraries. Access control is provided through the DNS permission and is either granted (Unrestricted) or denied. Granting this permission allows an assembly to perform domain name resolution and associated services through the managed libraries. Host information may be requested such as the IP addresses and aliases corresponding to a domain name or the domain name corresponding to an IP address. Note that this permission only controls requests for information from a DNS service. It does not allow the assembly to create DNS records or control the DNS service. This permission may be used by code to collect information about a local network. Thus, it should only be granted to code that originates in the local network (that is, code that has been strong-named with a key known to be associated with local network entities), or to code that originates from highly trusted external entities.



Recommendation: The DNS permission should typically be granted only to code that originates from within the local network (evidenced by a strong name with a public key associated with a local entity). Code from an external entity should be highly trusted to be granted this permission.

Network Sockets

The Socket Access permission allows code to receive data at specified ports on a local host IP address, or to send data to a specified port on a remote host. The permission is either Unrestricted, or restricted to a list of allowed sockets (IP address and port number pairs). Connections may be made to any remote socket on the list, and the operating system will bind the local (receive) endpoint of the connection to an available local IP address and port number (this may not take place until the first data transmission). If a specific local socket is desired, access must be explicitly granted to the IP address and port number.

The .NET Framework configuration tool (`mscorcfg.msc`) currently allows permission to be granted for TCP only, UDP only, or all transport protocols (specified by selecting both TCP and UDP). Access may be granted to all ports by specifying a port value of -1.

The Socket Access permission protects the ability to perform low-level data transmission that is typically reserved for applications that provide network infrastructure services. Thus, although the ability to send and receive data across a network is at the heart of distributed computing, the Socket Access permission may be too permissive for many managed applications. Specialized types of transmission are provided through the Web Access or Message Queue permissions, and .NET Remoting (the .NET Framework's native application-to-application communication facility).



Recommendation: The Socket Access permission should only be granted to highly trusted code or code that originates from the local network (evidenced by a strong name with a public key associated with a local entity) and provides networking services.

Web Access

The Web Access permission allows code to issue HTTP requests to specified URLs. Unrestricted Web Access also allows code to modify some HTTP settings such as the maximum allowed HTTP response header length. In the default CAS policy, code that satisfies the Membership Conditions for the `Trusted_Zone`, `LocalIntranet_Zone`, or `Internet_Zone` Code Groups (typically code obtained through a Web URL) is automatically granted Connect access to its site of origin through a Net Code Group.

Unrestricted Web access should only be granted to administrative tools or other highly trusted networking applications. Connect access to a URL should not be granted to code that also has access to information that should not be transmitted to the remote site, unless the code is trusted to appropriately protect information that it can access.



Recommendation: Grant the Web Access permission to a specified URL only to code that is denied access to information or resources that should not be shared with the remote site, or is trusted to protect resources that it can access. Unrestricted Web Access should only be granted to highly trusted code that performs networking services.

Message Queues

Message Queuing is a network service maintained by the Windows operating system to allow applications to communicate with each other across a local network. The .NET Framework libraries allow managed code with sufficient permissions to use this facility. Assemblies may send and receive messages as well as browse specific message queues on local or remote machines.

Access to these services is controlled by the Message Queue permission which may be configured for any combination of five Message Queue access types that can be applied to specific named local or remote message queues: Browse, Send, Peek, Receive, and Administer.

Browse permission for a queue allows an assembly to read the message headers of all messages in that queue, but does not allow the assembly to send messages to the queue or to receive/peek at messages. Browse access to all queues allows an assembly to determine whether a path is a valid path to a message queue.

Send, Peek, Receive, and Administer access all include Browse access, with additional authorized actions as follows. Send permission for a queue allows an assembly to append a message to that queue. Peek permission for a queue allows an assembly to retrieve the next message from that queue but not to delete the message from the queue. Receive permission for a queue allows an assembly to retrieve and delete a message from that queue. Administer permission for a queue includes Send and Receive access and also allows an assembly to set queue properties or to delete a queue. Unrestricted queue access allows an assembly to create queues.

Access permissions are associated with specific message queues, identified either by paths or descriptions. Queue path identifiers have the form “<machinename>/<queueName>”, where “.” can stand for the local host and “*” for all queues. A queue may also be identified by descriptive terms such as category and/or labels defined by the queue administrator. Category names are identifiers that group queues with common features and allow Message Queue permissions to be granted to all queues in a category. Queue labels are short descriptive identifiers that are intended to be unique across the entire local network. The Message Queue service will resolve queue labels to determine the host machine where the queue resides. If two queues on the same network have the same label, this resolution process may fail, preventing messages from being sent to those queues.

The CLR library that provides access to message queues (`System.Messaging.dll`) is not marked for direct use by partially trusted code, so code that is allowed to use messaging may still need to rely on a Fully Trusted helper assembly.

Since message queuing is a local network facility, its use should only be granted to code that originates in the local network, or to code that originates from external parties with prior trust relationships. In keeping with the principle of least privilege, code should be granted access only to specific queues or categories of queues, and only the minimum access needed to function. Code that only needs to send alerting messages should not be granted the Receive or Peek permission. Since Send access includes the ability to browse message headers, it is not possible to grant permission to send a message without also granting access to some message queue information unrelated to the sent message.



Recommendation: The Message Queue permission should only be granted to code that originates from within the local network (evidenced by a strong name with a public key associated with a local entity) or from a highly trusted external entity. Administer access to any queue and Browse access to all queues should only be granted to highly trusted administrative tools.

Windows Services

The Service Controller permission controls access to the control infrastructure of Windows Services rather than to the consumption of those Services. A Windows Service is controlled by sending control messages to the Service Control Manager resident on the machine where the service is running. Access is granted to managed code in three levels, None, Browse, and Control, that are applied to named services on specified computers. Browse access allows an assembly to get the current status of a service as well as determine other services that a services depends on or supports. Control access permits the code to issue commands to a service such as Start, Stop, Pause, Continue, or service-specific custom commands.

The CLR library that provides access to services (`System.ServiceProcess.dll`) is not marked for direct use by partially trusted code, so code that is allowed to issue commands to Windows services may still need to rely on a Fully Trusted helper assembly.

Since control of services is typically an administrative function that affects the system asset of service availability, the Service Controller permission for a Windows service should only be granted to applications that are as trusted as the service itself.



Recommendation: Grant the Service Controller permission for a Windows service only to assemblies whose trust is as high as the service itself and commensurate with the value of the availability of the service.

Databases

The .NET Framework allows an administrator to control assembly access to databases by restricting connection information used by data providers. A data provider is a software library that assemblies can use to store and retrieve data in a database. Some data providers are designed to use a standardized or well-known data access protocol. They are “generic” in that they can be used to access multiple vendor-specific database formats, provided the database vendor has built in support for the protocol. These data providers typically interact with other product-specific data providers. Version 1.0 of the .NET Framework is distributed

with the generic OLE DB data provider and a product-specific data provider for Microsoft SQL Server (called the “SQL Client” data provider). Two additional data providers, a generic ODBC provider and an Oracle product-specific data provider (“Oracle Client”), can be installed separately. Version 1.1 of the .NET Framework is distributed with all four of these data providers.

CAS policy is designed to provide access control over the use of data providers. By itself, this does not control access to any specific database or data sources. Fine-grained access control to individual databases is configured through the CAS policy settings for each data provider permission. The standard means of initiating interaction with a specific database is through a connection string that is processed by the data provider. CAS policy provides access control over the use of components of connection strings (i.e., (key, value) pairs). Unfortunately, the .NET Framework Configuration Tool, `mscorcfg.msc`, does not support the level of access control that can be implemented in XML in the CAS policy configuration file.

In both version 1.0 and 1.1 of the .NET Framework, `mscorcfg.msc` offers only a limited set of ways to configure access to data providers or to databases. Neither version of `mscorcfg.msc` has built-in support for the ODBC or Oracle Client data provider permissions. In addition, `mscorcfg.msc` allows only coarse access control to the use of connection string components for the OLE DB or SQL Client data providers. Furthermore, when `mscorcfg.msc` is used to view database permission settings for permissions that include the more fine-grained access control features in the XML files, it will discard these elements.

Version 1.0 of the .NET Framework

In version 1.0 of the .NET Framework, the XML structure for a database permission is as shown in Figure 8:

```
<IPermission class="{System.Data.OleDb.OleDbPermission |
    System.Data.SqlClient.SqlClientPermission}, System.Data,
    Version 1.0.5000.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089"
    version="1"
    AllowBlankPassword="{True | False}"
    Unrestricted="{True | False}">
</IPermission>
```

Figure 8. XML Structure of a Database Permission in Version 1.0.

An optional additional child element `<keyword>` is available for the OLE DB permission. This element will be described in more detail in the section on OLE DB.

The class attribute of the `<IPermission>` element fully identifies the CLR library software that implements the specified permission. The example above shows the possible values for

the built-in data provider permissions for version 1.0 of the .NET Framework. The installable ODBC and Oracle Client permissions would be formatted similarly. The `version` attribute specifies a version of the `IPermission` interface. The `AllowBlankPassword` attribute indicates whether the assembly granted this permission will be allowed to specify a blank password in a connection string. In version 1.0 of the .NET Framework, a connection string is considered to contain a blank password if “password=;” or “pwd=;” is a connection string component. The `Unrestricted` attribute, if set to “True”, indicates that the data provider may be used with any connection string (including ones with blank passwords).

Version 1.1 of the .NET Framework

In version 1.1, the XML structure for a database permission is as shown in Figure 9:

```
<IPermission class="{System.Data.OleDb.OleDbPermission |
    System.Data.SqlClient.SqlClient.Permission |
    System.Data.Odbc.Odbc.Permission}, System.Data,
    Version 1.0.5000.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089"
    version="1"
    AllowBlankPassword="{True | False}"
    Unrestricted="{True | False}">
  <add ConnectionString= "... " KeyRestrictions= "... "
    KeyRestrictionBehavior= "{AllowOnly | PreventUsage}"/>
  ...
  <add ConnectionString= "... " KeyRestrictions= "... "
    KeyRestrictionBehavior= "{AllowOnly | PreventUsage}"/>
</IPermission>
```

Figure 9. XML Structure of a Database Permission in Version 1.1.

The Oracle Client data provider permission is similar, except that the class attribute begins “System.Data.OracleClient.OracleClientPermission, System.Data.Oracleclient, ...”.

In version 1.1 of the .NET Framework, a connection string is considered to contain a blank password if there is an explicit blank password, that is, “password=;” or “pwd=;” are included in the connection string, or an implicit blank password. An implicit blank password occurs when the user identity is passed as “userid={any value};” or “uid={any value};”, but no password is provided at all, that is, the keys “password” or “pwd” do not appear in the connection string.

Version 1.1 also provides support for the `<add>` element. This element is used to provide access control to specific databases by controlling what components are permissible in

connection strings. When an assembly accesses a database through one of the data providers, it supplies a connection string that specifies parameters for the access such as the server name or IP address of the database server, the name of the database, a connection timeout, a username and password, and so on.

When access is attempted with a particular connection string, the CLR will check that the assembly has the proper data provider permission. If granted access to the desired data provider, the CLR will then check that there are no restrictions on the (key, value) pairs specified in the assembly's connection string. Each <add> element for a data provider permission defines a connection string template. The `ConnectionString` attribute specifies a base set of components. Each <add> element only defines restrictions for connection strings that contain at least the (key, value) pairs in its `ConnectionString` attribute. A missing `ConnectionString` attribute is equivalent to an empty string, and will define restrictions that will apply to all connection strings.

The CLR will parse the assembly's connection string into its component (key, value) pairs and discard duplicates. When checking for duplicates, the comparison is case-insensitive, and alternate forms of key names are taken into consideration. For example, when specifying the network address of the database server, the key name can be "data source," "addr," "address," "server," or "network address." The last value is kept of any set of duplicates in the same string.

After parsing the connection string, the CLR checks each <add> element that applies (i.e., whose parsed `ConnectionString` attribute value is a subset of the desired connection string). If the <add> element has a `KeyRestrictionBehavior` attribute of "PreventUsage", then the key names listed in the `KeyRestrictions` attribute are prohibited from appearing in the assembly's connection string. If the <add> element has a `KeyRestrictionBehavior` attribute of "AllowOnly", then any key names not listed in the `KeyRestrictions` attribute are prohibited. Alternate key names are taken into account – if a key name is allowed or prohibited, so are any alternate forms.

The <add> element does not restrict key values. The key values present in the `ConnectionString` attribute determine the base connection string to which the defined restrictions will apply. If a connection string includes the same key names as the `ConnectionString` attribute, but has different values, the <add> element will not apply. If a connection string does not contain the base (key, value) pairs from the `ConnectionString` attribute for any <add> element, it is prohibited. The <add> element can thus be considered permission to use the base connection string specified by the `ConnectionString` attribute with the potential addition of certain other key names (having any corresponding values). Table 1 illustrates the combined effect of <add> element attributes. The leftmost columns represent the value of the `ConnectionString`, `KeyRestrictions`, and `KeyRestrictionBehavior` attributes, respectively. In this table, the uppercase letters "X," "Y," and "Z" represent disjoint sets of (key, value) pairs, while lowercase letters "x," "y," and "z" represent the corresponding sets of key names only. Sets may be empty. A missing `ConnectionString` or `KeyRestrictions` attribute is equivalent to an empty string.

UNCLASSIFIED

CS	KR	KRB	Combined Effect
""	""	AllowOnly	Only the empty connection string is permitted, which is equivalent to denying access to all databases, since this <add> element will apply to all connection strings.
""	""	PreventUsage	All connection strings are permitted. Any connection string will contain the base (empty) string, and no additional keys are prevented.
""	x	AllowOnly	Permit only connections strings whose key names are all listed in x.
""	x	PreventUsage	Permit only connection strings that do not contain any key names listed in x.
X	""	AllowOnly	Permit the exact connection string X but reject any connection string that contains X plus additional components.
X	""	PreventUsage	Permit all connection strings that contain at least the (key, value) pairs in X. Any additional (key, value) pairs are permitted.
X	y	AllowOnly	Permit all connection strings that contain all the (key, value) pairs in X, with the possible addition of key names listed in y (with any corresponding values).
X	y	PreventUsage	Permit all connection strings that contain all the (key, value) pairs in X, with the possible addition of key names that are not listed in y (with any corresponding values).
X+Y	x	AllowOnly	In this combination, the key names in the KeyRestrictions attribute also appear in the ConnectionString attribute. Permit the exact connection string X+Y but reject any connection string that contains X+Y plus additional components.
X+Y	x	PreventUsage	In this combination, the key names in the KeyRestrictions attribute also appear in the ConnectionString attribute. This <add> element cannot permit any connection strings, since the very connections strings to which it applies (those containing all the (key, value) pairs from X+Y), also contain prohibited key names by definition. Thus, this combination will cause the rejection of all connection strings that contain at least the (key, value) pairs in X+Y.

CS	KR	KRB	Combined Effect
X+Y	y+z	AllowOnly	In this combination, some key names in the KeyRestrictions attribute appear in the ConnectionString attribute. Permit connections strings that contain all the (key, value) pairs in X+Y, with the possible addition of key names listed in z (with any corresponding values).
X+Y	y+z	PreventUsage	In this combination, some key names in the KeyRestrictions attribute appear in the ConnectionString attribute. This <add> element cannot permit any connection strings, since the very connection strings to which it applies (those containing all the (key, value) pairs from X+Y), also contain prohibited key names by definition. Thus, this combination will cause the rejection of all connection strings that contain at least the (key, value) pairs in X+Y.

Table 1. Combined Effect of <add> Element Attributes.

An assembly's connection string must be permitted by all <add> elements that apply to it. Thus, multiple <add> elements combine to create more complex access control rules. Element and attribute names are case-sensitive in XML, so if the CLR finds misspelled or wrong-case elements or attributes, they will be ignored and their default values used instead. For example, the <add> element

```
<add ConnectionString="database=mydb;server=myserver"
      KeyRestrictionBehavior="PreventUsage"/>
```

may have been intended to allow only connection strings that contain at least the pairs (database, mydb) and (server, myserver), but instead will allow any connection string whatsoever. The misspelling of "ConnectionString" makes this equivalent to:

```
<add ConnectionString="" KeyRestrictions=""
      KeyRestrictionBehavior="PreventUsage"/>
```

If any attribute values are unexpected or misspelled, a policy exception will result whenever the corresponding permission is parsed by the CLR during its policy resolution process. This may also cause mscorcfg.msc to discard the containing <PermissionSet> element. mscorcfg.msc may discard <add> elements whenever the properties of a database permission are viewed through that tool and the **OK** button is clicked (even if no changes were made). Thus, manual editing of the CAS policy files is a challenge, but it is the only way to insert <add> elements to create a fine-grained access control policy for database access.

OLE DB

Native OLE DB data providers are available to support a wide variety of databases, including Microsoft SQL Server, Oracle, and Microsoft Access. These native libraries are available through the OLE DB data provider, the use of which is controlled by the OLE DB permission.

In version 1.0 of the .NET Framework, the OLE DB permission may be configured to allow specific native OLE DB data providers as well as to specify whether or not blank passwords are permitted. The optional `<keyword>` element is used to specify native OLE DB data providers (Figure 10):

```
<keyword name="provider">
  <value value="..." />
  ...
  <value value="..." />
</keyword>
```

Figure 10. XML Structure of an OLE DB Data Provider `<keyword>` Element.

If the list of allowed providers is empty, either by having no `<value>` elements or by having no `<keyword>` element, then all providers are allowed. In version 1.1 this behavior changes if there are `<add>` elements. The presence of `<add>` elements, even ones that do not apply to a particular connection string, will cause a “fail secure” behavior. For example, if no `<add>` elements apply to and allow a connection string, the connection string will be rejected. This replaces the blanket access granted by the absence of a `<keyword>` element or a `<keyword>` element containing no `<value>` elements.

In version 1.1, the `<add>` elements are used by `mscorcfg.msc` to implement the function of the `<keyword>` element (although the `<keyword>` element is still used). `mscorcfg.msc` will create `<add>` elements of the form

```
<add ConnectionString="provider={provider name}" KeyRestrictions=""
  KeyRestrictionBehavior="PreventUsage"/>
```

which will allow any connection string that contains the specified provider as a value for the “provider” key name. It will also discard any `<add>` elements that do not correspond to `<value>` elements under the `<keyword>` element.

In version 1.0 of the .NET Framework, the calling assembly (but not all other assemblies in the call stack) must be Fully Trusted to invoke the managed OLE DB data provider library. This prevents any partially trusted code from accessing the OLE DB data provider directly – access must be through a Fully Trusted intermediate assembly that has been written to securely provide data access services to partially trusted code (for example, by fully validating all user input and making appropriate demands for the OLE DB permission). In

version 1.1, additionally, the calling assembly and its callers, all the way up the call stack, must have Full Trust. This forces the intermediate library to explicitly assert its permission to access data providers on behalf of partially trusted code.

ODBC

The capability to use native ODBC data providers is included in .NET Framework version 1.1 or later, and is available for .NET Framework version 1.0 as a separate managed data provider download. In version 1.1, the ODBC permission provides access control over connection string components. As of version 1.1 of the .NET Framework, the ODBC permission is not configurable using `mscorcfg.msc`. Instead, the configuration files must be edited by hand or the XML must be imported into the CAS policy files using `caspol.exe`. In version 1.0 of the .NET Framework, an assembly (but not all the other assemblies in the call stack) must be Fully Trusted to invoke the ODBC data provider library. For version 1.1, additionally, every assembly in the call stack must be Fully Trusted while using the managed ODBC data provider.

Oracle Client

The Oracle Client data provider can connect to Oracle databases through Oracle client software versions 8.1.7 or later. This provider is not distributed with version 1.0 of the .NET Framework, but is available as a separate download. The Oracle Client data provider is included in version 1.1.

In version 1.0, an assembly invoking the Oracle Client data provider must have Full Trust to invoke the managed library. In version 1.1, this is further restricted by requiring that the entire call stack have Full Trust while using the Oracle Client data provider.

The Oracle Client data provider has a built-in list of allowed key names and their alternates. Only key names from this list can be used, regardless of any permission settings. If the CAS policy files are edited to include a permission for the Oracle Client data provider that uses a key name not found in this list, a policy exception will occur, and the assembly whose permissions are being granted will not be allowed to execute. `mscorcfg.msc` will discard any permission set that contains an Oracle Client permission with an invalid key name specified in either the `ConnectionString` or `KeyRestrictions` attributes of an `<add>` element.

SQL Client

The SQL Client data provider allows access to Microsoft SQL Server version 7.0 or later. Earlier versions of SQL Server can be accessed through its OLE DB interface. In version 1.0 of the .NET Framework, an assembly and its entire call stack must be Fully Trusted at run time to use the SQL Client data provider. In version 1.1 of the .NET Framework, partially trusted assemblies may use SQL Server databases if the SQL Client permission is granted. As with the Oracle Client data provider, SQL Client has a built-in list of allowed key names and alternates. For version 1.1, `mscorcfg.msc` will always set the `AllowBlankPassword` attribute to "False" for this permission.

Summary of Trust Requirements

Table 2 summarizes when Full Trust is required to use each data provider.

Data Provider	Demands Full Trust at link time (only the calling assembly must be Fully Trusted)	Demands Full Trust at run time (all assemblies in the call stack must be Fully Trusted)
.NET Framework version 1.0		
SQL Client		X
OLE DB	X	
ODBC	X	
Oracle Client	X	
.NET Framework version 1.1		
SQL Client		
OLE DB	X	X
ODBC	X	X
Oracle Client	X	X

Table 2. Summary of Trust Requirements for Data Providers.

Since automated tool support is lacking for the fine-grained access control features offered by the CLR, misconfiguration becomes a greater risk. This is partially mitigated by the Full Trust requirements for the OLE DB, ODBC, and Oracle Client data providers. However, since administrative tool support is centered around a more coarsely-grained configuration that does not allow access control over connection parameters, it may be preferable to prevent database access except through highly trusted code.



Recommendation: Grant any of the database permissions only to assemblies that are highly trusted.

Protected Administrative Resources

Protected Administrative Resources are resources that aid assemblies in operating smoothly or more efficiently, aid administrators in measuring system performance, or aid administrators in configuring the system.

- Security Settings
- Performance Counters
- Environment
- Event Logs
- Registry
- Directory Services

Security Settings

The .NET Framework has 13 security permissions that manage the security environment of assemblies. Unlike other resource permissions that control access to data or services on a local or remote host, these permissions allow or disallow specific actions by assemblies that could have a significant impact on the security posture of the system. The Security permissions can be grouped into categories of control:

- Runtime Environment
- Execution
- CAS Policy

Runtime Environment

Permissions that affect the runtime environment of the .NET Framework are those that allow code to act as an extension of the CLR's trusted library base, or to configure and manage the execution environment. These permissions include:

- Extend Infrastructure
- Enable Remoting Configuration
- Enable Serialization Formatter
- Enable Thread Control
- Allow Principal Control
- Create and Control Application Domains

Extend Infrastructure

The ability to extend the .NET Framework "infrastructure" allows an assembly to insert custom code into the handling chain that sends and receives messages between managed applications (whether local or remote), or to serve as part of the chain. The .NET Framework provides the ability to create custom message handlers to support distributed applications whose transaction and messaging requirements are too specialized to be served by the provided CLR libraries. The Extend Infrastructure permission essentially allows code to become or to create a trusted intermediary that can intercept cross-application messages as they are processed, perform some additional or substitute processing, or discard and replace the original messages altogether, and then pass the results on to the next party in the communication process. Clearly, this requires the code itself to be trusted to perform only its intended function. In addition, code that is serving as part of the communications infrastructure is unlikely to require access to a broad range of protected resources, and should be granted only those additional permissions it requires.

Code that provides infrastructure extensions is typically not called directly by other applications. Thus, it should only be referenced by highly trusted code. This can be achieved through the software development process by making implicit or explicit demands that callers have Full Trust, are strong named with a private key or digitally signed with a publisher's certificate associated with a highly trusted party.



Recommendation: Grant the Extend Infrastructure permission only to code that is trusted to have complete control over message processing.

Enable Remoting Configuration

An application's Remoting Services configuration determines the type and properties of the communications channels that the application will use, and the code components with which it may communicate. The configuration settings are stored in the application's configuration file in its installation directory. In addition, some settings may be stored in the machine-wide configuration file `machine.config` for a particular .NET Framework version. The machine-wide configuration is automatically applied by the CLR, but may be overridden by an application's configuration file if the application is granted the Enable Remoting Configuration permission. The Remoting Services configuration contained in `machine.config` is applied to all applications, and thus will typically define common communication channels rather than specify individual software components that will send or receive messages.

An application's own configuration file is available for use only when granted this permission, that is, the Enable Remoting Configuration permission grants an assembly the ability to load the settings in its own configuration file. Since these override settings in `machine.config`, this is a privileged action and should be granted only to software that is from a highly trusted source.

In order to achieve this access control granularity, CAS policy should be structured to prevent this permission from being granted to code in a specified set. CAS policy does not allow the specification of explicit denials of a permission, and omitting this permission in the Named Permission Set associated with one Code Group does not prevent it from being granted through another Code Group. One alternative is to create assembly-specific Code Groups based on strong name or hash for each assembly that should not receive this permission and grant exactly the permissions that code is authorized to receive (the Exclusive attribute must be set on the Code Group). If this is infeasible, then CAS policy should be configured so that the Enable Remoting Configuration permission is not granted based on a broadly defined Membership Condition, such as Zone or Site.



Recommendation: The Enable Remoting Configuration permission should be granted only to software from a highly trusted source. The same considerations apply that would govern the granting of Unrestricted Web access or Unrestricted network socket access.

Enable Serialization Formatter

Serialization is the process of converting structured (but not necessarily sequential) data in memory into a form that can be stored or transmitted as a sequence of bits. This sequence may then be converted back (deserialized) into the proper structure at a later time or in a different context. This process may also be used to create an exact clone of structured data (software “objects” or collections of objects) that may be passed to and used by a different assembly. Serialization records all of the data needed to recreate an object, even the internal values that are not meant to be directly accessible. Thus, the sequence of bits that is stored or transmitted could contain internal state information of the original object.

Assemblies may contain publicly accessible serialization code that defines how its internal data is meant to be serialized. Software objects usually do not initiate their own serialization; rather, this code is typically used by the trusted CLR libraries to store or transmit data as needed. Unless protected, it may also be called by any other code to transform the internal state of an object to a sequence of bits and then inspect or modify those bits in unauthorized ways. The serialization code must be marked during the software development process to require that any code attempting to serialize an object must have the Serialization Formatter permission. Thus, the effectiveness of the Serialization Formatter permission as an access control relies on a partnership with software developers. Nevertheless, it can be used to protect the confidentiality and integrity of the many objects in the .NET Framework libraries that do demand this permission.

Serialization is a process internal to the execution environment of the .NET Framework – assemblies that perform serialization of other objects should be considered extensions of the CLR infrastructure. Thus, Serialization Formatter permission should only be granted to highly trusted code.



Recommendation: Grant Enable Serialization Formatter permission only to highly trusted code that will be considered an extension to the CLR’s trusted library base.

Enable Thread Control

Assemblies may spawn multiple threads that may simultaneously execute. In addition, the CLR maintains a pool of worker threads that can execute code from any assembly as needed. Code with the Enable Thread Control permission may abort, suspend, or resume its thread. These types of activities are privileged operations. The ability to manipulate a thread should only be granted to code that is equal or higher in trust than the threads that it can manipulate. Since code may run in the context of CLR worker threads that are Fully Trusted, the Enable Thread Control permission should be granted only to code that is Fully Trusted. Note that this permission is not necessary for code to provide normal multithreaded operation.



Recommendation: Grant Enable Thread Control permission only to Fully Trusted code.

Allow Principal Control

Role-based access control (RBAC) is supported in the .NET Framework through CLR library software that assemblies can use to determine information about its own Principal. A Principal is the user context under which an assembly is executing and consists of the user identity plus the roles that the user has assumed for the current logon session. User roles can be the ones corresponding to the built-in Windows operating system groups of “User,” “Administrator,” “Guest,” “AccountOperator,” “BackupOperator,” etc., or custom roles/groups. The assembly may then decide to grant access to resources based on the roles that the current user holds.

The Allow Principal Control permission allows an assembly to determine the default Principal that will be associated with a new thread of execution, to determine the identity of the current Windows user, to impersonate another user, or to change the Principal associated with itself. Note that this permission is not necessary for code to use RBAC to determine what actions it will perform. It is only necessary for code that intends to impersonate another user. Thus, it should be granted only to code that is trusted at least as much as the most trusted Principal (user identity and role) available on the system.



Recommendation: Grant the Allow Principal Control permission only to code that is trusted at least as much as the most trusted user account on the system.

Create and Control Application Domains

The .NET Framework uses Application Domains to provide isolation between managed applications. When a runtime host application starts, it loads the CLR into its process space, creates an Application Domain, and then loads and executes assemblies within the Application Domain. Other Application Domains may be created as needed for assemblies that must be kept isolated from one another. More than one Application Domain may exist within a single operating system process. One task of the CLR is to maintain separation and isolation of memory and other resources between Application Domains.

The Create and Control Application Domains permission allows an assembly to isolate other assemblies in separate domains, or to load multiple, isolated instances of the same assembly. Created Application Domains will not have any AppDomain level CAS policy unless one is provided by the Application Domain’s creator (this requires the Allow Domain Policy Control permission). A blank AppDomain level CAS policy is ignored by the CLR when determining permissions.

Execution

The following Security permission settings determine how tightly “managed” an assembly will be, that is, how strictly the CLR will be able to control its execution:

- Enable Assembly Execution
- Skip Verification

- Allow Calls to Unmanaged Assemblies

Code that has none of these permissions cannot execute at all. Code with the Enable Assembly Execution permission but not the other two permissions is strictly managed by the CLR, and will not be allowed to run unless it passes a verification process designed to ensure that it is strictly manageable. Code with the Skip Verification permission is allowed to run even if it contains some code that the CLR cannot manage. In this case, some program instructions may be executed even though the CLR cannot verify the validity of the instruction parameters. Code with the Allow Calls to Unmanaged Assemblies permission can invoke software that is outside the sphere of the CLR's management capability. In this case, code management by the .NET Framework is effectively suspended until the unmanaged software returns control to the assembly that invoked it. These three permissions are described more fully below.

Enable Assembly Execution

The Enable Assembly Execution permission allows code to execute. This permission supports a performance enhancement of the CLR that checks for this permission during the assembly load process and aborts the process if this permission is not granted. This avoids wasting resources on managed code that will never be allowed to execute. It also serves to create a more finely grained security mechanism, as execution itself can be abstracted away from access to other resources. The default Execution Named Permission Set facilitates this separation. The permissions that control access to resources should be granted in keeping with the principle of least privilege or separation of duties; however, the Execution permission is not tied to any specific functional requirements of code, but is a statement of the trustworthiness of the code's origin. As such, this permission should be tied to some policy-driven threshold of assembly evidence that provides assurance of trusted origin. If the organizational security policy is to consider Internet code untrusted, then granting this permission based on membership in the Internet URL Security Zone is a violation of policy. Instead, code from the Internet should present some additional evidence by which to determine trust.

In general, the Assembly Execution permission is tied to origin, not to functionality, and as such should probably be granted based on a known public key used to create an assembly's strong name or a publisher's digital signature, rather than based on the assembly's purpose. If the origin is later determined to be untrusted, all code from that source may be denied execution regardless of what each assembly from that source claims to do.



Recommendation: The Enable Assembly Execution permission should be granted based on the level of trust associated with the assembly's origin, as established by evidence stronger than URL Security Zone. If possible, separate the Enable Assembly Execution permission from resource access permissions, so that the former is tied to origin and embodies a trust relationship, while the latter are tied to functional requirements of code and embody the principle of least privilege.

Skip Verification

Managed code is considered “type safe” if it only accesses data through carefully defined and restricted means and only converts data between compatible forms. The code isolation and access control features of the .NET Framework assume that code is type safe. When a managed assembly is about to execute, the CLR attempts to verify its type safety. This is not a completely reliable determination – the verification process may fail if it encounters code that cannot be unambiguously determined to be type safe. Thus, code could actually be type safe, but still fail the verification process. Nevertheless, if it cannot be verified to be type safe, there is no assurance that CAS policy can be reliably enforced, and it will not be allowed to execute.

Some programming languages and compilers (such as the Microsoft Visual C++ compiler) regularly produce non-verifiably type safe code. Yet it may be necessary for operational reasons to use such code. In this case, the Skip Verification permission may be granted to allow non-verifiable code to execute.

The decision to grant the Skip Verification permission should be applied based on the origin of the code, not on any claimed functionality. In this regard, this permission is similar to the Enable Assembly Execution permission, and is best abstracted away from permissions that provide access to resources. The default SkipVerification Named Permission Set facilitates this separation. Granting the Skip Verification permission is a strong statement about the relationship of trust between the organization and the source of the code. This is a much stronger permission than Execution, and should never be granted to code that is not highly trusted. Although this permission is based on the level of trust in its source, it is still advisable to grant it only where necessary rather than broadly to all code with a given trusted origin. Thus, it should be tied to an assembly’s strong name evidence that includes a public key known to be associated with the trusted source, as well as the assembly’s name and version.



Recommendation: Skip Verification should be granted only to highly trusted code based on a hash identity or strong name evidence that includes the assembly’s name, version, and public key associated to a trusted party. If possible, separate the Skip Verification permission from resource access permissions, so that the former is tied to a specific assembly from a trusted point of origin and embodies a trust relationship, while the latter are tied to functional requirements of code and embody the principle of least privilege.

Allow Calls to Unmanaged Assemblies

Unmanaged code refers to software that runs outside the CLR’s execution environment and thus is not constrained by the security enforcement mechanisms of the .NET Framework. All code is unmanaged unless it is compiled for the .NET Framework by a .NET-aware compiler. Currently, most office automation applications, networking utilities, Web browsers, etc., as well as most of the Windows libraries that support them, are unmanaged code. As such, they are not subject to CAS policy. Unmanaged code runs with full control over the computing environment and is subject only to the Windows operating system security. Unmanaged

code may view, use, modify, or delete any resource available to the user, and thus is particularly risky.

Managed code can only invoke unmanaged code if it is granted the Unmanaged Assemblies permission. This permission should only be granted to highly trusted code, since the unmanaged components will execute with the same privilege as the CLR itself.



Recommendation: Allow Calls to Unmanaged Assemblies should be granted only to code that is trusted to execute with the same privileges as the CLR itself.

CAS Policy

The CAS Policy settings allow an assembly to control the configuration or application of security policy. These permissions include:

- Allow Policy Control
- Allow Domain Policy Control
- Allow Evidence Control
- Assert any Permission that Has Been Granted

Allow Policy Control

The Allow Policy Control permission allows an assembly to view and modify the current CAS policy settings, including the ability to turn CAS policy off. Obviously, this is a powerful permission that should only be granted to highly trusted administrative tools.



Recommendation: The Allow Policy Control permission should be granted only to highly trusted .NET Framework administrative tools.

Allow Domain Policy Control

The Allow Domain Policy Control permission allows an assembly to set the CAS policy for its own Application Domain or one that it creates. This would allow an assembly to override any CAS policy for the AppDomain level that has been configured by the Runtime Host process, or to simulate the assembly loading functions of a Runtime Host. This is an appropriate function for trusted administrative tools. Partially trusted code may receive this permission when CAS policy at the AppDomain level is not used by a Runtime Host to implement organizational security policy, but is reserved for custom use by assemblies that need to create and maintain Application Domains (this is the default case). In this case, the Enterprise, Machine, and User level CAS policies are used to enforce policy, with the AppDomain level providing additional restrictions appropriate to the code that will be loaded into the new Application Domain.



Recommendation: If custom Runtime Host applications are in use that implement organizational policy using the AppDomain CAS policy level, then the Allow Domain Policy Control permission should be granted only to code that is highly trusted. In other cases (including the typical default installation), this permission should be granted only to code that is designed to dynamically launch other applications that may be less trusted than itself.

Allow Evidence Control

The Allow Evidence Control permission allows an assembly to supply or modify the evidence that will be associated with itself, other assemblies it loads, or Application Domains it creates. Since the supplied evidence will be used by the CLR in conjunction with CAS policy to determine access to resources, this effectively allows control over the .NET Framework's security system. This permission is appropriate for code that provides access control services as an extension of the trusted CLR base libraries. For example, software that implements custom permissions that are used to extend the CAS policy system must have this permission.

Code that is designed to extend the trusted CLR base libraries must be developed using secure coding practices. For a more detailed discussion of these practices, see [LaMacchia, et al., 2002], [Microsoft, 2002], or [Meier, et al., 2003]. This permission should only be granted to software developed by trusted parties that have a demonstrated secure development process.



Recommendation: The Allow Evidence Control permission should be granted only to code developed by trusted parties with demonstrated secure coding practices. Code granted this permission effectively becomes an extension of the CLR's access control system.

Assert Any Permission That Has Been Granted

Permissions granted to an assembly through the application of CAS policy are not necessarily usable during code execution, because access to a resource is granted only if the assembly plus the code that invoked it both have the required permission. If multiple assemblies are in the "calling chain," then all will have to have the permission. Thus, Fully Trusted library code that has wide-ranging access to resources may safely expose its functionality to partially trusted code without the partially trusted code being able to access any resources that it has not been authorized to access. There are times when a more trusted assembly may need to access a resource that any less trusted callers shouldn't be able to access directly, for example, to look up an internal setting in a registry key. In these cases, the more trusted assembly may temporarily assert its privilege to access the registry key even though its caller may not have permission to do so. The CLR will allow access to the registry while the assertion is in effect. This ability to temporarily assert privilege is granted by the Assert any Permission that Has Been Granted setting.

Code that asserts a granted permission must be developed using secure coding practices. While asserting a permission, the code must not provide discretionary access to the protected

resource, or the trusted code has effectively transferred its access to the resource to less trusted code and violated the security policy that the CAS policy implements. This is the software equivalent of “piggybacking” through a physical checkpoint. Thus, assertion of a permission is only appropriate when code must access pre-defined resources that are used internally and will not be exposed to other code or to users.



Recommendation: The Assert any Permission that Has Been Granted permission should be granted only to software that is from a trusted developer with demonstrated secure coding practices. Typically, this permission is granted to highly trusted extensions to the CLR base libraries.

Performance Counters

Performance Counters are a Windows feature that allows applications to capture and publish performance information about the system and itself. The CLR libraries provide support for managed code to access the Windows performance counters. Performance counters monitor the performance of physical system components such as processors, disks, and memory, system objects such as processes and threads, and services and applications such as Web or FTP services or print queues.

Performance counters are grouped into categories that form the basis for .NET Framework access control. Access is granted to a named category of performance counters on a named network host. A performance counter category may have several instances that represent different subjects of performance information. For example, the Process category will have an instance for each process running on the system. Each performance counter in a category provides separate data values for each instance (although there may be an instance that represents an aggregate of all other instances). When access is granted to a category, it is granted to every performance counter for every available instance in the category.

Access is granted in four levels, None, Browse, Instrument, and Administer (Administer is equivalent to the Unrestricted state). Browse access allows an assembly to enumerate the instances and performance counters and to read the current value of any performance counter in the permitted category. It will also allow the enumeration of the available categories on any host if access is granted to the “*” category. Note that host-independent code may use “.” as a substitute for the local machine name. Instrument access allows an assembly the additional ability to write to a non-read-only counter. The Windows system performance counters are all read-only, but custom categories and counters may be defined that allow managed code with Instrument access to function as a performance monitor and write data that can be consumed by other managed code with Browse access to the named custom counter. Administer access allows read and write of existing counters and the creation of custom categories and counters.

Since the Performance Counter permission is configured with the text names of performance counter categories, the exact name of the category must be used. Available performance counter categories and counters on the system can be identified by executing the `perfmon.exe` tool. To see a list of the system performance counters, perform the following steps:

- Run `perfmon.exe`.
- Click the properties button on the toolbar (see Figure 11).

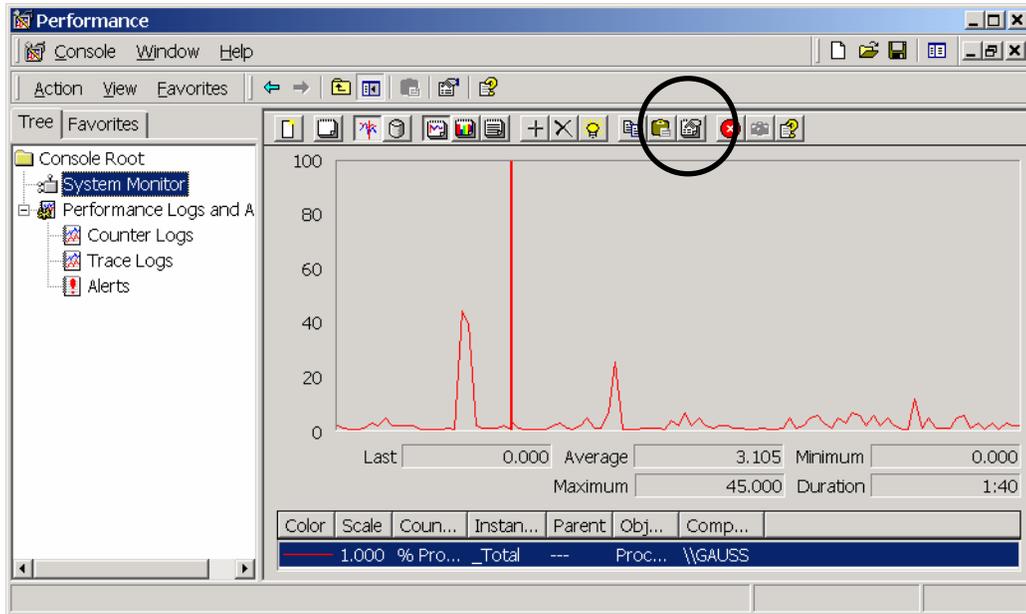


Figure 11. Properties Button in `perfmon.exe` Window.

- Select the **Data** tab in the System Monitor Properties dialog box (see Figure 12), and click **Add...** to open the Add Counters dialog box (see Figure 13).

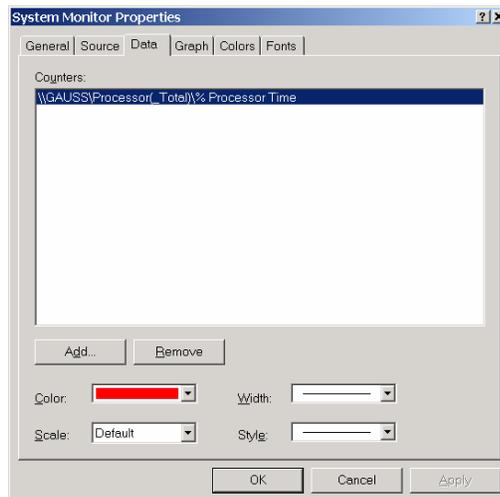


Figure 12. System Monitor Properties Dialog Box.

The performance counter category names are listed in the dropdown box labeled **Performance object**. The names of the individual performance counters within the selected category appear in the **Select counters from list** list box, while the names of any

performance counter instances appear in the **Select instances from list** list box. Each named performance counter will provide separate values for each available instance.

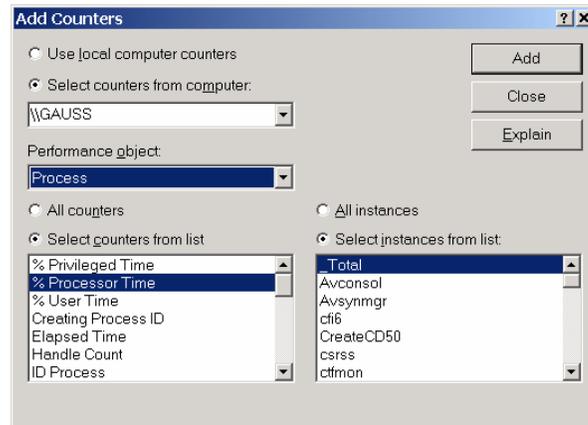


Figure 13. Add Counters Dialog Box.

As with other finely-grained access control mechanisms, the principle of least privilege should govern the use of this permission.



Recommendation: Grant Performance Counter access to the most restrictive set of performance counter categories possible. Grant Instrument or Administer access only to trusted code that provides or administers a monitoring service.

Environment

Sensitive information about the execution environment is protected by the Environment permission. The .NET Framework configuration tool (`mscorcfg.msc`) refers to this permission as the Environment Variables permission, although more is protected than just environment variables. Each permission entry is specified by an access type and a resource name. These permission semantics are basic to access control systems and are found in many other types of CAS permissions. In the Environment permission, access types include NoAccess, Read, Write, and AllAccess (Read and Write). In addition, Unrestricted access grants AllAccess permission to all resource names. Resource names in the Environment permission are case-insensitive. When the resource name is the name of an environment variable, the specified access is granted to the value of that variable. However, the resource name could be any text, and an assembly could use the existence of a specified (access type, resource name) pair to authorize access to any type of resource.

Note that the CLR libraries in the .NET Framework 1.0 and 1.1 do not support adding, changing, or deleting environment variables. Thus, the Write access type is not used in these versions of the .NET Framework. The Unrestricted state grants an assembly access to the values of all environment variables, plus some additional environment information. A summary of the information protected by the Environment permission and the corresponding permission setting required for access is shown in Table 3.

UNCLASSIFIED

Protected information	Required Environment permission entry
Environment variable values.	(Read, <variable name>) for access to the value of <variable name> Unrestricted for access to all values.
Expansion of environment variables. This refers to the ability to replace defined environment variables that are embedded in a string with their current values. The embedded environment variable names must be “quoted” with the percent sign ‘%’.	Unrestricted
Command line of the application which loaded the current assembly. Note that in Windows 98 and Windows ME, the command line includes the full path of the application executable, a potentially much more sensitive piece of information.	(Read, “Path”) Note that the value of the PATH environment variable is not actually used to get the command line – it is the existence of this permission that is used to authorize access to the command line data.
NetBIOS name of the current host, or the NetBIOS name of the virtual server if this computer is a node of a cluster.	Unrestricted
Stack trace data. This consists of the sequence of software procedure calls that have been made to this point in the code, excluding procedures that have completed and returned control back to their callers. The stack trace begins with the application’s starting procedure, includes the currently executing procedure (i.e., the procedure that requested the stack trace data), and ends with the code that actually gets the stack trace.	Unrestricted
Local host or network domain containing the account of the currently logged-on user. If the user’s logon name is the name of a local machine account, then the value of this property is the name of the local host computer. If not, the value is the name of the network domain that contains the user’s account.	(Read, “UserDomainName”) when accessed through the System.Environment object. When accessed through the System.Windows.Forms.SystemInformation object, Unrestricted access is required.
User logon account name associated with the current thread.	(Read, “UserName”) when accessed through the System.Environment object. When accessed through the System.Windows.Forms.SystemInformation object, Unrestricted access is required.
Amount of physical memory mapped to the current process (this value is not available in Windows 98 or Windows ME).	Unrestricted
Logical drives on the local computer.	Unrestricted

UNCLASSIFIED

Protected information	Required Environment permission entry
Boot mode (Normal, Safe mode, Safe mode with networking)	Unrestricted
Debug or release version of the Windows operating system.	Unrestricted
Security Model. This value indicates whether the NT Security Model is in use. This model implements discretionary access control by associating ACLs with NTFS files, registry keys, and other objects and checking access to protected objects with a privileged operating system component (the Security Reference Monitor). This model is only in use in the NT family of operating systems (Windows NT/2000/XP/Server 2003).	Unrestricted
Temporary Folder. This is the full path of a folder used by Windows to store temporary files for the current process. In Windows NT/2000/XP/Server 2003, this is the first value found in the sequence: "TMP" environment variable, "TEMP" environment variable, "USERPROFILE" environment variable, and the Windows directory (i.e., "C:\WinNT"). This last data source is not tied to any environment variable. Note that in the Windows NT family, the value of the environment variables does not have to be an existing folder. In Windows 95/98/ME, the sequence is: "TMP" environment variable, if the value is an existing folder, "TEMP" environment variable, if the value is an existing folder, and the current directory.	Unrestricted Note that this value could also be made accessible by granting the more restrictive permission of Read access to specific environment variables.
Value of the WSID (Workstation ID) connection string component that was used for a prior connection to a Microsoft SQL Server database. This optional component identifies the name of the local computer making the connection. If the "wsid=<...>" component was not present in the connection string, then the NetBIOS name of the current host (or the NetBIOS name of the virtual server if a node of a cluster) is used.	Unrestricted

Protected information	Required Environment permission entry
<p>Default system credential associated with the current thread. What is protected is an object that represents the default network credential (username, password, domain name) that can be used, for example, with HTTP Web requests requiring authentication. The system credential data itself are not viewable by assemblies.</p>	<p>(Read, "USERNAME")</p>
<p>Credential data for arbitrary network credential objects. A network credential consists of a username, password, and domain name. The username and domain attributes are protected by the Environment permission. The password attribute is protected by the much stronger Allow Calls to Unmanaged Assemblies flag of the Security permission. Note that the system credential consisting of the logged-on user's account name, password, and account domain name are not accessible through the default credential object. Even with the noted permissions, these attributes are merely empty placeholders that are populated as needed when a Web request is actually submitted.</p>	<p>(Read, "USERNAME") to access the username attribute (Read, "USERDOMAIN") to access the domain attribute</p>

Table 3. Information Protected by the Environment Permission.

Because the environment contains sensitive information about the local host, Unrestricted access should not be granted indiscriminately. However, it is common for applications to use environment variables or other platform information to customize their execution for the local environment. Since this use varies from application to application, it may be difficult to define generic environment permissions that would satisfy the needs of many applications. The principle of least privilege requires that access be granted only to the environment variables (or resource names) used. This can be achieved by granting the Environment permission through a Code Group whose Membership Condition is based on a strong name, including the software vendor's public key and the name of the application but not the version number. By not including the version in the Membership Condition, this Code Group would still apply to new versions of the application. By including the application name, the permissions granted would only apply to the specific application, not to all code originating from that software vendor. Unrestricted access should only be granted to code that is from a highly trusted source, either of local origin or from an external party with a prior trust relationship (identified by a strong name using a key known to be associated with a trusted entity).



Recommendation: The Environment permission with Unrestricted access should be granted only to highly trusted code.

Event Logs

The event logging facility in Windows provides a means for disparate applications, device drivers, or operating system processes to combine standardized error and event reporting into a small number of logs that can be easily monitored by an administrator. Windows provides three built-in event logs (Application, System, Security), and, for Windows 2000 and higher, the ability to define custom event logs. Since event logs contain sensitive system state information, their access is controlled by operating system security. For example, an application must be running under an administrator account to write to the System event log, or to clear the Application or System event logs. The Security event log is intended for use by the operating system itself, although policy settings can grant a user account the privilege of accessing Security log data.

The Event Log permission controls access to event logs through the specification of a host computer and an access type of None, Browse, Audit, Instrument, or Unrestricted. The specified access is granted to all the event logs registered for the specified machine. Access control cannot be configured separately for individual event logs.

Browse access allows an assembly to determine what event logs are defined for a given computer, either by browsing a list of defined event logs, or checking the existence of a specified event log name. The list of defined event logs for a given machine will contain the built-in logs (Application, System, and Security), as well as any custom event logs. Browse access also allows an assembly to discover the display name of an event log. This is the text that is used as the label for its node in the Console Tree pane of the Event Viewer Microsoft Management Console snap-in (`eventvwr.msc`). No access type, including Unrestricted, allows code to browse a list of all the sources registered for each event log; however, Browse access permits an assembly to determine if a specified source is registered on a given machine and to which event log it is associated. Browse access is included in the Audit, Instrument, and Unrestricted access types, so the limited information available only through Browse access is appropriate for limited administrative tools that report on local or remote system configurations.

Audit access is designed to allow some administrative control over event logs without the ability to create or modify individual log entries. Assemblies with Audit access may read existing event log entries, clear or delete an entire event log, or receive notification when a new entry has been appended to an event log. This represents a limited supervisory control over the event logging mechanism rather than the ability to act as a source of event information. Control is limited in that the CLR libraries do not currently support the creation of custom event logs, or the creation of backup files when clearing event logs. Since access can only be granted at the granularity of a host machine, Audit access provides a window into the behavior and configuration of the host system and many of its installed applications. It is inappropriate to expose this data to applications that do not perform administrative monitoring functions or are not of trusted origin. Code that is granted Audit access is trusted to read and handle system and application state information that can be gleaned from event

log entries. This information may include notification of the termination or malfunction of security controls or information about new system configurations. Thus, Audit access is appropriate only for administrative tools from trusted developers that monitor system and application events.

Instrument access provides an assembly the ability to act as a source of event information for any existing event log on the specified host machine, subject to operating system security rules. Instrument access includes Browse but not Audit access, so assemblies with Instrument access may write to an event log, but not view the entries in the log. Note that the Unrestricted access state includes both Instrument and Audit access. The Instrument access type is safe for most code.



Recommendation: The Event Log permission with Audit or Unrestricted access should be granted only to administrative tools from trusted developers that monitor system and application events.

Registry

The CLR grants access to registry keys or hives via the Registry permission. As always, access is subject to the privileges of the Windows user account under which the managed code is being executed. Access types in the Registry permission include NoAccess, Read, Write, Create, AllAccess, and Unrestricted. Permissions are inherited by subkeys and values, e.g., Read access to a registry key includes permission to read any values in that key as well as Read access to any subkeys. Read access allows an assembly to enumerate its subkeys and values and to open subkeys for read-only access. Write access permits code to modify existing values, to open existing subkeys for writing, and to delete registry keys and values. Create access allows code to create new subkeys and values.

Read, Write, and Create access types are independent: Create access does not grant code the ability to modify existing values or to delete keys or values. Write access, while allowing code to overwrite the contents of an existing value, does not include the ability to read the original contents, or even to enumerate the existing values by name. AllAccess is the combination of Read, Write, and Create. Unrestricted is equivalent to AllAccess to all registry keys.

In keeping with the principle of least privilege, Registry access should be granted only to the specific registry objects that are necessary for an assembly to function.



Recommendation: Grant the Registry permission with the most restrictive access type and to the most restrictive set of registry keys possible.

Directory Services

Access to the information stores known as directories is provided through Active Directory Services Interfaces (ADSI), a Windows facility that presents a unified approach to querying and managing information stored in local or remote directories. The software libraries that implement ADSI for use by applications are called directory services providers. A provider

hides the underlying structure of the data as it is stored, and presents the data as a hierarchical tree whose nodes represent objects with associated properties and child objects. Data stored in a directory is specified by a text path that describes the node of the directory tree containing the data. The format of a directory node path varies from provider to provider. The combination of a provider name and a directory node path is known as ADsPath syntax and has the form <provider name>://<path>. Provider names are case-sensitive, but paths are not.

The .NET Framework controls access to specific directory nodes by specifying an access type (None, Browse, and Write) and a path to a directory node in ADsPath syntax. Browse access allows managed code to read the properties of the specified node and traverse the directory tree hierarchy that begins with the specified node. Write access includes Browse access and also allows properties and child nodes to be created, modified, or deleted. It is also possible to grant Unrestricted access to all directory services.

The Windows operating system uses directory structures to store critical system and network information such as domain data in Active Directory or Web and FTP server information in the Microsoft Internet Information Services (IIS) metabase. In addition, applications may use directories to store configuration settings or other information. The type of information generally available in the Windows system directories should be restricted to use within an organization. Thus, Browse access to system directories should be granted only to assemblies of local origin that need to dynamically locate network resources or rely on information about domain computers and users to function. Write access to the Windows directories should be granted only to administrative tools of trusted origin. Access to application-specific directories should be granted only to code distributed by the application developer. In all cases, access should be granted to the most specific directory tree node path that an assembly needs to access.



Recommendation: Grant the Directory Services permission with the most restrictive access type and to the most restrictive set of directory node paths possible. Grant Browse access to the Windows system directory services (Active Directory/Global Catalog, IIS Metabase) only to code of local origin (evidenced by a strong name with a public key associated with a local entity). Only highly trusted administrative tools should be granted Write access to the Windows system directory services.

Summary

The .NET Framework provides access control over a wide variety of resources. The granularity of CAS permissions allows applications from sources with varying degrees of trust to be granted access commensurate with that trust, in keeping with an organization's security policy. The variety and granularity of CAS settings adds a great deal of complexity to the task of protecting users from code that is unstable, malicious, or promiscuous in its use of resources. To administer CAS effectively, some compromises will have to be made in the grouping of permissions into Named Permission Sets and the grouping of code into Code Groups. It is unlikely that application-specific permission grants can be efficiently configured and maintained for even a small baseline set of applications. Compromises must

be based on a risk assessment that balances the principle of least privilege against scalability, with exceptions being made to adhere to specific organizational policy requirements.

The use of strong names to associate code with a known source is an important part of creating a scalable CAS policy solution. Note that the public keys associated with each trusted source should never be obtained by examining the properties of an assembly that claims to be from that source, but should be communicated through a secure channel. Strong names can be used to associate managed code with an ultimate source, regardless of how the code was obtained or where it currently resides. In addition, an organization may use code signing to differentiate between assemblies and applications that have different access requirements, are in different broad functional categories, or originate with different organizational components. For example, administrative tools that require a wide range of access may be given strong names using a privileged (and closely guarded) private key, while office automation applications or libraries might be strong named with a second private key, and networking libraries might be strong named with a third private key. Code Groups based on the corresponding public keys can be used to grant sets of permissions appropriate to the category of software indicated by the strong name.

Recommendations in This Section



Recommendation: Only grant the File IO access permissions Read, Write, or Append to code that is trusted not to allow unauthorized access to file system resources. Grant File IO access to the most restrictive set of files and folders possible. Do not grant File IO access to file system roots or other broadly specified resources simply because they contain a few scattered files of interest. In many cases, the File Dialog or Isolated Storage File permissions are viable alternatives



Recommendation: Grant the File Dialog permission to code that needs user-discretionary access to files and folders. Use the File Dialog permission to allow the user rather than partially trusted code to browse the file system to the desired items. Where Append access is necessary or direct file system access cannot be allowed, the Isolated Storage File permission may be a viable alternative.



Recommendation: Grant Administer Isolated Storage by User access only to highly trusted administrative tools. Grant Assembly Isolation by User/Roaming User access only to assemblies that need to use user-specific data applicable to many applications, and do not use application-specific data. Grant Domain Isolation by User/Roaming User access to all other assemblies. Note that this recommendation entails a separation of duties among assemblies: those that process data of common relevance to multiple applications should not also process application-specific data and vice versa.



Recommendation: Code with limited trust should be granted at most Safe subwindows permission. Highly trusted code that accepts user authentication information or allows the user to authorize program actions through a graphical interface should be granted at most Safe top-level windows permission.

-  ***Recommendation: The clipboard is a convenience for users who wish to change the presentation context of data or reuse data without retyping it into another application. It is a “broadcast” channel in that most software can programmatically read the contents of the clipboard and write data to it whether initiated by user input or not. Nevertheless, software should use other means to communicate and reserve the clipboard for discretionary use by the user. Read access (i.e., through the All Clipboard permission) should be reserved for highly trusted code.***
-  ***Recommendation: Grant the Type Information permission only to highly trusted code that requires access to implementation details—typically this is restricted to software engineering tools or software interoperability services. Grant the Member Access permission only to highly trusted code.***
-  ***Recommendation: Grant All Printing permission only to highly trusted code.***
-  ***Recommendation: The DNS permission should typically be granted only to code that originates from within the local network (evidenced by a strong name with a public key associated with a local entity). Code from an external entity should be highly trusted to be granted this permission.***
-  ***Recommendation: The Socket Access permission should only be granted to highly trusted code or code that originates from the local network (evidenced by a strong name with a public key associated with a local entity) and provides networking services.***
-  ***Recommendation: Grant the Web Access permission to a specified URL only to code that is denied access to information or resources that should not be shared with the remote site, or is trusted to protect resources that it can access. Unrestricted Web Access should only be granted to highly trusted code that performs networking services.***
-  ***Recommendation: The Message Queue permission should only be granted to code that originates from within the local network (evidenced by a strong name with a public key associated with a local entity) or from a highly trusted external entity. Administer access to any queue and Browse access to all queues should only be granted to highly trusted administrative tools.***
-  ***Recommendation: Grant the Service Controller permission for a Windows service only to assemblies whose trust is as high as the service itself and commensurate with the value of the availability of the service.***
-  ***Recommendation: Grant any of the database permissions only to assemblies that are highly trusted.***

-  ***Recommendation: Grant the Extend Infrastructure permission only to code that is trusted to have complete control over message processing.***
-  ***Recommendation: The Enable Remoting Configuration permission should be granted only to software from a highly trusted source. The same considerations apply that would govern the granting of Unrestricted Web access or Unrestricted network socket access.***
-  ***Recommendation: Grant Enable Serialization Formatter permission only to highly trusted code that will be considered an extension to the CLR's trusted library base.***
-  ***Recommendation: Grant Enable Thread Control permission only to Fully Trusted code.***
-  ***Recommendation: Grant the Allow Principal Control permission only to code that is trusted at least as much as the most trusted user account on the system.***
-  ***Recommendation: The Enable Assembly Execution permission should be granted based on the level of trust associated with the assembly's origin, as established by evidence stronger than URL Security Zone. If possible, separate the Enable Assembly Execution permission from resource access permissions, so that the former is tied to origin and embodies a trust relationship, while the latter are tied to functional requirements of code and embody the principle of least privilege.***
-  ***Recommendation: Skip Verification should be granted only to highly trusted code based on a hash identity or strong name evidence that includes the assembly's name, version, and public key associated to a trusted party. If possible, separate the Skip Verification permission from resource access permissions, so that the former is tied to a specific assembly from a trusted point of origin and embodies a trust relationship, while the latter are tied to functional requirements of code and embody the principle of least privilege.***
-  ***Recommendation: Allow Calls to Unmanaged Assemblies should be granted only to code that is trusted to execute with the same privileges as the CLR itself.***
-  ***Recommendation: The Allow Policy Control permission should be granted only to highly trusted .NET Framework administrative tools.***
-  ***Recommendation: If custom Runtime Host applications are in use that implement organizational policy using the AppDomain CAS policy level, then the Allow Domain Policy Control permission should be granted only to code that is highly trusted. In other cases (including the typical default installation), this permission should be granted only to code that is designed to dynamically launch other applications that may be less trusted than itself.***

-  ***Recommendation: The Allow Evidence Control permission should be granted only to code developed by trusted parties with demonstrated secure coding practices. Code granted this permission effectively becomes an extension of the CLR's access control system.***

-  ***Recommendation: The Assert any Permission that Has Been Granted permission should be granted only to software that is from a trusted developer with demonstrated secure coding practices. Typically, this permission is granted to highly trusted extensions to the CLR base libraries.***

-  ***Recommendation: Grant Performance Counter access to the most restrictive set of performance counter categories possible. Grant Instrument or Administer access only to trusted code that provides or administers a monitoring service.***

-  ***Recommendation: The Environment permission with Unrestricted access should be granted only to highly trusted code.***

-  ***Recommendation: The Event Log permission with Audit or Unrestricted access should be granted only to administrative tools from trusted developers that monitor system and application events.***

-  ***Recommendation: Grant the Registry permission with the most restrictive access type and to the most restrictive set of registry keys possible.***

-  ***Recommendation: Grant the Directory Services permission with the most restrictive access type and to the most restrictive set of directory node paths possible. Grant Browse access to the Windows system directory services (Active Directory/Global Catalog, IIS Metabase) only to code of local origin (evidenced by a strong name with a public key associated with a local entity). Only highly trusted administrative tools should be granted Write access to the Windows system directory services.***

CAS Policy

CAS policy rules enforced by the CLR can be set at four policy levels: Enterprise, Machine, User, and Application Domain. These sets of policy rules are combined to form the CAS policy for the computer. The .NET Framework security administrator sets policy rules at the Enterprise and Machine levels. Any user may further refine the policy for assemblies executing on their behalf by modifying the User level policy. Application Domain policy is set dynamically by the CLR in conjunction with information provided by the runtime host process. Application Domain policy is not administratively configurable and will not be discussed. This section will focus on policy creation and administration for the Enterprise and Machine levels, but the same principles apply to the User policy level as well. The actions that an administrator may take for the Enterprise and Machine policy levels may be

taken by a user for the User policy level. User level settings apply only to the currently logged on user, and persist across logon sessions.

The administrator sets policy rules by defining Code Groups and Named Permissions Sets. Code Groups are logical groupings of assemblies based on evidence. Named Permission Sets are predefined sets of access permissions that provide a generic, scalable way of assigning these groups one or more of the permissions described in the previous section. The rest of this section describes some of the .NET Framework CAS components and features, and how these components are combined across all the policy levels in the calculation of an assembly's Granted Permission Set.

Code Groups and Membership Conditions

Code Groups define collections of assemblies that will be granted the same permissions based on a shared characteristic. Code Groups are hierarchically arranged. An assembly cannot be a member of a Code Group unless it is a member of all parent Code Groups. An assembly belongs to a Code Group if it satisfies the administratively defined Membership Condition for that group. An assembly may satisfy the Membership Condition to more than one Code Group, and thus belong to more than one Code Group at the same time. The way permissions are granted when an assembly belongs to multiple Code Groups will be discussed in detail below.

The Membership Condition of a Code Group is typically the value or presence of a single type of assembly evidence. Thus, the available Membership Conditions closely correspond to the types of evidence an assembly may present. Exceptions to this rule are the All Code Membership Condition, and custom Membership Conditions.

- All Code: All assemblies satisfy the All Code Membership Condition.
- Hash: An assembly will satisfy this Membership Condition if its hash value using the specified algorithm (MD5 or SHA-1) matches the hash value specified in the policy. This Membership Condition may be used to create a Code Group which contains exactly one assembly. A drawback of this Membership Condition is that the hash value must be updated for each new version of an assembly. Note that the .NET Framework Configuration tool allows the administrator to browse to the location of an assembly and import its hash into the policy. This should never be done for assemblies accessed through the Internet. Instead, use a secure channel of communication to transfer the hash value or the assembly to the local host.
- Publisher: An assembly will satisfy this Membership Condition if it is signed with a software publisher's Authenticode X.509v3 digital certificate that can be verified by the Windows operating system as having a chain of trust that leads to a trusted root certificate stored in the user's certificate store. This Membership Condition can be used to identify an organization, developer, vendor, or other entity as the ultimate source of the assembly, even if the code itself was obtained from a third party such as a mirror site. Access to resources may then be granted based on the trust relationship with the identified entity. The certificate data to include in the Membership

Condition should never be obtained by importing the data from an assembly accessed through the Internet. Use a secure channel to transfer the certificate data or the assembly to the local host.

- **Strong Name:** An assembly will satisfy this Membership Condition if its metadata contains the strongly identifying data associated with the specified strong name. At the least, this means it has been digitally signed with the private key associated with the public key recorded in the policy. If specified, it may also mean it has the same file name and version recorded in the policy. Although culture is also an element of a strong name, there is no provision to restrict Code Group membership based on culture. As long as an assembly satisfies the signature, name, and version elements, any culture is accepted. The Strong Name Membership Condition is the best way to identify a particular assembly or a particular organization, developer, vendor, or other entity. By specifying a public key known to be associated with an external or internal entity, access control decisions may be made based on a prior trust relationship. By leaving the version data unspecified, the Membership Condition will remain useable for all future versions of the same assembly.

Note that the presence of a strong name itself does not provide any assurance that code is trustworthy. It is the association of the public key with a known party that provides the basis for trust. The public key data to include in the Membership Condition should never be obtained by importing the data from an assembly accessed through the Internet. Instead, a secure authenticated channel should be used to transfer the assembly or the strong name data to the local host. Note also that the strong name does not use a digital certificate format, and thus is not verifiable through a chain of trust to a root certificate like the Publisher Membership Condition. Thus, the public key used in a strong name has no associated expiration date or revocation status.

When managed software is under development, it may be in a “delay-signed” status. A delay-signed assembly contains a public key, but has not yet been digitally signed with the corresponding private key. Software developers who do not have access to an organization’s private keys may test their software under simulated operational (i.e., fully signed) conditions, and then submit the final version for digital signing by the organization’s private key custodians. To register a delay-signed assembly to simulate a fully-signed version, the assembly name and public key must be stored in a list in the registry under HKLM\Software\Microsoft\StrongName\Verification. This list is maintained with the .NET Framework Strong Name Utility (`sn.exe`). Using `sn.exe`, the list may be displayed or cleared, and individual entries may be added or removed. Wildcards may be used to specify any assembly with a given public key, or all assemblies. Registration causes the normal strong name verification process to be bypassed. Delay-signed assemblies corresponding to registered entries will satisfy the strong name Membership Condition for the public key contained in the assembly manifest. Computers used for operations must never skip the strong name verification process.



Recommendation: Strong name verification should never be simulated in an operational environment.

- **URL:** An assembly will satisfy this Membership Condition if it has been obtained from the specified URL. The URL associated with an assembly's origin is provided by the Runtime Host process that loads the assembly. URL evidence may be a Website, a UNC local Intranet path, or a local file system path. To satisfy the URL Membership Condition, the exact URL must be matched, including protocol, domain, port number (if specified), and path (including file name if specified). If a "*" is appended as a wildcard, then subpaths will also satisfy the Membership Condition. This Membership Condition should never be used with a file name to try to restrict Code Group membership to a specific assembly. Instead, use the Hash or Strong Name Membership Conditions for this purpose.
- **Site:** An assembly will satisfy this Membership Condition if it has been obtained from the specified domain or subdomain, as extracted from the assembly's URL. Assemblies obtained via a UNC path or local file system path never satisfy a Site Membership Condition. This Membership Condition is a weak means of identifying an organization, developer, vendor, or other entity. Where possible, use Publisher or Strong Name Membership Conditions.
- **Zone:** An assembly will satisfy this Membership Condition if it has been loaded from the specified URL Security Zone. This is the weakest form of identification (and conversely, the most scalable). Zone membership may be subject to manipulation outside the administrative scope of CAS policy. It should not be used for access control decisions.
- **Application Directory:** An assembly will satisfy this Membership Condition if it is a private assembly of the currently executing application, i.e., if it has been loaded from a subpath of the directory or URL associated with the current application. This Membership Condition is automatically satisfied by a managed application itself. Note that this Membership Condition does not apply to a particular application. To grant a set of permissions only to the private assemblies of a particular application, use this Membership Condition with a Code Group that is the child of a parent Code Group whose Membership Condition is satisfied only by the particular application.

The .NET Framework comes with a number of default Code Groups. The Enterprise and User policy levels have exactly one default Code Group with the All Code Membership Condition. The default Code Groups defined for the Machine level in version 1.1 of the .NET Framework are shown in Figure 14. The Internet_Same_Site_Access Code Group is not present in version 1.0 of the .NET Framework. Others may be created as needed by the administrator.

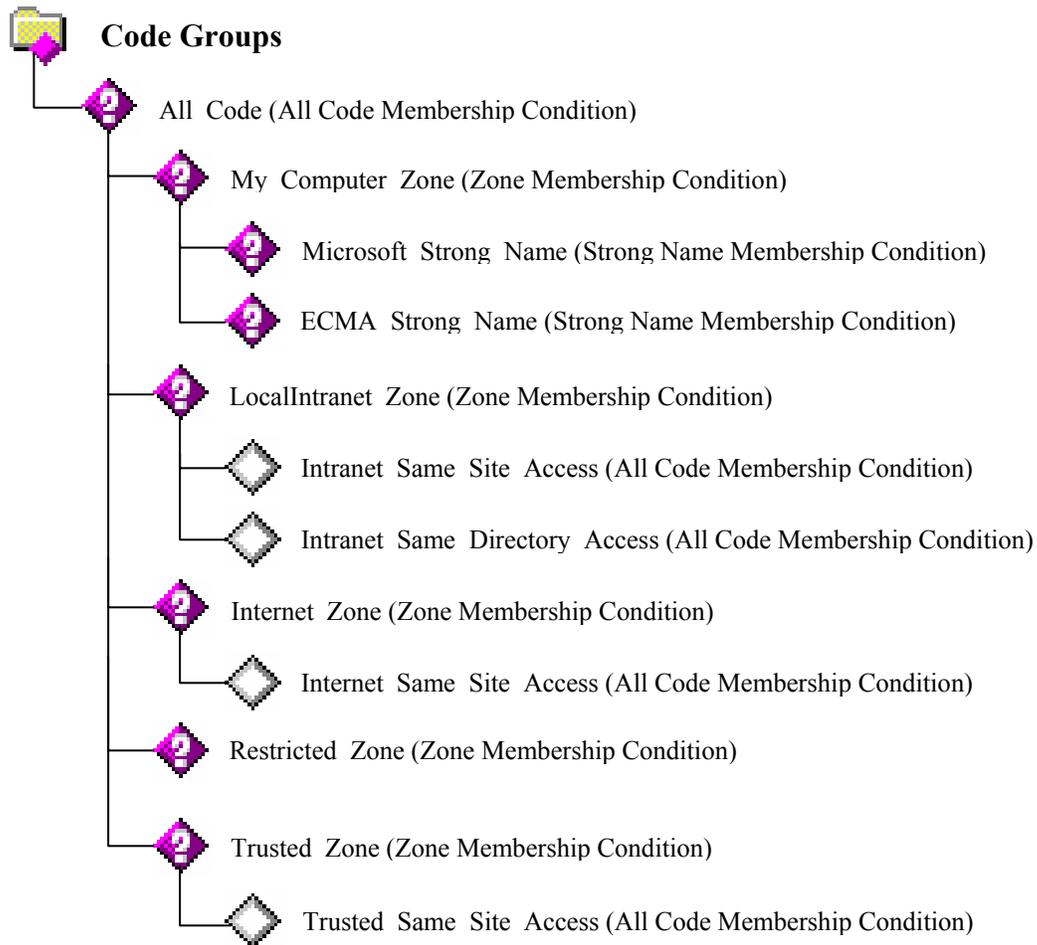


Figure 14. Default Code Groups at the Machine Level in Version 1.1 of the .NET Framework.

There are four different types of Code Group that may be defined:

- Union Code Group
- First Match Code Group
- Net Code Group
- File Code Group

Union Code Groups and First Match Code Groups

The most common type of Code Group will be the Union Code Group. This is the type of Code Group created through the .NET Framework Configuration Tool (`mscorcfg.msc`), the primary administrative tool distributed with the .NET Framework. When the CLR applies

CAS policy to determine Code Group membership for an assembly, it will walk the Code Group hierarchy for each policy level. If the assembly fails to satisfy the Membership Condition of a Code Group, none of that Code Group's child groups will be checked. If the assembly satisfies the Membership Condition of a Union Code Group, the Membership Conditions of every child Code Group will be checked. This is a recursive procedure, so satisfying the Membership Condition of a child Code Group will continue the process down the Code Group hierarchy.

If the assembly satisfies the Membership Condition of a First Match Code Group, the Membership Conditions of the child Code Groups will be checked, but this process will stop when the first child Code Group is encountered whose Membership Condition is also satisfied by the assembly.

A tree of Union Code Groups will resolve to the set of all paths of Code Groups in the tree that correspond to Membership Conditions satisfied by the assembly. A tree of First Match Code Groups will resolve to a single path consisting of the root Code Group and exactly one of its child Code Groups, and exactly one of that child Code Group's children, etc., until no child Code Group's Membership Condition is satisfied.

`mscorcfg.msc` cannot be used to create First Match Code Groups. These must be created by editing the CAS policy files. Since this may result in a corrupt or invalid policy file, it is recommended that CAS policy be configured using Union Code Groups. An unparseable CAS policy file will result in the default CAS policy being applied, which may not be consistent with local security policy.



Recommendation: Editing CAS policy files to use First Match Code Groups may create invalid or corrupt XML, as these files are also modified by automated tools and parsed by the CLR. Thus, it is recommended that CAS policy be configured using Union Code Groups configured through `mscorcfg.msc`.

File Code Groups and Net Code Groups

File Code Groups and Net Code Groups behave the same way as Union Code Groups with respect to how child Code Groups are evaluated for assembly membership. The default Code Groups contain one File Code Group (`Intranet_Same_Directory_Access`) and three Net Code Groups (`Intranet_Same_Site_Access`, `Internet_Same_Site_Access`, and `Trusted_Same_Site_Access`).

A File Code Group grants access to the directory from where the assembly was loaded and its subdirectories. This is not the same as the Application Directory Membership Condition, which is satisfied if the assembly under consideration is being loaded from the same directory (or a subdirectory) of another assembly. The File Code Group grants permission to a single assembly to the directory from which it was loaded. The default File Code Group (`Intranet_Same_Directory_Access`) uses the All Code Membership Condition, so it relies on the parent Code Group to determine the conditions under which this access is granted. `Intranet_Same_Directory_Access` grants Read and Path Discovery access to the source

directory. These permissions were discussed above in the .NET Framework Protected Resources section under the File IO permission.

A Net Code Group grants the ability to initiate connections to the Website from which the assembly was downloaded. The same protocol must be used, with the exception that an assembly downloaded using the HTTP protocol may connect to its site of origin using either HTTP or HTTPS. Even though the assembly may have been downloaded from a subpath of the root URL of the Website, connections may be made to any URL within that Website. For example, if an assembly `myapp.exe` was downloaded from `http://www.example.com/↓ apps/new/myapp.exe` and satisfies a Net Code Group's Membership Condition, it will have the permission to connect to any URL in the `www.example.com` Website, using either HTTP or HTTPS. The default Net Code Groups (`Intranet_Same_Site_Access`, `Internet_Same_Site_Access`, `Trusted_Same_Site_Access`) use the All Code permission, so they rely on their parent Code Group to determine the conditions under which the connect access is granted.

Copying File Code Groups and Net Code Groups

File Code Groups and Net Code Groups cannot be created directly through `mscorcfg.msc`, but the existing default ones may be safely copied using this tool through the following steps:

- Right click on the File Code Group or Net Code Group and click **Properties** from the context menu.
- Select the **Custom Code Group** tab. This will display the XML representation of the Code Group. The display for the `Intranet_Same_Site_Access` Net Code Group is shown in Figure 15.

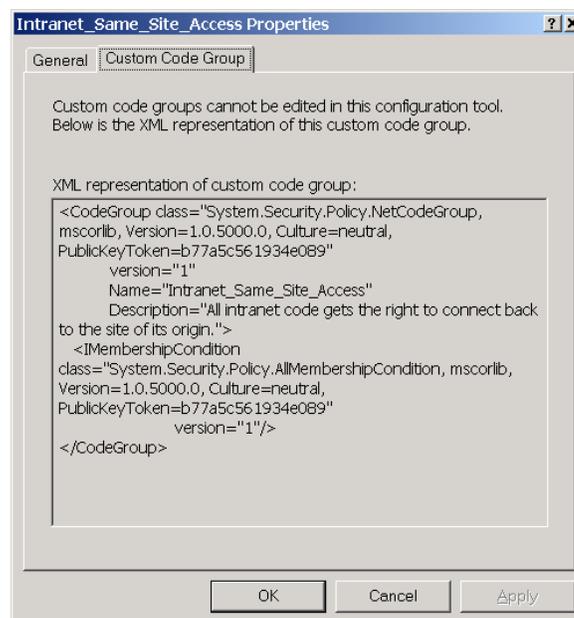


Figure 15. Custom Code Group Tab for `Intranet_Same_Site_Access`.

- Select the XML text and copy it to a text file.
- Click **Cancel**.
- Right click the Code Group under which you want to create a child File Code Group or Net Code Group, and select **New...**

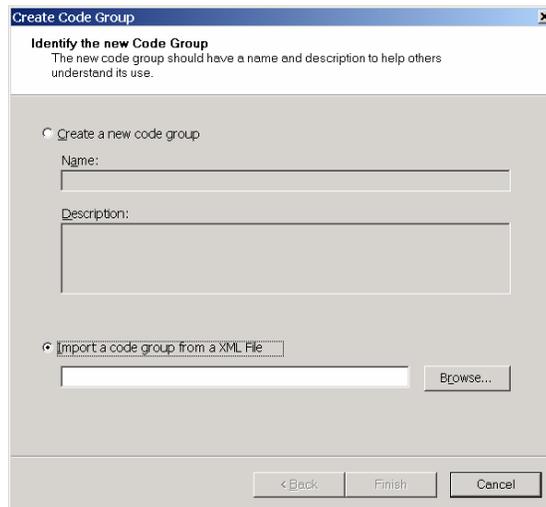


Figure 16. Create Code Group Dialog Box.

- In the Create Code Group dialog box (shown in Figure 16), select the **Import a code group from a XML file** radio button.
- Click **Browse...** and select the text file in which the XML of the source Code Group was saved.
- Click **Finish**.

This is a safe way to copy File Code Groups and Net Code Groups and is the recommended technique for adding such groups.



Recommendation: Editing CAS policy files to create File Code Groups or Net Code Groups may create invalid or corrupt XML, as these files are also modified by automated tools and parsed by the CLR. Thus, it is recommended that these groups be avoided or the XML of the default groups be copied and imported using mscorcfg.msc.

Named Permission Sets

A Named Permission Set is a predefined set of permissions granting access to resources. Named Permission Sets are created in advance by the administrator and stored in the CAS policy files. Each Union Code Group or First Match Code Group has an associated Named Permission Set that defines the access rights granted to members of that Code Group. The

File Code Groups and Net Code Groups do not have associated Named Permission Sets; instead, they have specific built-in permissions associated with them. A Named Permission Set may be associated with multiple Code Groups. The default CAS policy has several predefined Named Permission Sets that cannot be changed by the administrator. These are shown in Table 4.

Default Named Permission Sets
<p>Nothing</p> <p>The Nothing Named Permission Set is associated with the All_Code and Restricted_Zone default Code Groups at the Machine policy level in the .NET Framework version 1.1. In version 1.0, Nothing is also associated with the Internet_Zone Code Group at the Machine policy Level. By its association with the All_Code Code Group at any level, this sets the default permissions granted to code. In order for code to be granted any permissions, it must satisfy some Membership Condition associated with another Code Group. This Named Permission Set grants no access to any resource.</p>
<p>Execution</p> <p>The Execution Named Permission Set is not associated with any default Code Group. This Named Permission Set only includes the single permission for code to execute. No other access to resources is granted. This is the weakest Named Permission Set that grants anything at all. In most cases, this Named Permission Set must be combined with others granted through other Code Groups in order for an assembly to perform its intended function.</p>
<p>Internet</p> <p>The Internet Named Permission Set is associated with the Internet_Zone default Code Group in .NET Framework version 1.1 and the Trusted_Zone default Code Group in .NET Framework versions 1.0 and 1.1 (all at the Machine policy level). Permissions in this Named Permission Set include:</p> <ul style="list-style-type: none">• Code is allowed to execute.• Code is allowed to read files specified by the user through the Open File dialog box.• Code may use the Isolated Storage facility to read and store up to 10,240 bytes of data associated with the user account and application that are executing the assembly. Multiple assemblies with this same permission may share data when executing in the context of the same application.• Code may print to a printer selected by the user through a Print dialog box.

Default Named Permission Sets

- Code may create windows on the desktop, but may not control some aspects of their appearance.
- Code may put data on the Windows clipboard, but may only copy data from the clipboard through a user action such as Ctrl-V.
- If code was downloaded from a URL, it is allowed to initiate connections to the originating Website.

LocalIntranet

The LocalIntranet Named Permission Set is associated with the LocalIntranet_Zone default Code Group. Permissions in this Named Permission Set include:

- Code is allowed to execute.
- Code is allowed to read and write files specified by the user through the Open File and Save File dialog boxes.
- Code is allowed to use the Isolated Storage facility to store virtually any amount of data associated with the assembly running under the current user account.
- Code may print to a printer selected by the user through a Print dialog box, or it may send print jobs to the system default printer.
- Code is allowed to access DNS information such as IP addresses and host names, however, the ability to make network connections must be granted through another Named Permission Set.
- Code may control all aspects of the desktop windows that it creates.
- Code may cut and paste without restriction to and from the system clipboard.
- Code may discover the value of the USERNAME Windows environment variable.
- Code is allowed to write new code and then execute it on the fly. Unless an assembly is also granted the Allow Evidence Control permission, code created dynamically by the assembly inherits the permissions of its creator.
- Code may write entries to event logs on the local host. Typically this will be the Windows Application event log, but may include custom event logs as well. This permission does not allow code to read the entries in the event logs.

Default Named Permission Sets
<ul style="list-style-type: none"> Code may access any resource that it has been granted permission to access, even if it has been invoked by code that is less trusted. This permission allows an assembly to access resources it needs to perform its function, but that will not be exposed for discretionary access to other code.
<p>SkipVerification</p> <p>The SkipVerification Named Permission Set is not associated with any default Code Group. This Named Permission Set includes only a single permission: code is allowed to skip the verification process when it is loaded that confirms that it is a well-formed program that can be reliably managed by the CLR. If code fails this process, it is not allowed to execute. Unfortunately, some compilers create safe and well-behaved code that will nevertheless fail the verification process, so this permission is sometimes necessary to allow code from a trusted origin to run.</p>
<p>Everything</p> <p>The Everything Named Permission Set is not associated with any default Code Group. Contrary to its name, the Everything Named Permission Set does not grant every possible permission. The CAS system is extensible through the creation of custom libraries that define new evidence types, permission types, and Code Group types. The Everything Named Permission Set includes all of the built-in permissions except for the ability to skip code verification, and will also not include any access to any custom permission. Each permission included in the Everything Named Permission Set grants unrestricted access to its associated resource.</p>
<p>FullTrust</p> <p>The FullTrust Named Permission Set is associated with the My_Computer_Zone, Microsoft_Strong_Name, and ECMA_Strong_Name default Code Groups at the Machine policy level, and the All_Code default Code Group at the Enterprise and User policy level. This is the true “everything” permission set in that it allows unrestricted access to all resources.</p>

Table 4. Default Named Permission Sets in the .NET Framework Version 1.1.

As the CLR walks the Code Group tree to apply CAS policy for an assembly, it will check the assembly’s evidence against the Membership Conditions of the Code Groups it encounters. When an assembly satisfies the Membership Condition for a Code Group, the assembly is granted the permissions contained in the Code Group’s associated Named Permission Set. If an assembly is a member of multiple Code Groups, the assigned permissions are calculated as a combination of the permissions in all the associated Named Permission Sets. Figure 17 illustrates the process by which an assembly is assigned permissions from more than one Code Group.

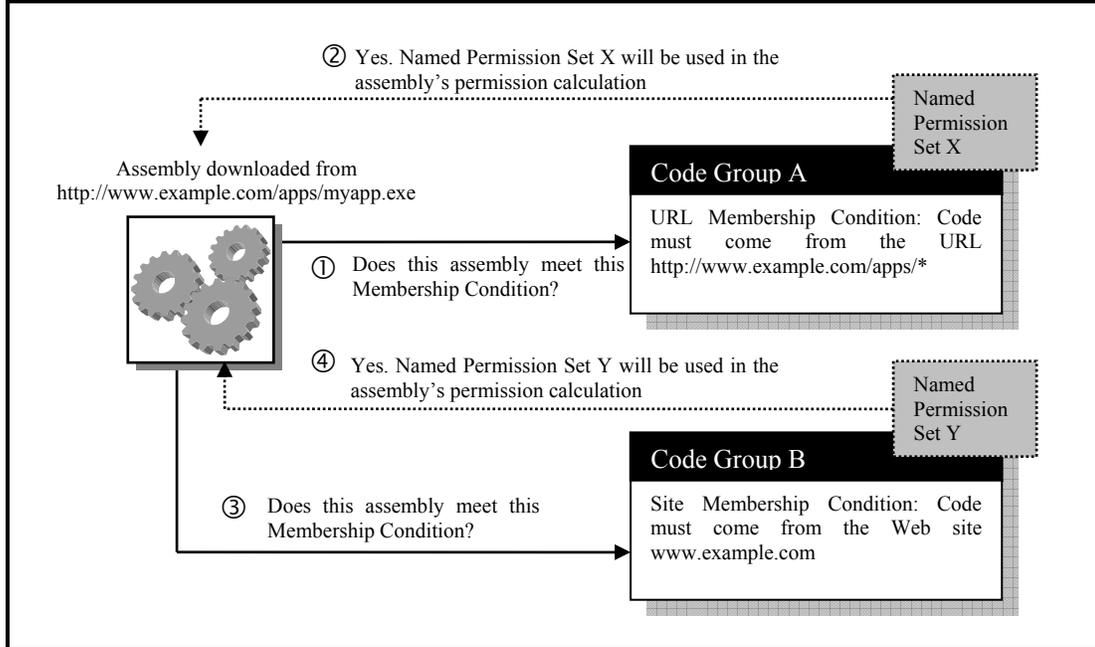


Figure 17. Assigning Assembly Permissions.

Policy Levels

Each Code Group or Named Permission Set is defined for only one policy level. For purposes of permission resolution, these policy levels are hierarchically related. The order of precedence from highest to lowest is Enterprise, Machine, User, and Application Domain. Higher policy levels cannot be overridden by lower policy levels. A lower policy may only further restrict code.

The Enterprise and Machine policy levels are the only two administratively configured levels. The Enterprise policy level may be used to configure CAS policy for an entire organization, with host-specific rules defined in the Machine policy level. However, there is no provision as of .NET Framework version 1.1 for the CLR to load the Enterprise CAS policy from a network server, so both the Enterprise and Machine policy levels must be deployed machine by machine.

The default Enterprise CAS policy grants unrestricted access to all code. Any restrictions placed on code at this highest level cannot be overridden by lower policy levels, so restrictions placed here should be few and carefully targeted. The Machine level policy is used to further refine CAS policy to suit the needs of a single host. The default CAS policy has a complex structure at this level.

The User level policy may add additional restrictions to code. This policy level is customizable by each logged on user. The Application Domain policy level is not

configurable. It is created by the CLR when an Application Domain is created to host a managed application.

Permission Resolution

When an assembly is loaded, the CLR determines the assembly’s permissions to access resources. This is done by computing three types of permission sets:

- Level Permission Set
- Allowed Permission Set
- Granted Permission Set

Level Permission Set

The Level Permission Set is determined by walking each policy level’s Code Group tree in accordance with the rules for Union Code Groups and First Match Code Groups, evaluating the assembly’s membership in each Code Group, and combining the associated Named Permission Sets for each Code Group containing the assembly as a member. This produces a set of permissions for that assembly that is the union of the assembly’s associated Named Permission Sets, i.e., the set of all permissions present in any of the assembly’s Named Permission Sets, even those missing in another Named Permission Set. Figure 18 illustrates the creation of the Level Permission Set.

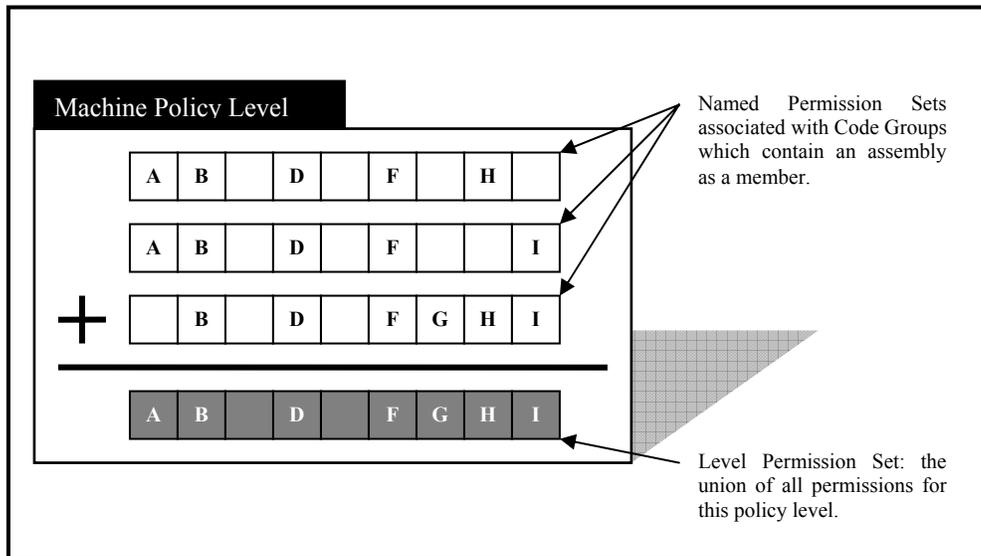


Figure 18. Creation of a Level Permission Set.

In Figure 18, since permissions C and E are absent from all three Named Permission Sets, they are excluded from the Level Permission Set for the Machine policy level. Note that although permission A is absent from one of the Named Permission Sets, it is present in at

least one Named Permission Set and so it is included in the Level Permission Set for this policy level.

Allowed Permission Set

The Allowed Permission Set is created by intersecting the Level Permission Sets for each policy level. This creates a set of permissions that contains each permission present in every one of the Level Permission Sets. Figure 19 illustrates the creation of the Allowed Permission Set.

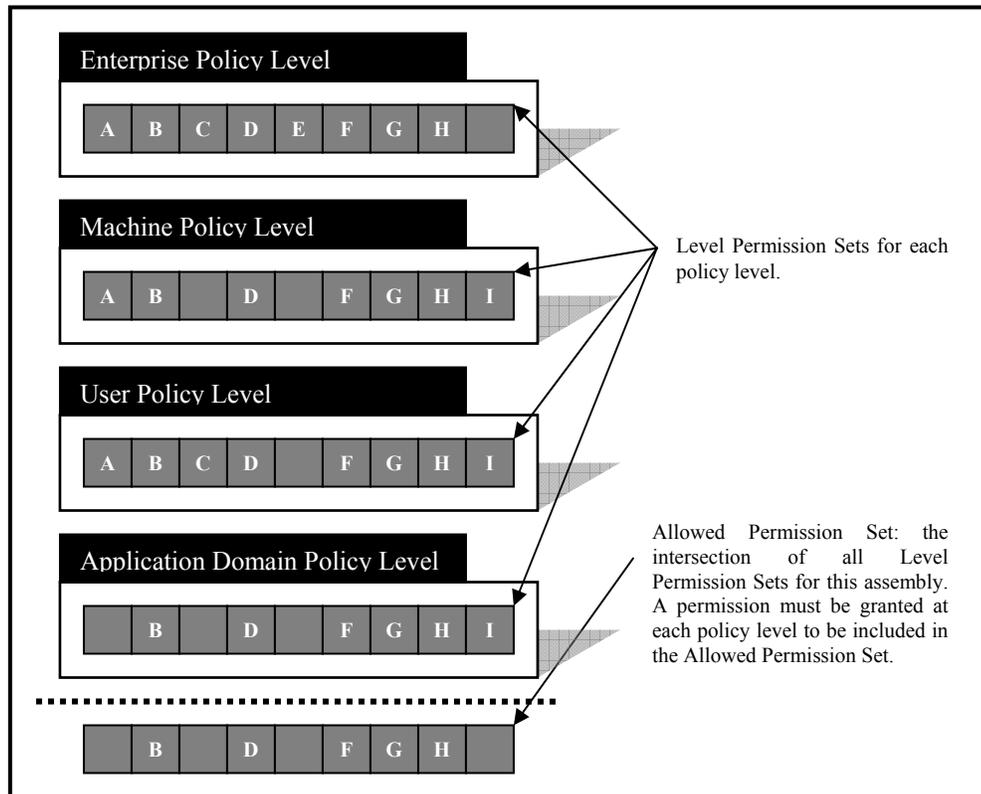


Figure 19. Creation of the Allowed Permission Set.

In Figure 19, since permissions A, C, E, and I are absent from at least one Level Permission Set, they are not included in the Allowed Permission Set. Intersecting the Level Permission Sets prevents the settings of any policy level from granting permissions that are denied at any other level. The Allowed Permission Set is the largest set of permissions available to the assembly.

Granted Permission Set

The Granted Permission Set is the set of permissions that are actually associated with an assembly during its execution. This set is the basis for access control decisions made by the CLR. The Granted Permission Set is created by using the permission requirements expressed by the assembly itself (Assembly Permission Requests) to further restrict the Allowed Permission Set.

Assembly Permission Requests

In its metadata, an assembly may explicitly state what set of permissions it requires in order to minimally function (its Minimum Request), what set of permissions it would prefer to have to be able to offer all of its features (its Optional Request), and what set of permissions it refuses, even if granted by CAS policy (its Refuse Request). These three types of permission requests may further restrict the permissions associated with an assembly, but will never increase the permissions beyond the Allowed Permission Set, i.e., permissions are only granted on the basis of CAS policy settings, but they may be taken away if they are not needed as determined by the assembly permission requests. Assembly permission requests are specified by the developer of the software and are not configurable by the administrator. They are a means for developers to communicate to consumers of the code what resources the code expects to access. The administrator may view an assembly's permission requests using `permview.exe`.

Computation of the Granted Permission Set

The Granted Permission Set is computed by the following sequence of steps:

- If an assembly has specified a Minimum and/or Optional set of permissions, then all permissions not in either the Minimum or Optional requests are removed from the Allowed Permission Set. These permissions are presumed not to be needed by the assembly, and therefore will not be granted, in keeping with the principle of least privilege. If the assembly has not specified either a Minimum or Optional set, then the Allowed Permission Set is not reduced.

Thus, if any Minimum or Optional permissions are explicitly requested, all other permissions are considered to be implicitly refused. If no Minimum or Optional permissions are explicitly requested, all permissions are considered to be implicitly requested as Optional.

- The modified Allowed Permission Set is checked for the permission to execute. This is a performance-enhancing measure. If not allowed to execute, the CLR will not load the assembly and will cause an exception to occur. Note that permission to execute does not need to be explicitly included in the Minimum permission set. It is considered an implicit requirement.
- The modified Allowed Permission Set is checked for the presence of all the permissions in the Minimum set. If the permissions have not all been granted, the CLR will not load the assembly and will cause an exception to occur.
- If the assembly has specified a Refuse Request, those permissions are removed from the modified Allowed Permission Set (even if they were also included in the Minimum or Optional sets). The result is the Granted Permission Set.

If an assembly does not make any permission requests, then the Granted Permission Set is simply the Allowed Permission Set. In no case will an assembly's permission requests cause the addition of permissions to the Allowed Permission Set.

Although developers may use assembly permission requests to adhere to an organizational policy about resource usage, a security policy with respect to code privileges should be implemented with a sound CAS policy configuration, and not based on software self-regulation.



Recommendation: Although software developers should implement the principle of least privilege through assembly permission requests, security policy should not rely on these requests, but should be implemented through CAS policy settings.

Code Group Attributes

The permission resolution process described above can be affected by two Code Group attributes, Exclusive and Level Final. These attributes are set by the administrator (for the Enterprise and Machine levels) or the user (for the User level) when the Code Groups are created.

Exclusive

The Exclusive attribute affects the creation of a Level Permission Set. If an assembly satisfies the Membership Condition of a Code Group marked Exclusive on a given policy level, the Level Permission Set for that level will consist of exactly the Named Permission Set associated with that Code Group and no other permissions. Child Code Groups of the Exclusive Code Group will also not contribute permissions. More than one Code Group on a given policy level may have the Exclusive attribute, but an assembly will not be allowed to execute if it belongs to more than one code group with this attribute.

Since the Exclusive attribute prevents the addition of other permissions on the same policy level, it is used to set a maximum Level Permission Set for code. Since the Level Permission Sets are intersected over all the policy levels, this is also a maximum for the entire CAS policy. This is useful for selectively restricting or disallowing software execution.

Level Final

The Level Final attribute affects the creation of the Allowed Permission Set. If an assembly satisfies the Membership Condition of a Code Group marked Level Final on a given policy level, the Allowed Permission Set will be created using only the Level Permission Sets for this level, higher levels, and the AppDomain level. Other than the AppDomain level, CAS policy levels below this one will be ignored. More than one Code Group on a given policy level may have the Level Final attribute, and an assembly may belong to multiple Code Groups that have this attribute. Note that the Level Final attribute does not affect the creation of the Level Permission Set for the level on which it is set. Note also that marking a Code Group Level Final will have no effect on permissions granted to assemblies that are not members of that Code Group.

The Level Final attribute on the Enterprise policy level is a means of granting a minimum set of permissions on an assembly, as lower policy levels can only further restrict the permissions granted to an assembly. On the Machine policy level, it prevents the user from further restricting the permissions granted by the Enterprise and Machine level policies.

This attribute should only be used if both of the following conditions are true:

- The Membership Condition of the Code Group receiving this attribute is narrowly tailored, such as by a hash or strong name. Since the Level Final attribute moves counter to security in preventing the tightening of restrictions, the Membership Condition of the Code Group should be as discriminating as possible.
- The functionality of the targeted assemblies must be assured. The Level Final attribute supports availability by preventing inadvertent modifications in the Machine or User policy from affecting the granted permissions. This particularly applies where the default Machine or User CAS policy would prevent the assemblies from functioning, since a missing or corrupted CAS policy configuration file at the Machine or User policy level results in the default policy being applied at that level.

Furthermore, this attribute, although “simplifying” the CLR’s internal policy resolution process, may actually increase the chance of security lapses, as the combined policy becomes more complex and difficult for administrators to assimilate. Good communication and coordination of policy changes is particularly important among all those responsible for maintaining the Enterprise and Machine policy levels, or the combined CAS policy may become unpredictable.



Recommendation: Code Groups with the Level Final attribute should have Membership Conditions that are as narrowly defined as possible.

Policy Enforcement

Once an assembly’s evidence has been evaluated against CAS policy and the Granted Permission Set determined, the CLR and its libraries enforce the Granted Permission Set by responding appropriately to access rights queries made by code. As mentioned in chapter 1, an assembly can choose whether or not to query the access monitor of the CLR, and if it does, whether or not to pay attention to the results. The CLR’s supporting libraries, which mediate access to resources on behalf of all other managed code, are designed to refuse to fulfill any request for resources after a failed access check.

The process of checking the access rights of an assembly is performed through a stack walk, which checks the Granted Permission Set of every assembly in the current logical path (stack) of code execution, all the way back to the original managed console, Windows GUI, or Web-based application. Every assembly in the logical path of code execution must have permission to access the desired resource in its Granted Permission Set, or the CLR will report a denial of access to the assembly requesting the access rights check. Figure 20 illustrates the stack walk.

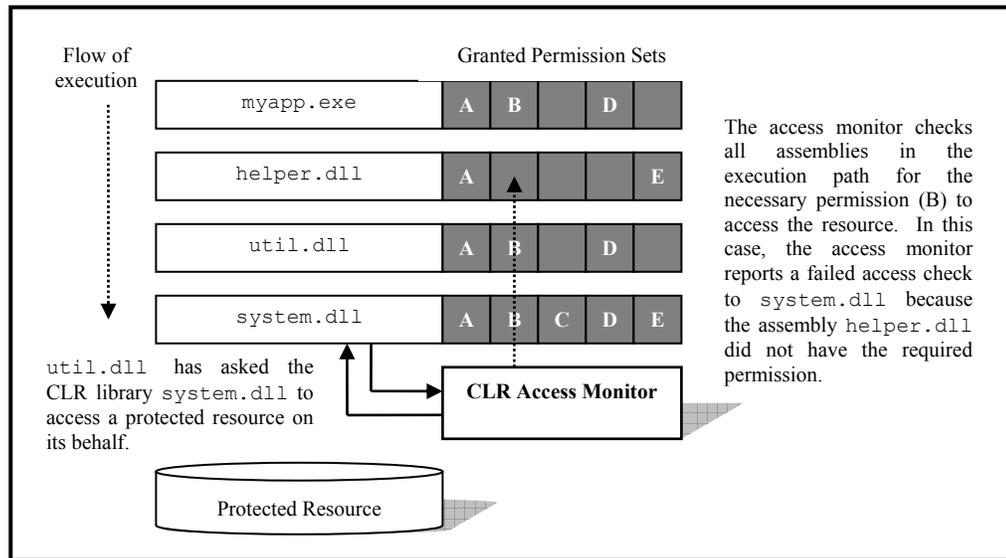


Figure 20. The Stack Walk.

In Figure 20, an application `myapp.exe` has loaded assemblies `helper.dll`, `util.dll`, and the CLR's library assembly `system.dll`. During execution, `myapp.exe` has invoked code in `helper.dll`, which in turn has invoked code in `util.dll`. Since access to resources is performed by calling the CLR's library functions, `util.dll` requests access to a protected resource through `system.dll`. Since `system.dll` is a trusted component of the .NET Framework, it will first check the CLR Access Monitor to verify that all of the assemblies in the calling stack have the required permission (in this case, permission B). The CLR's access monitor checks each assembly in turn, determines that `helper.dll` does not have permission B in its Granted Permission Set, and returns a failure response to `system.dll`.

The stack walk prevents privileged code from being tricked into accessing resources on behalf of less privileged code. In this case, because `helper.dll` does not have permission B in its Granted Permission Set, it cannot access the resource itself or ask a more privileged assembly to access the resource on its behalf. `util.dll` and `system.dll` can both be used by assemblies that do not have permission B without danger of exposing the resource to unauthorized access.

Summary

CAS policy is a means of granting access permissions to code based on evidence that the code itself presents, rather than based only on the Windows user account in the context of which the execution occurs. The policy is administratively configured through the creation of Code Groups and associated Named Permission Sets. The process of determining the permissions granted to a specific assembly is complex, so creation of CAS policy should rely on a consideration of the effect of the entire policy, not just on one Code Group or policy level.

Technology and policy can be mutually constraining. Existing technology can give policy creators “tunnel vision,” that is, policy specifications can be written, whether consciously or subconsciously, with existing technical controls in mind. Conversely, an organizational security policy constrains, in part, what technological measures can be employed to implement its specifications. The .NET Framework is a new technical control that enables more fine-grained policy implementation, while providing additional flexibility in policy specification as well.

Recommendations in This Section

-  ***Recommendation: Strong name verification should never be simulated in an operational environment.***

-  ***Recommendation: Editing CAS policy files to use First Match Code Groups may create invalid or corrupt XML, as these files are also modified by automated tools and parsed by the CLR. Thus, it is recommended that CAS policy be configured using Union Code Groups configured through `mscorcfg.msc`.***

-  ***Recommendation: Editing CAS policy files to create File Code Groups or Net Code Groups may create invalid or corrupt XML, as these files are also modified by automated tools and parsed by the CLR. Thus, it is recommended that these groups be avoided or the XML of the default groups be copied and imported using `mscorcfg.msc`.***

-  ***Recommendation: Although software developers should implement the principle of least privilege through assembly permission requests, security policy should not rely on these requests, but should be implemented through CAS policy settings.***

-  ***Recommendation: Code Groups with the Level Final attribute should have Membership Conditions that are as narrowly defined as possible.***

Deploying .NET Framework CAS Policy Using Group Policy

.NET Framework CAS policy is not centrally maintained, but is stored locally on each computer running the .NET Framework. This allows CAS policy to be tailored to the operational needs of each host computer, but could also make policy enforcement more difficult, since organization-wide changes must be deployed to each computer individually. This section will discuss .NET Framework policy deployment and a few pitfalls to avoid. It will be assumed that the administrator has some understanding of Active Directory and the use of Group Policy to distribute software applications.

Deployment Options

Since the configurable part of .NET Framework CAS policy consists of a set of XML files for the Enterprise, Machine, and User levels of each .NET Framework version, a new policy can be deployed to a host computer simply by copying new policy files over the old ones. Since the CAS policy files for Enterprise and Machine level are protected, a local computer account with elevated privileges is required to change these files. To scale well to even a small network, it is desirable to have an automated process that runs with sufficient privileges and uses a centrally stored installation script and policy files.

An installation script and CAS policy file can be bundled as a Windows Installer package (.msi file). A Windows Installer package is a database containing all the information that Windows Installer (`msiexec.exe`) needs to install or uninstall a policy configuration file. Windows Installer packages can be deployed in several ways:

- Processing the .msi file on the computer where the policy is to be deployed, either from the local disk or from a shared network folder. Double-click the .msi file icon to invoke Windows Installer, or run

```
msiexec.exe /i <filename.msi>
```

from the command line.

- Automated installation using Group Policy software installation.
- Automated installation using Microsoft Systems Management Server.

- Automated installation using logon or startup scripts that reference a shared network location.
- Distribution as an e-mail attachment with subsequent discretionary user processing.
- Publishing on a shared network location with subsequent discretionary user processing.

Of the above methods, only Group Policy and SMS are automated processes that can be guaranteed to perform the installation with elevated privileges and can be configured to apply to groups of computers. In this document, we will discuss automated CAS policy deployment using Windows Installer packages and Group Policy.

Creating a Windows Installer Package for CAS Policy Deployment

The .NET Framework Configuration tool (`mscorcfg.msc`) provides a Wizard for creating Windows Installer packages. The Wizard can create an Installer package that corresponds to exactly one of the three configurable policy levels (Enterprise, Machine, or User) for the version of the .NET Framework administered through the tool. Thus, CAS policy deployment through Installer packages must be performed separately for each policy level and each .NET Framework version.

The Wizard creates the Installer package using the current policy settings of the computer where the Wizard executes. For example, to deploy a custom Machine level policy for version 1.1 of the .NET Framework to a group of computers, you must back up the current Machine level CAS policy file (`security.config`) on a computer, use `mscorcfg.msc` to configure the Machine level policy for that version, create the Installer package with the Wizard as described below, and then restore the computer's original Machine level policy.

When configuring policies for deployment to other machines, the policy back up and restoration process can be done by creating a Windows Installer package with the original policy for the configuring host. When all the Installer packages for policies to be deployed have been created, the host's original CAS policy can be restored by double-clicking the corresponding `.msi` file.



Recommendation: Back up any custom host CAS policy using a Windows Installer package before configuring different policies for deployment.

Since editing the policy temporarily affects the actual policy enforced on the local computer, it is recommended that policy configuration for deployment purposes be performed on a protected host, such as a standalone computer or one with no Internet access.



Recommendation: Configure policy for deployment on a protected host.

To create a Windows Installer package for the current CAS policy level and version, perform the following steps:

- Start the .NET Framework Configuration tool (`mscorcfg.msc`) for the .NET Framework version whose policy you wish to deploy.
- In the console tree on the left, select **Runtime Security Policy**.
- In the **Tasks** list in the **Code Access Security Policy** details pane on the right, click on **Create Deployment Package** to open the Deployment Package Wizard (see Figure 21).

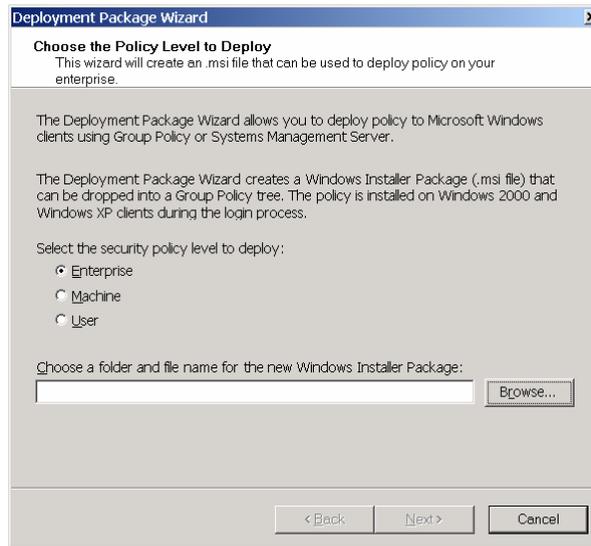


Figure 21. Deployment Package Wizard.

- Select a policy level to deploy by selecting one of the radio buttons: **Enterprise**, **Machine**, or **User**.
- Enter a folder and file name (or browse to a location) for the new Windows Installer package. Note that in order to use Group Policy for deployment, the Installer package must be located in or copied to a folder that is shared across the domain. A possible convention for naming CAS policy Installer packages is the format:

```
cas.{version}.{level}.{yymmdd}.{targeted hosts/users}.msi,
```

for example, `cas.1.0.3705.Machine.20030714.AccountingOU.msi` or `cas.1.1.4322.User.20030714.BackupOperatorsGroup.msi`.

Using this convention, each Installer package can be easily and uniquely identified.

- Select **Next** and then **Finish** and the Wizard will create the package in the specified location.

Deploying CAS Policy Using Group Policy Objects

A Windows Installer package can be deployed to a group of domain computers or domain user accounts defined by an Active Directory container using Group Policy Objects (GPOs). Perform the following steps to deploy CAS policy to the computers or users in an Active Directory container:

- Create the Windows Installer package (.msi file) for the desired policy level and .NET Framework version.
- Copy the Installer package to a shared network folder. Share the folder with appropriate permissions to give access only to authenticated administrators who are responsible for creating and maintaining the Installer packages. Group Policy refreshes will still work even when users cannot read the restricted shared folder, as the refresh process authenticates with a domain machine account rather than a domain user account. Restricting access to the shared folder supports the “need-to-know” doctrine appropriate for security configurations by preventing unauthorized users from determining what CAS policy settings will be deployed to various types of network hosts.



Recommendation: When using a shared network folder as a software distribution point for CAS policy, set the folder permissions to restrict access to administrators or others authorized to maintain .NET Framework CAS policy deployment files.

Either the Active Directory Users and Computers Console (dsa.msc) or the Group Policy Management Console (gpmc.msc) can be used to create and edit GPOs and manage their contents and links to Active Directory containers. Both of these consoles invoke the Group Policy Object Editor Console (gpedit.msc) to add Installer packages to a GPO, but also support automatically linking new GPOs with Active Directory containers.

Creating or Selecting a GPO Using the Active Directory Users and Computers Console

- Log on to the network as a domain administrator, and start the Active Directory Users and Computers snap-in (see Figure 22).
- Find and select the target container node that is or will be linked to the GPO that will contain the CAS policy Installer package.

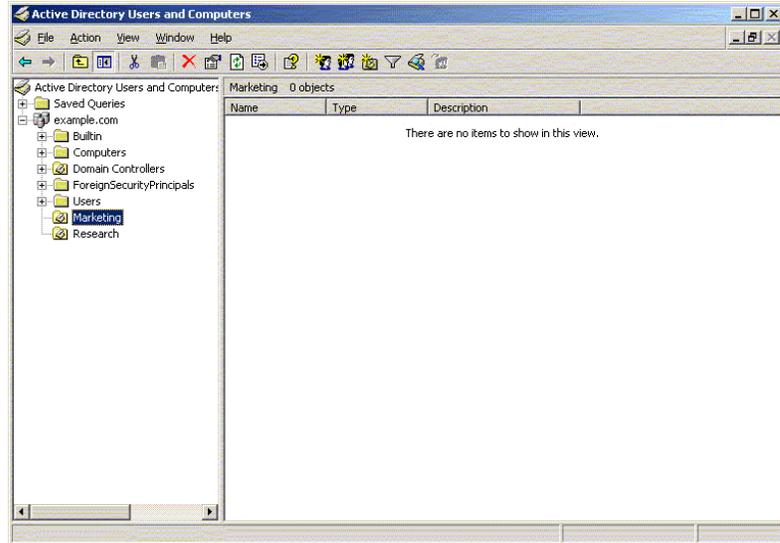


Figure 22. Active Directory Users and Computers Microsoft Management Console Snap-in.

- Right-click the container (or select **Action** from the menu bar) and select **Properties** from the shortcut menu.
- Select the **Group Policy** tab. If the Group Policy Management Console is also installed on this computer, this will simply present a button that will open it, with the focus preset to the selected container. See the section on using Group Policy Management Console below for further steps. Otherwise, a list of GPOs linked to the current container will be displayed (see Figure 23). If the GPO that will contain the Installer package is not currently displayed, click **New** to create a GPO that will automatically be linked to the selected container, or click **Add...** to link an existing GPO to this container.

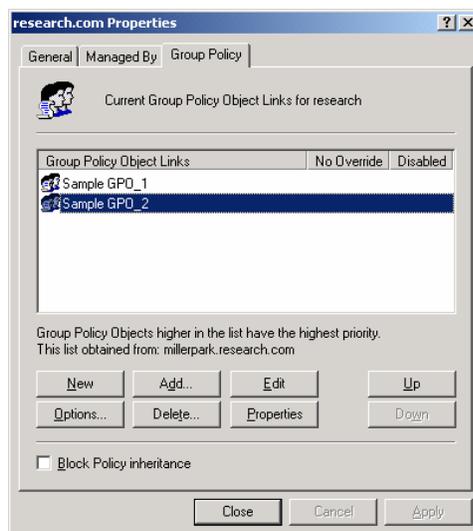


Figure 23. Group Policy Objects Linked to a Container.

- Select the GPO that will contain the Installer package and click **Edit**. This starts the Group Policy Object Editor and lets you edit the properties of this GPO. To add an installer package to this object, follow the steps below in Adding an Installer Package to a GPO Using the Group Policy Object Editor.

Creating or Selecting a GPO Using the Group Policy Management Console

- Log on to the network as a domain administrator, and start the Group Policy Management Console.
- Find and select the target container node that is or will be linked to the GPO that will contain the CAS policy Installer package. The linked GPOs will be shown as subnodes of the container in the console tree pane (see Figure 24).

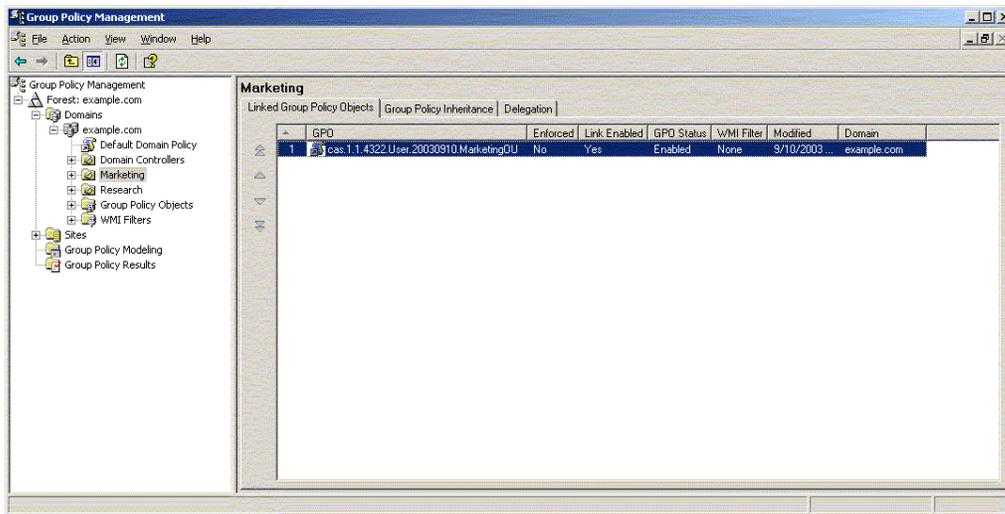


Figure 24. Displaying GPOs Linked to an Organizational Unit with the Group Policy Management Console.

- If the GPO that will contain the CAS policy Installer package is not yet linked to the container, right-click the container (or select **Action** from the menu bar) and select **Create and Link a GPO Here...** or **Link an Existing GPO...** If you are creating a new GPO, you will be prompted to give it a name.
- Right-click the GPO and select **Edit...** This starts the Group Policy Object Editor and lets you edit the properties of this GPO. To add an installer package to this object follow the steps below.

Adding an Installer package to a GPO Using the Group Policy Object Editor

- To deploy Enterprise or Machine level policy, expand the **Software Settings** node under **Computer Configuration**. To deploy User level policy, expand the **Software Settings** node under **User Configuration** (see Figure 25).

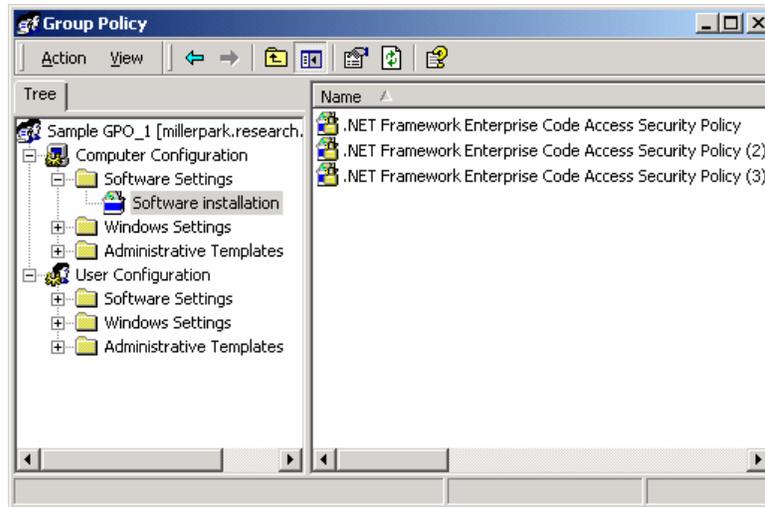


Figure 25. Viewing Software Installation Packages in the Group Policy Object Editor.

- Right-click **Software installation** (or select **Action** from the menu bar) and select **New** and then **Package...**
- Browse to the location of the Windows Installer package, select the file, and click **Open**.

Even if the Windows Installer package resides on the local hard disk, do not use a local path (for example, `c:\sharename\path\filename.msi`). Instead, use a UNC path (such as `\\servername\sharename\path\filename.msi`) to indicate the location of the Installer package or browse through My Network Places to the shared folders on the local host. If a local file system path is used, client computers that try to install the package will look on their own local hard disks in the location that was indicated. Since they will not find the `.msi` file at that location, the installation will fail.

- Select a deployment method from the **Deploy Software** dialog box (see Figure 26).

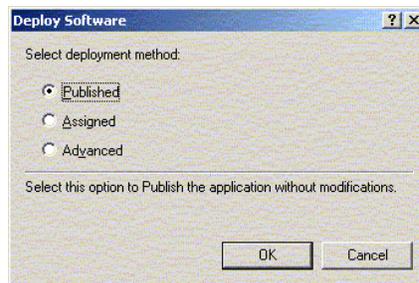


Figure 26. Deploy Software Dialog Box.

The **Published** deployment method, available only through the User Configuration node, permits the Installer package to be made available from the Add New Programs

task of the Add/Remove Programs Control Panel extension. They may then be installed at the discretion of the user. The **Assigned** deployment method directs Group Policy to automatically install the package at the next Group Policy refresh. Both the Published and Assigned selections set default settings that can be configured later, including changing the deployment method between Published and Assigned if this is a User Configuration package. The **Advanced** option (labeled **Advanced published or assigned** in some versions of the snap-in) simply brings up the properties dialog box to allow the administrator to immediately configure the deployment method and settings of the package.

Deployment Modes of Group Policy

The actual deployment of the Installer package will take place during the processing of software installation packages during a Group Policy refresh. Some Group Policy refreshes may not include this phase. The Group Policy deployment modes determine when and how Group Policy will be refreshed and when a Group Policy refresh will include the processing of software installation packages. The point at which a new CAS policy will take effect is dependent on these factors.

Synchronous and Asynchronous Modes

Group Policy refreshes may occur in synchronous or asynchronous mode. In synchronous mode, user interaction with the system is blocked for up to 60 minutes while Group Policy is applied. At computer restart this means that presentation of the logon dialog box will be delayed up to 60 minutes while computer policy is applied. At user logon, this means that, after the user is authenticated, the user's interaction with the desktop will be delayed up to 60 minutes while user policy is applied. In asynchronous mode, user interaction is not blocked. Thus, the user may immediately log on while computer settings are applied, and a logged-on user may have immediate interactive access to the desktop while user settings are being applied.

Software installation only occurs in synchronous Group Policy refreshes. For software under the Computer Configuration node of a GPO, deployment occurs when any computer in the linked Active Directory container restarts with Group Policy refresh in synchronous mode. The deployment of a User Configuration software package will take place when a domain user in the linked Active Directory container logs on, and Group Policy performs a synchronous refresh. Note that logging on to a local machine account will not trigger a Group Policy refresh and thus new software installation packages will not be deployed at that time.

In addition to computer restart and domain user logon ("foreground" policy application), Group Policy refresh can also occur at periodic intervals ("background" policy application). Both types of refresh events are discussed below.

Foreground Policy Application

When settings are applied at restart for Computer Configuration and at logon for User Configuration, this is referred to as foreground policy application. The way policy is applied in a foreground refresh depends on the Windows operating system version. For Windows 2000, all foreground processing is synchronous. Windows XP systems support both synchronous and asynchronous foreground policy processing, including the ability to switch automatically between the two modes (Fast Logon Optimization, described below).

Fast Logon Optimization

Fast Logon Optimization is available in Windows XP Professional and applies to domain and workgroup accounts. This feature allows foreground policy application to use asynchronous (“fast”) mode as a default, but switch to synchronous mode as needed. Since this feature applies both to computer restart and user logon, a better name would be “Fast Restart/Fast Logon Optimization.” When this feature is enabled, user logon is asynchronous unless

- This is the first time the user has logged on to the computer.
- The user is logging on with a roaming profile.
- The user has logon scripts that are configured to require synchronous processing.

Asynchronous restart (“Fast Restart”) may still take place even when the Group Policy refresh at user logon is synchronous. If a pending software installation package is detected during an asynchronous foreground refresh, then the next restart or logon will automatically be synchronous, allowing the software installation to proceed. Thereafter, it will switch back to asynchronous if possible as described above.

The use of Fast Logon Optimization could prevent updated CAS policy settings from being applied during an interactive logon session even though a new policy is available and configured for deployment. An asynchronous restart or logon would simply note the existence of the new CAS policy Installer package, and then initiate the installation at the next restart or logon.

Disabling Fast Logon Optimization

Fast Logon Optimization in Windows XP may be disabled through the Group Policy Object Editor. Under the **Computer Configuration** node, select **Administrative Templates**, **System**, **Logon**, and then enable the setting **Always wait for the network at computer start and logon**.

Background Policy Application

In addition to foreground processing, policy can be applied without restart or logon by the use of background processing. Background processing is always asynchronous and by default a background refresh occurs approximately every 90 minutes. A random time of up to 30 minutes is added to this interval in order to spread out the network traffic associated

with policy refreshes. Because a background policy refresh is asynchronous, software installation (including CAS policy installation via a Windows Installer package) is not performed during a background policy refresh.

Summary

CAS policy deployment via Group Policy software installation occurs only when Group Policy is refreshed synchronously during computer restart or domain user logon. Under Windows XP, the occurrence of synchronous refreshes may be subject to changing conditions that may make CAS policy deployment difficult to predict. To make policy deployment occur as soon as possible and in a predictable way, disable Fast Logon Optimization.

 ***Recommendation: When CAS policy is deployed via Group Policy software installation, disable Fast Logon Optimization.***

Group Policy Processing

Group Policy Processing Precedence

When multiple CAS policy Installer packages are deployed through Group Policy, the Group Policy processing rules may affect which CAS policies will take effect. Since CAS policy installation completely replaces any previous CAS policy, the order in which conflicting policies are installed is important.

Precedence of Active Directory Containers

Group Policy is cumulative in that GPOs applied later may override some settings of GPOs applied earlier, but settings not explicitly overridden are generally retained from the earlier GPOs. GPOs linked to the various levels of the container hierarchy are processed and applied in the following order of containers: Local, Site, Domain, Organizational Unit, Child Organizational Unit. For the order of application of multiple GPOs linked to the same container, see the section Precedence of Linked GPOs below. As a result of the container precedence, child containers generally inherit all policy from parent containers that they don't explicitly override. The Block Inheritance and No Override properties of containers can be used to modify this process, but a full description of Group Policy processing is outside of the scope of this document. For a complete description of Group Policy processing please review [Haney, 2001] and [Sanderson and Rice, 200] or Microsoft's online documentation at msdn.microsoft.com [Microsoft, MSDN].

Group Policy Processing Example 1

The Active Directory Domain container for example.com has a linked GPO that contains a Machine level CAS policy for .NET Framework version 1.1 under its Computer Configuration node. This policy includes a custom Code Group called "DomainAppsCodeGroup". This Domain container contains an Organizational Unit

container called “Research Department” that has a linked GPO containing a Machine level CAS policy for the same version under its Computer Configuration node that contains a custom Code Group called “ResearchToolsCodeGroup”.

When any computer within the “Research Department” container is restarted, the Domain policy is applied first and then the Organizational Unit policy. Since CAS policy installation is simply the replacement of one XML file with another, the policy installed with the Domain container is completely overwritten by the “Research Department” policy. The two policies are not merged and only the Code Group “ResearchToolsCodeGroup” will appear in the final Machine level CAS policy.

If the two policy packages were for different CAS policy levels or different versions of the .NET Framework, then there is no conflict and both custom Code Groups would be present in their respective policies.

Precedence of Linked GPOs

When multiple GPOs are linked to the same Active Directory container, their order of application is determined by their order as configured in the Group Policy snap-in. This order can be modified by the administrator using the Management Consoles. The entries are applied from the bottom of the list to the top (i.e., the policy labeled “1” is applied last), so the higher GPOs have greater precedence. Note that when installing multiple CAS policy packages through Group Policy, the last policy deployed for a given level and version will completely overwrite any policy previously installed (see Figure 27).

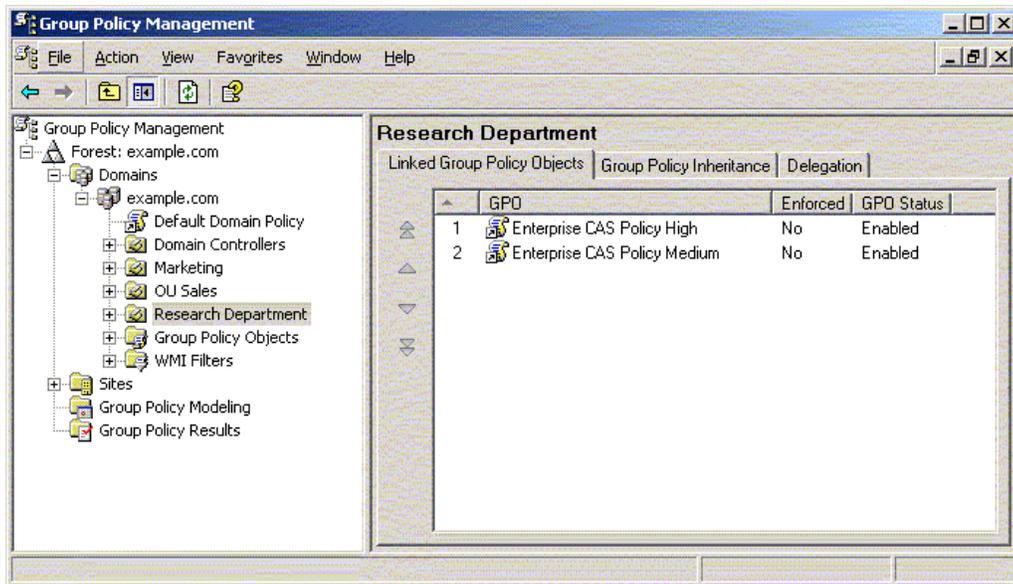


Figure 27. Displaying the Order of Precedence of GPOs Linked to an Organizational Unit with the Group Policy Management Console.

Group Policy Processing Example 2

An Active Directory container is linked to two GPOs which have the order of precedence 1) “Enterprise CAS Policy High” and 2) “Enterprise CAS Policy Medium”. “Enterprise CAS Policy High” will install an Enterprise level policy for .NET Framework version 1.0 that assigns the Nothing+Level Final+Exclusive permissions to a custom Code Group “RestrictedAppsCodeGroup,” and “Enterprise CAS Policy Medium” installs a policy at the same level and .NET Framework version that assigns the Internet+Level Final+Exclusive permissions to the same Code Group.

Since Group Policy processing installs GPOs for a single container from the bottom to the top of the linked GPOs, the .NET Framework Enterprise level policy is installed from “Enterprise CAS Policy Medium” first and then that policy is overwritten by the installation of “Enterprise CAS Policy High.” The final .NET Framework policy thus assigns Nothing+Level Final+Exclusive permissions to the Code Group “RestrictedAppsCodeGroup” at the Enterprise level.

Precedence of Software Installation Packages

When Windows Installer packages are added to a GPO, the default name assigned by the Group Policy snap-in to the software installation entry is the software product name as recorded in the Installer package’s internal database. For Installer packages created with `mscorcfg.msc`, this product name is set to “.NET Framework <policy level> Code Access Security Policy”. If more than one software installation entry in the same GPO has the same internal product name, the names of the additional entries are listed as “.NET Framework <policy level> Code Access Security Policy (2)”, etc. These policy installations will all be installed when the GPO is processed during a Group Policy refresh. The order of installation is the alphabetical order of the software installation entries. As with multiple GPOs linked to one container, the last policy to be installed will overwrite any previously installed policy for the same level and version (see Figure 28).

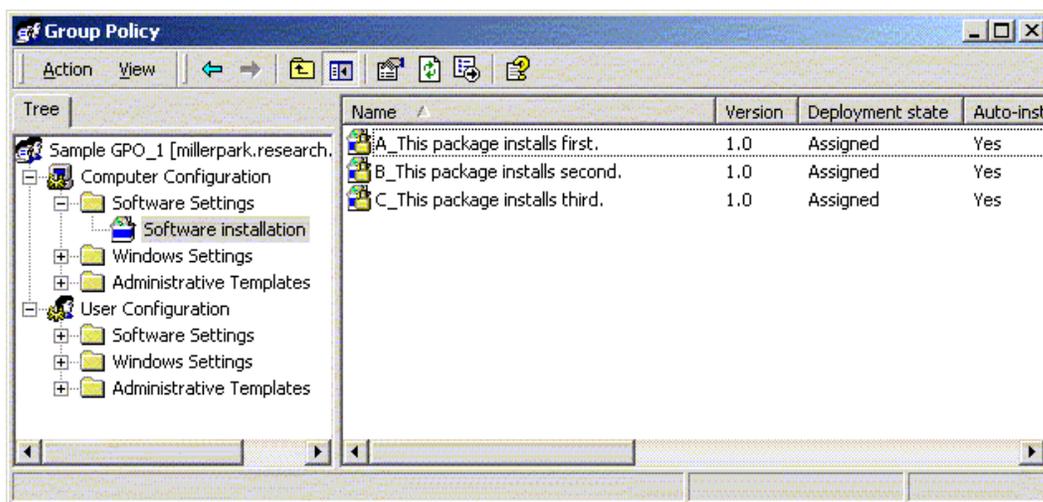


Figure 28. Custom Software Installation Item Names for CAS Policy Installer Packages.

Group Policy Processing Example 3

Windows Installer packages “.NET Framework Enterprise Code Access Security Policy” and “.NET Framework Enterprise Code Access Security Policy (2)” are created using `mscorcfg.msc` for the Enterprise level for .NET Framework versions 1.0 and 1.1, and added to the same GPO. An additional Windows Installer package “.NET Framework Enterprise Code Access Security Policy (3)” is created for the Enterprise level for version 1.1 and added to the same GPO. When the GPO is deployed, the policies will be applied in alphabetical order. If the names of the Installer packages are not changed from their defaults, “.NET Framework Enterprise Code Access Security Policy (3)” will be applied last, and will overwrite the policy previously installed for version 1.1.

The names of the software installation entries may be changed after creation to set a particular CAS policy installation precedence.

Loopback Processing

When a user logs on to a domain account, a Group Policy refresh is triggered that applies the User Configuration settings for all GPOs linked to Active Directory containers that contain the user’s domain account. The Group Policy engine will create an ordered list of all the applicable GPOs based on its precedence rules, and the User Configuration settings will be applied in that order. Active Directory containers that contain only the host computer will not contribute to this process, nor will any Computer Configuration settings for any GPO.

In some situations, it may be desirable for all users logging on to a particular host computer to be subject to a machine-wide policy, rather than their user account policy. This is called Loopback processing and relies on the User Configuration settings for the host computer’s Active Directory containers rather than (or in addition to) the User Configuration settings associated with the user’s domain account. Loopback provides a way to keep CAS policy consistent on machines that are used by a wide population, such as kiosks or classroom workstations, or machines that require special protection, such as servers. Loopback processing can be set to one of two modes, Merge or Replace, which determine how the machine-wide and user policies are combined.

In Merge mode, Loopback applies the User Configuration settings for GPOs linked to Active Directory containers that contain the user’s domain account, and then the User Configuration settings for GPOs linked to Active Directory containers that contain the host computer. Since later settings take precedence, the computer policy will override the user policy wherever there are conflicts, but non-conflicting user policy settings will still take effect. In Replace mode, Loopback applies only the User Configuration settings for GPOs linked to Active Directory containers that contain the host computer. Group Policy settings associated with the user’s domain account are ignored. In both Merge and Replace mode, the computer settings will be applied for all users. Since CAS policy installation consists of simple file replacement, any Installer package associated with the computer will completely determine the final policy for the CAS policy level installed.

To configure Loopback processing, the **User Group Policy loopback processing mode** must be enabled as either **Merge** or **Replace** on the Computer Configuration node of a GPO linked to the targeted host (see Figure 29). Then, when the host restarts or Group Policy is otherwise refreshed, the host will be configured to apply user policy using Loopback in the specified mode.

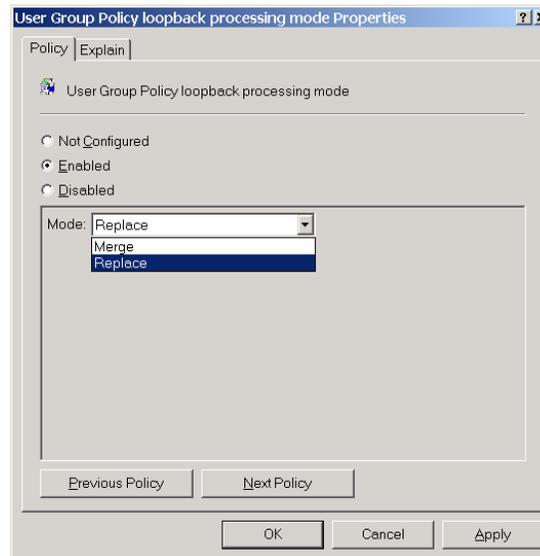


Figure 29. Setting Loopback Processing Mode.

Forcing Policy Deployment

Forcing Policy Deployment Locally

To ensure that an Installer package is deployed to a particular host, a user or administrator may run

```
secedit /refreshpolicy {machine_policy | user_policy} /enforce
```

or

```
gpupdate /force
```

from the command line of that host. `gpupdate.exe` is available on Windows XP and later versions. When running `gpupdate.exe`, the user will be asked for permission to restart the computer. Select **Yes** and the package will be installed on restart.

Forcing Policy Deployment Remotely

After installation and while a Group Policy Object is still linked to a container, the Installer package will remain eligible for subsequent reinstallation on the target computers associated with that container. During each Group Policy software installation, the installed programs list on the targeted computer is consulted and the Installer package is reapplied if it has been

removed. This ensures that as long as a policy object exists, it will be continually reapplied if necessary at each restart or logon.

Software installation relies on the names of the packages to determine if a given package has been removed and needs to be reinstalled. If a Group Policy Object is updated to include a new Installer package with the same name as a prior package, its behavior will be as if it had not changed, that is, it will not be installed on any host that has the prior package still installed. If the Group Policy Object simply points to a generically named package on a Software Distribution Point, replacing the Installer package with a new CAS policy deployment package with the same name will not trigger a deployment. To automatically deploy CAS policy through Group Policy Objects, new CAS policy deployment packages should be given unique names and explicitly linked to a Group Policy Object.

An alternative to using a naming convention to force the deployment of new versions of CAS policy is to configure Group Policy to install/reinstall software installation packages whether or not the named packages are already installed on the targeted machines. This will ensure that the deployed CAS policy overwrites any local modifications at each restart. To configure the software installation behavior of a GPO to reinstall every time, perform the following steps in the Group Policy Object Editor. Under the **Computer Configuration** node, select **Administrative Templates, System, Group Policy**, and then open **Software Installation policy processing** property dialog box. Check **Enable** to configure software installation policy processing and then check **Process even if the Group Policy objects have not changed** (see Figure 30). Note that this will apply to all software installation packages, not just CAS policy.

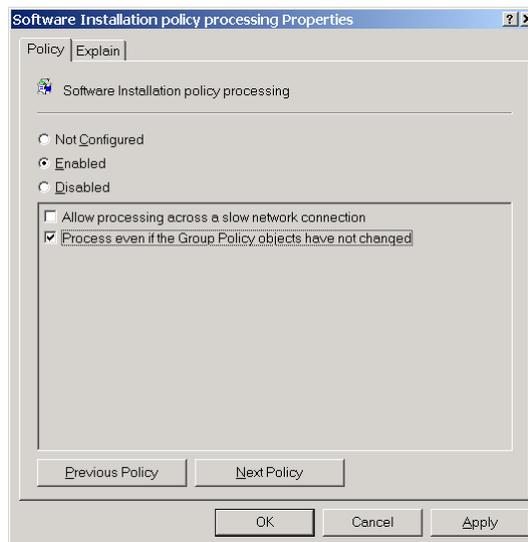


Figure 30. Forcing Software Installation for Unchanged GPOs.

Uninstalling CAS Policy

Windows Installer packages may contain a defined rollback process that can be used to uninstall the package contents. Each Installer package added to a GPO can be configured as an installation or an uninstall/remove task. However, the Installer packages created through `mscorcfg.msc` do not define a rollback process. Therefore, their uninstall task will actually do nothing, although the message dialog may state that the package is being uninstalled.

CAS policy rollback must be performed by reinstalling the previous policy. For this reason, copies of the Windows Installer packages for prior CAS policy deployments (including the default CAS policies) should be archived as restoration points for use when rollback is desired.

 ***Recommendation: Archive Windows Installer packages for all CAS policy deployments, including the default CAS policy, for use as restoration points when rollback to a prior policy state is desired.***

Summary

.NET Framework CAS policy can be deployed by using `mscorcfg.msc` to create Windows Installer packages for a given policy level and .NET Framework version. The Active Directory Users and Computers Console or Group Policy Management Console can be used to associate these packages with GPOs linked to Active Directory containers. These GPOs are deployed when a Group Policy refresh occurs for a given domain computer (normally at restart) or for a domain user (normally at logon). They are applied in the order of highest container GPOs to lowest container GPOs within the Active Directory container hierarchy. Within a given container, the ordered list of linked GPOs is applied bottom to top, and within a GPO, Windows Installer packages are applied in the alphabetical order of the name of their software installation entries.

Rollback of a deployed CAS policy is achieved by reinstalling the previous policy through a separate Windows Installer package.

Recommendations in This Section

 ***Recommendation: Backup any custom host CAS policy using a Windows Installer package before configuring different policies for deployment.***

 ***Recommendation: Configure policy for deployment on a protected host.***

 ***Recommendation: When using a shared network folder as a software distribution point for CAS policy, set the folder permissions to restrict access to administrators or others authorized to maintain .NET Framework CAS policy deployment files.***



Recommendation: When CAS policy is deployed via Group Policy software installation, disable Fast Logon Optimization.



Recommendation: Archive Windows Installer packages for all CAS policy deployments, including the default CAS policy, for use as restoration points when rollback to a prior policy state is desired.

This page has been intentionally left blank.

URL Security Zones and the .NET Framework Zone Membership Condition

Microsoft Windows provides a facility to classify URLs into predefined groups called URL Security Zones. The URL to zone mapping information is contained in the registry and is also available to applications through a programming interface. Its most common use is by Microsoft Internet Explorer to limit the capabilities of code embedded in Web pages. The .NET Framework also uses this data to implement a family of Zone Membership Conditions based on the URL from which an assembly is loaded. Table 5 shows the five built-in Zone Membership Conditions that correspond to the five predefined URL Security Zones used by Internet Explorer.

Zone Membership Condition	URL Security Zone
My Computer	 My Computer
Local Intranet	 Local intranet
Trusted Sites	 Trusted sites
Internet	 Internet
Untrusted Sites	 Restricted sites

Table 5. .NET Framework Zone Membership Conditions and URL Security Zones.

When the CLR loads an assembly through a URL reference, it is associated with a URL Security Zone by checking the mapping data stored in the registry. This zone, along with all other evidence about the assembly, is then used to determine Code Group membership and the set of access permissions the assembly will be granted.

The default Machine level policy contains a Code Group for each of the five built-in Zone Membership Conditions. These default Code Groups and their associated Named Permission Sets are listed in Table 6. Note that the default Named Permission Set for the Internet_Zone Code Group has changed from version 1.0 to version 1.1 of the .NET Framework.

UNCLASSIFIED

Zone Membership Condition	Code Group	Named Permission Set
My Computer	My_Computer_Zone	FullTrust
Local Intranet	LocalIntranet_Zone	LocalIntranet
Trusted Sites	Trusted_Zone	Internet
Internet	Internet_Zone	Nothing (1.0), Internet (1.1)
Untrusted Sites	Restricted_Zone	Nothing

Table 6. Default Code Groups and Named Permission Sets for Zone Membership Conditions.

The default permissions associated with these Named Permission Sets are listed in Table 7.

Named Permission Set	Default Permissions
Nothing	None
Internet	Security: Enable Code Execution File Dialog: Open Isolated Storage File: Usage Allowed = Domain Isolation By User, Disk Quota = 10240 User Interface: Windowing = Safe top-level windows, Clipboard = Own Clipboard Printing: Safe Printing Web Access: If downloaded from a URL, code may connect to the Web site from which it was downloaded
LocalIntranet	Security: Enable Code Execution, Assert any permission that has been granted Environment Variables: USERNAME = Read File Dialog: Unrestricted Isolated Storage File: Usage Allowed = Assembly Isolation By User, Disk Quota = 9,223,372,036,854,775,807 = $2^{63} - 1$ (the largest 64 bit signed integer value) Reflection: Reflection Emit DNS: Access permitted User Interface: Unrestricted Printing: Default Printing Event Log: Instrument access to the local host Web Access: If downloaded through a protocol other than file://, code may connect to the Web site from which it was downloaded. File IO: If loaded from a file:// protocol, such as through a UNC path or a local file system path, code is granted Read and Path Discovery access to the directory from which it was loaded.
FullTrust	All permissions are in the unrestricted state.

Table 7. Default Permissions for Named Permission Sets Associated with Zone Code Groups.

URL Security Zone Settings

Windows maintains security settings for each zone that are used by Internet Explorer to “sandbox” code embedded in or referenced by Web pages. The security settings for each URL Security Zone determine whether Internet Explorer will allow itself to host code managed by the .NET Framework. Each zone can be configured independently to allow or disallow the execution of managed controls, i.e., assemblies referenced by <object> HTML tags in a Web page. In addition, managed controls that are signed with an Authenticode digital certificate can be configured differently than unsigned managed controls. Table 8 shows the URL Security Zone settings that impact the execution of managed controls. These settings are configurable through Internet Explorer.

URL Security Zone Setting	Impact on Managed Code Execution
.NET Framework-reliant components	
Run components not signed with Authenticode	This setting must be enabled to permit the execution of managed code that is not digitally signed with a software publisher’s certificate.
Run components signed with Authenticode	This setting must be enabled to permit the execution of managed code that is digitally signed with a software publisher’s certificate.
ActiveX controls and plug-ins	
Run ActiveX controls and plug-ins	This ActiveX setting also applies to managed controls and must be enabled to permit their execution.
Script ActiveX controls marked safe for scripting	This ActiveX setting also applies to managed controls and must be enabled to permit scripts that are part of a Web page to invoke code supplied by managed controls.

Table 8. URL Security Zone Settings That Impact Managed Code Execution.

Note that an assembly loaded directly through a URL to a managed executable will not be governed by these settings. Although its execution must be explicitly permitted through an open/save dialog box, once allowed to run, it will simply execute under the restrictions imposed by the current CAS policy configuration.

Once Internet Explorer begins hosting managed code in accordance with the above settings, access to resources is based on the access control mechanism in the .NET Framework rather than on the browser security configuration. The Named Permission Sets associated with the Code Groups to which an assembly belongs will determine the level of access granted. Although the above settings may determine whether managed code gets to run at all, once it is running, the mapping of URLs to zones is the only aspect of Windows URL Security Zones that is used to determine .NET Framework resource permissions.

The zone security settings that allow or disallow the execution of signed or unsigned managed code components add a coarsely-grained authorization layer before the .NET Framework CAS policy is applied. CAS policy provides a much finer degree of control in that code is authorized to access specific resources. Moreover, CAS policy does not have a built-in way to grant access to resources based on the mere fact that code was signed by some Authenticode digital certificate.

An Authenticode digital certificate provides assurance that the code has been distributed without modification since it was signed. By itself, it does not provide any basis for trusting the origin of the code, nor does a certificate by itself prove the identity of its claimed origin. There must be a prior association between the public key used in the digital certificate and a trusted source. This association must be achieved by the trusted distribution of the public keys of trusted parties. In addition, digital certificates are subject to revocation in infrequent cases. Thus, code should not be granted additional access to resources based solely on the fact that it is signed with an Authenticode digital certificate. In the .NET Framework, access is granted to code based on the presence of specific Authenticode digital certificates whose public keys have been previously obtained through an authenticated channel.

Internet Explorer Enhanced Security Configuration

Windows Server 2003 comes with a feature called Internet Explorer Enhanced Security Configuration (ESC). ESC creates a more restrictive environment for browsing Web pages containing embedded or referenced executable code, and maintains an alternate zone mapping. In addition, when ESC is enabled the Windows Update site `http://*.windowsupdate.com` and Microsoft's Online Crash Analysis site `http(s)://oca.microsoft.com` are added to the Trusted sites zone, and `http(s)://localhost` and `hpc://system` are added to the Local intranet zone.

A registry setting enables or disables ESC for the Administrators group. Another registry setting enables or disables ESC for all other groups. To install, enable, or configure ESC, see the Windows Server 2003 documentation at `msdn.microsoft.com`.

URL to Zone Mappings

Separate URL to Zone mappings are stored in the Current User (HKCU) and Local Machine (HKLM) hives of the registry, as shown in Table 9. The Current User mappings are saved to a roaming user's network profile.

Registry Key	Description
HKLM/Software/Policies/Microsoft/Windows/CurrentVersion/InternetSettings/ Security_HKLM_Only	If this key exists and has the value 1, then only the Local Machine mappings will be used to map a URL to a zone.
HKLM/Software/Microsoft/Windows/CurrentVersion/Internet Settings/	

Registry Key	Description
ZoneMap/Domains ZoneMap/EscDomains	Maps network domains and UNC paths to security zones. If ESC is installed and enabled for the current user's group, EscDomains is used instead of Domains.
ZoneMap/ProtocolDefaults	Maps protocols to default security zones. These settings are used if a mapping is not found for the domain or IP range.
ZoneMap/Ranges	Maps IP address ranges to security zones.
Zones/<zone ID>	Defines the security settings and other properties for each security zone. The zone IDs are: 0 My Computer 1 Local intranet 2 Trusted sites 3 Internet 4 Restricted sites These settings are easily configured through Internet Explorer.
HKCU/Software/Microsoft/Windows/CurrentVersion/Internet Settings/	
ZoneMap/Domains ZoneMap/EscDomains	Maps network domains and UNC paths to security zones. If ESC is installed and enabled for the current user's group, EscDomains is used instead of Domains.
ZoneMap/ProtocolDefaults	Maps protocols to default security zones. These settings are used if a mapping is not found for the domain or IP range.
ZoneMap/Ranges	Maps IP address ranges to security zones.
Zones/<zone ID>	Defines the security settings and other properties for each security zone. These settings are easily configured through Internet Explorer.

Table 9. URL Security Zone Registry Keys.

When an assembly is obtained from a URL, the zone mappings in the registry are checked to determine Zone membership. The Local Machine mappings are checked first for the first matching entry (or default protocol mapping). Note that domain names are not resolved into IP addresses, so the same Web site could be in one zone if specified by domain name and in another zone if specified by IP address. Port number suffixes and resource path components beyond the domain/IP address are ignored. If user settings are not disabled, the Current User mappings are then checked and may override the Local Machine settings if there is a conflict. If ESC is installed and enabled for the current user's group, then the EscDomains mappings are used instead of the Domains mappings for both the Local Machine and Current User settings. ESC does not use a separate set of IP address ranges.

The user has the ability to modify the URL security zones under the Current User (HKCU) hive in the registry, either through Internet Explorer or by directly editing the registry. Thus, the user may add any URL to the Trusted sites security zone, or even add arbitrary URLs to the My Computer zone. This means that .NET Framework CAS policy that grants

permissions based on Zones cannot enforce a machine-wide mapping policy unless user mappings are disabled using the Security_HKLM_Only value. Where Zone Membership Conditions are present and user mappings are not disabled, users may allow arbitrary code to satisfy any Zone Membership Condition.

 ***Recommendation: Do not grant or restrict access to resources based on a Zone Membership Condition in support of an organizational policy unless user mappings are disabled.***

 ***Recommendation: Only use Zone Membership Conditions as part of a multi-factor code authorization check that relies on at least one additional type of evidence before granting access to resources.***

One way to implement this recommendation is by removing the Security: Enable Code Execution flag from any Named Permission Set associated with a Code Group with a Zone Membership Condition. This will allow Zones to define certain broad types of access, but the right to actually execute must be based on a more specific form of evidence.

Summary

The .NET Framework Zone Membership Condition relies on the configuration of the URL Security Zones in the Windows operating system. Thus, the set of permissions granted to code based on an administratively-configured CAS policy that includes Zone Membership Conditions is partly determined by an external operating system component that is potentially configurable by any user. Unlike Windows file permissions or user account restrictions, this external component (URL to Zone mappings) does not create an additional layer of security, but interacts with and determines the meaning of CAS policy. This presents a challenge to the enforcement of organizational policy decisions. This challenge can be addressed by disabling user settings and designing CAS policy to grant access to resources based on stronger forms of evidence than Zone membership.

Recommendations in This Section

 ***Recommendation: Do not grant or restrict access to resources based on a Zone Membership Condition in support of an organizational policy unless user mappings are disabled.***

 ***Recommendation: Only use Zone Membership Conditions as part of a multi-factor code authorization check that relies on at least one additional type of evidence before granting access to resources.***

Cryptographic Localization in the .NET Framework

The CLR libraries include many classes that provided cryptographic services such as hash functions, asymmetric and symmetric encryption algorithms, and digital signatures. Table 10 shows the types of cryptographic services available in the .NET Framework as of version 1.1. Note that this document does not address the strength or correctness of the cryptographic algorithm implementations provided by the .NET Framework libraries. In addition to the cryptographic services provided in the .NET Framework, cryptographic services in the Windows operating system may be configured and customized through third-party Cryptographic Service Providers (CSPs). CSPs are not administered through the .NET Framework and their configuration is not covered in this document.

Cryptographic Services by Type	
Hash	
	MD5
	SHA-1
	SHA-256
	SHA-512
Keyed Hash	
	HMAC
	CBC MAC
Symmetric Encryption	
	DES
	Triple DES
	RC2
	Rijndael (the candidate algorithm selected by NIST for the AES)
Asymmetric Encryption	
	RSA
Pseudo-Random Number Generation and Related Services	
	Pseudo-Random Number Generator (PRNG)
	Password-Based Key Derivation Function
	PKCS #1 MGF1 (Mask Generation Function)
Digital Signature	
	DSA signature encoded in PKCS #1 version 1.5 digital signature format
	RSA signature encoded in PKCS #1 version 1.5 digital signature format
XML Digital Signature Transforms	
	Canonicalization, with or without comments

Cryptographic Services by Type
Base64 decoding
XSLT transform
XPath filtering
Enveloped signature transform
Key Exchange
RSA encryption of Optimal Asymmetric Encryption Padding (OAEP) encoded key
RSA encryption of PKCS #1 version 1.5 encoded key

Table 10. Types of cryptographic services in the .NET Framework version 1.1.

The .NET Framework provides default algorithm specifications and implementations for some families of cryptographic functions and also provides the means to selectively override these defaults. This is done by allowing managed code to request cryptographic functions through cryptographic service names. These names are resolved by the CLR into specific execution paths based on administrator-configurable cryptographic settings that map names to software libraries. Many predefined names exist for common cryptographic services. In addition, new names and mappings may be defined by the administrator to describe policy-driven cryptographic standards.

The default cryptographic algorithms must be chosen carefully to provide appropriate protection for sensitive information. Table 11 shows appropriate defaults for each class of algorithm from among the built-in cryptographic services available in version 1.1 of the .NET Framework. Note that operational and mission needs for an information system may require a higher degree of protection than that afforded by the built-in services of the .NET Framework; however, custom cryptography should only be obtained from highly trusted and appropriately accredited expert sources.

Algorithm Type	Default Algorithm
Hash	SHA-256
Keyed Hash	HMAC with SHA-1. HMAC with SHA-256 is recommended where possible, but this is not available with the built-in services in version 1.1 of the .NET Framework.
Symmetric Encryption	Rijndael with 256-bit cryptovvariable (this is the default). Triple DES (168-bit cryptovvariable) may be used where interoperability with legacy systems is critical.
Asymmetric Encryption	RSA with 2048-bit key size
Pseudo-Random Number Generator	Microsoft Strong Cryptographic Provider. This API is wrapped by the cryptographic service System.Security.Cryptography.RNGCryptoServiceProvider.
Digital Signature	RSA PKCS #1 version 1.5 digital signature with 2048-bit key size
XML Digital Signature Transforms	Any
Key Exchange	RSA PKCS #1 version 1.5 key exchange with 2048-bit key size

Table 11. Recommended default cryptographic algorithms.

Table 12 shows the named cryptographic services in version 1.1 of the .NET Framework and their default mappings. When cryptographic services are invoked by software using one of the quoted names, the behavior is as described in the following paragraph. For example, invoking “MD5” or “System.Security.Cryptography.MD5” results in the same behavior; namely, to invoke the currently configured default implementation of the MD5 hash algorithm.

Many names correspond to specific internal software identifiers of cryptographic services. For example, System.Security.Cryptography.MD5CryptoServiceProvider is an internal software identifier for a specific implementation of MD5 in the .NET Framework library mscorlib.dll. This implementation is invoked through the name “System.Security.Cryptography.MD5CryptoServiceProvider.” In addition to these implementation-specific names, some generic names are defined for use where a type of algorithm is needed, but not any particular implementation. As noted above, the names “MD5” and “System.Security.Cryptography.MD5” are examples of this type of name, as is “HashAlgorithm.” When managed code requests the generic “HashAlgorithm” service, the default behavior is to invoke System.Security.Cryptography.SHA1CryptoServiceProvider, the SHA-1 implementation in mscorlib.dll, but another hash function implementation could be substituted using the cryptographic configuration settings.

Mapped Behavior of Named Cryptographic Services	
Hash	
“System.Security.Cryptography.HashAlgorithm”	Invokes the currently configured default hash algorithm. The default is System.Security.Cryptography.SHA1CryptoServiceProvider in mscorlib.dll. See the name “System.Security.Cryptography.SHA1CryptoServiceProvider” below for details.
“MD5”, “System.Security.Cryptography.MD5”	Invokes the currently configured default MD5 implementation. The default implementation is System.Security.Cryptography.MD5CryptoServiceProvider in mscorlib.dll. See the name “System.Security.Cryptography.MD5CryptoServiceProvider” below for details.
“System.Security.Cryptography.MD5CryptoServiceProvider”	MD5, 128 bit message digest. This invokes managed code in mscorlib.dll (System.Security.Cryptography.MD5CryptoServiceProvider) that wraps the Windows operating system default CSP implementation of MD5.
“SHA”	Invokes the currently configured default SHA implementation. The default is System.Security.Cryptography.SHA1CryptoServiceProvider in mscorlib.dll. See the name “System.Security.Cryptography.SHA1CryptoServiceProvider” below for details.

UNCLASSIFIED

Mapped Behavior of Named Cryptographic Services	
<p>“SHA1”, “System.Security.Cryptography.SHA1”, “http://www.w3.org/2000/09/xmlldsig#sha1”</p>	<p>Invokes the currently configured default SHA-1 implementation. The default is System.Security.Cryptography.SHA1CryptoServiceProvider in mscorlib.dll. See the name “System.Security.Cryptography.SHA1CryptoServiceProvider” below for details.</p>
<p>“System.Security.Cryptography.SHA1CryptoServiceProvider”</p>	<p>SHA-1, 160 bit message digest. This invokes managed code in mscorlib.dll (System.Security.Cryptography.SHA1CryptoServiceProvider) that wraps the Windows operating system default CSP implementation of SHA-1.</p>
<p>“System.Security.Cryptography.SHA1Managed”</p>	<p>SHA-1, 160 bit message digest, .NET Framework library implementation in mscorlib.dll.</p>
<p>“SHA256”, “SHA-256”, “System.Security.Cryptography.SHA256”</p>	<p>Invokes the currently configured default SHA-256 implementation. The default is System.Security.Cryptography.SHA256Managed in mscorlib.dll. See the name “System.Security.Cryptography.SHA256Managed” below for details.</p>
<p>“System.Security.Cryptography.SHA256Managed”</p>	<p>SHA-256, 256 bit message digest, .NET Framework library implementation in mscorlib.dll.</p>
<p>“SHA384”, “SHA-384”, “System.Security.Cryptography.SHA384”</p>	<p>Invokes the currently configured default SHA-384 implementation. The default is System.Security.Cryptography.SHA384Managed in mscorlib.dll. See the name “System.Security.Cryptography.SHA384Managed” below for details.</p>
<p>“System.Security.Cryptography.SHA384Managed”</p>	<p>SHA-384, 384 bit message digest, .NET Framework library implementation in mscorlib.dll.</p>
<p>“SHA512”, “SHA-512”, “System.Security.Cryptography.SHA512”</p>	<p>Invokes the currently configured default SHA-512 implementation. The default is System.Security.Cryptography.SHA512Managed in mscorlib.dll. See the name “System.Security.Cryptography.SHA512Managed” below for details.</p>
<p>“System.Security.Cryptography.SHA512Managed”</p>	<p>SHA-512, 512 bit message digest, .NET Framework library implementation in mscorlib.dll.</p>

UNCLASSIFIED

Mapped Behavior of Named Cryptographic Services	
Keyed Hash	
"System.Security.Cryptography.KeyedHashAlgorithm"	
Invokes the currently configured default keyed hash algorithm. The default is System.Security.Cryptography.HMACSHA1 in mscorlib.dll. See the name "System.Security.Cryptography.HMACSHA1" below for details.	
"HMACSHA1", "System.Security.Cryptography.HMACSHA1"	
HMAC SHA-1, 160 bit message digest, .NET Framework library implementation in mscorlib.dll. Any size cryptovvariable may be provided or the "System.Security.Cryptography.RandomNumberGenerator" named service will be invoked to generate a 64 byte key.	
"System.Security.Cryptography.MACTripleDES"	
MAC Triple DES, 64 bit message digest, .NET Framework library implementation in mscorlib.dll. A 64, 128, or 192 bit cryptovvariable may be provided or the "System.Security.Cryptography.RandomNumberGenerator" named service will be invoked to generate a 192 bit key. Any named implementation of Triple DES may be specified, but by default the "System.Security.Cryptography.TripleDES" named service will be invoked.	
Symmetric Encryption	
"System.Security.Cryptography.SymmetricAlgorithm"	
Invokes the currently configured default symmetric encryption algorithm. The default is System.Security.Cryptography.RijndaelManaged in mscorlib.dll. See the name "System.Security.Cryptography.RijndaelManaged" below for details.	
"DES", "System.Security.Cryptography.DES"	
Invokes the currently configured default DES implementation. The default is System.Security.Cryptography.DESCryptoServiceProvider in mscorlib.dll. See the name "System.Security.Cryptography.DESCryptoServiceProvider" below for details.	
"System.Security.Cryptography.DESCryptoServiceProvider"	
DES, 56 bit cryptovvariable, 64 bit block size. This invokes managed code in mscorlib.dll (System.Security.Cryptography.DESCryptoServiceProvider) that wraps the Windows operating system default CSP implementation of DES. The default mode is CBC, the default padding is zeros.	
"3DES", "TripleDES", "Triple DES", "System.Security.Cryptography.TripleDES"	
Invokes the currently configured default Triple DES implementation. The default is System.Security.Cryptography.TripleDESCryptoServiceProvider in mscorlib.dll. See the name "System.Security.Cryptography.TripleDESCryptoServiceProvider" below for details.	
"System.Security.Cryptography.TripleDESCryptoServiceProvider"	

UNCLASSIFIED

Mapped Behavior of Named Cryptographic Services	
	Triple DES, 112 or 168 bit cryptovvariable, 64 bit block size. This invokes managed code in mscorlib.dll (System.Security.Cryptography.TripleDESCryptoServiceProvider) that wraps the Windows operating system default CSP implementation of Triple DES. The default mode is CBC, the default padding is zeros.
“RC2”, “System.Security.Cryptography.RC2”	
	Invokes the currently configured default RC2 implementation. The default is System.Security.Cryptography.RC2CryptoServiceProvider in mscorlib.dll. See the name “System.Security.Cryptography.RC2CryptoServiceProvider” below for details.
“System.Security.Cryptography.RC2CryptoServiceProvider”	
	RC2, 40 bit cryptovvariable, 64 bit block size. This invokes managed code in mscorlib.dll (System.Security.Cryptography.RC2CryptoServiceProvider) that wraps the Windows operating system default CSP implementation of RC2. The default mode is CBC, the default padding is zeros.
“Rijndael”, “System.Security.Cryptography.Rijndael”	
	Invokes the currently configured default Rijndael implementation. The default is System.Security.Cryptography.RijndaelManaged in mscorlib.dll. See the name “System.Security.Cryptography.RijndaelManaged” below for details.
“System.Security.Cryptography.RijndaelManaged”	
	Rijndael, 128, 192, or 256 bit cryptovvariable (default is 256), 128, 192, or 256 bit block size (default is 128), .NET Framework library implementation in mscorlib.dll. The default mode is CBC, the default padding is zeros.
Asymmetric Encryption	
“System.Security.Cryptography.AsymmetricAlgorithm”	
	Invokes the currently configured default asymmetric encryption algorithm. The default is System.Security.Cryptography.RSACryptoServiceProvider in mscorlib.dll. See the name “System.Security.Cryptography.RSACryptoServiceProvider” below for details.
“RSA”, “System.Security.Cryptography.RSA”	
	Invokes the currently configured default RSA implementation. The default is System.Security.Cryptography.RSACryptoServiceProvider in mscorlib.dll. See the name “System.Security.Cryptography.RSACryptoServiceProvider” below for details.
“System.Security.Cryptography.RSACryptoServiceProvider”	
	RSA, default key size is 1024 bits. This invokes managed code in mscorlib.dll (System.Security.Cryptography.RSACryptoServiceProvider) that wraps the Windows operating system default “PROV_RSA_FULL” type CSP implementation of RSA.

UNCLASSIFIED

Mapped Behavior of Named Cryptographic Services	
Pseudo-Random Number Generation and Related Services	
<p>“RandomNumberGenerator”, “System.Security.Cryptography.RandomNumberGenerator”</p>	<p>Invokes the currently configured default PRNG. The default is System.Security.Cryptography.RNGCryptoServiceProvider in mscorlib.dll. See the name “System.Security.Cryptography.RNGCryptoServiceProvider” below for details.</p>
<p>“System.Security.Cryptography.RNGCryptoServiceProvider”</p>	<p>This service invokes managed code in mscorlib.dll (System.Security.Cryptography.RNGCryptoServiceProvider) that wraps calls to any specified installed CSP. The default is the “PROV_RSA_FULL” type implementation for the Microsoft Strong Cryptographic Provider.</p>
<p>“System.Security.Cryptography.PasswordDeriveBytes”</p>	<p>This service invokes managed code in mscorlib.dll (System.Security.Cryptography.PasswordDeriveBytes) that provides password-based key derivation functions using either an installed CSP or iterating a hash algorithm. The default CSP used is the default “RSA_PROV_FULL” provider. The default iterative hash method is to use 100 iterations of the named service “System.Security.Cryptography.SHA1.” See this name above for details.</p>
<p>“System.Security.Cryptography.PKCS1MaskGenerationMethod”</p>	<p>PKCS #1 MGF1 (Mask Generation Function), .NET Framework library implementation in mscorlib.dll. By default, invokes the named service “SHA1” as the hash component. See this name above for details.</p>
Digital Signature	
<p>“DSA”, “System.Security.Cryptography.DSA”</p>	<p>Invokes the currently configured default DSA implementation. The default is System.Security.Cryptography.DSACryptoServiceProvider in mscorlib.dll. See the name “System.Security.Cryptography.DSACryptoServiceProvider” below for details.</p>
<p>“System.Security.Cryptography.DSACryptoServiceProvider”</p>	<p>DSA, default key size is 1024 bits. This invokes managed code in mscorlib.dll (System.Security.Cryptography.DSACryptoServiceProvider) that wraps the Windows operating system default “PROV_DSS_DH” type CSP implementation of DSA.</p>
<p>“System.Security.Cryptography.DSASignatureFormatter”, “System.Security.Cryptography.DSASignatureDeformatter”</p>	<p>DSA PKCS #1 version 1.5 digital signature. By default, invokes the named service “SHA1” as the hash algorithm. See this name above for details. Must be invoked with a specified implementation of DSA, as no default is defined.</p>
<p>“RSA”, “System.Security.Cryptography.RSA”</p>	

Mapped Behavior of Named Cryptographic Services	
	Invokes the currently configured default RSA implementation. The default is System.Security.Cryptography.RSACryptoServiceProvider in mscorlib.dll. See the name “System.Security.Cryptography.RSACryptoServiceProvider” below for details.
	“System.Security.Cryptography.RSACryptoServiceProvider”
	RSA, default key size is 1024 bits. This invokes managed code in mscorlib.dll (System.Security.Cryptography.RSACryptoServiceProvider) that wraps the Windows operating system default “PROV_RSA_FULL” type CSP implementation of RSA.
	“System.Security.Cryptography.RSAPKCS1SignatureFormatter”, “System.Security.Cryptography.RSAPKCS1SignatureDeformatter”
	RSA PKCS #1 version 1.5 digital signature. Must be invoked with a named hash service and any implementation of RSA specified, as no defaults are defined.
	“http://www.w3.org/2000/09/xmldsig#dsa-sha1”
	Invokes the currently configured XML digital signature algorithm with DSA and SHA-1. The default implementation, from the .NET Framework library system.security.dll, invokes the named services “System.Security.Cryptography.SHA1CryptoServiceProvider”, “System.Security.Cryptography.DSACryptoServiceProvider”, “System.Security.Cryptography.DSASignatureFormatter”, and “System.Security.Cryptography.DSASignatureDeformatter” for hashing the data, signing the hash, and encoding/decoding the signature, respectively. See these names above for details.
	“http://www.w3.org/2000/09/xmldsig#rsa-sha1”
	Invokes the currently configured XML digital signature algorithm with RSA and SHA-1. The default implementation, from the .NET Framework library system.security.dll, invokes the named services “System.Security.Cryptography.SHA1CryptoServiceProvider”, “System.Security.Cryptography.RSACryptoServiceProvider”, “System.Security.Cryptography.RSAPKCS1SignatureFormatter”, and “System.Security.Cryptography.RSAPKCS1SignatureDeformatter” for hashing the data, signing the hash, and encoding/decoding the signature, respectively. See these names above for details.
XML Digital Signature Transforms	
	“http://www.w3.org/TR/2001/REC-xml-c14n-20010315”
	Invokes the currently configured XML canonicalization algorithm that ignores comments. The default is the .NET Framework library implementation System.Security.Cryptography.Xml.XmlDsigC14NTransform in system.security.dll.
	“http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments”

Mapped Behavior of Named Cryptographic Services	
	<p>Invokes the currently configured XML canonicalization algorithm that includes comments. The default is the .NET Framework library implementation System.Security.Cryptography.Xml.XmlDsigC14NWithCommentsTransform in system.security.dll.</p>
	<p>“http://www.w3.org/2000/09/xmldsig#base64”</p>
	<p>Invokes the currently configured Base64 decoding algorithm. The default is the .NET Framework library implementation System.Security.Cryptography.Xml.XmlDsigBase64Transform in system.security.dll.</p>
	<p>“http://www.w3.org/TR/1999/REC-xslt-19991116”</p>
	<p>Invokes the currently configured XSLT transformer. The default is the .NET Framework library implementation System.Security.Cryptography.Xml.XmlDsigXsltTransform in system.security.dll.</p>
	<p>“http://www.w3.org/TR/1999/REC-xpath-19991116”</p>
	<p>Invokes the currently configured XPath filter processor. The default is the .NET Framework library implementation System.Security.Cryptography.Xml.XmlDsigXPathTransform in system.security.dll.</p>
	<p>“http://www.w3.org/2000/09/xmldsig#enveloped-signature”</p>
	<p>Invokes the currently configured Enveloped Signature transform (this transform specifies all the XML elements in the containing document except the <Signature> element that contains the transform specification itself). The default is the .NET Framework library implementation System.Security.Cryptography.Xml.XmlDsigEnvelopedSignatureTransform in system.security.dll.</p>
Key Exchange	
	<p>“System.Security.Cryptography.RSAOAEPKeyExchangeFormatter”, “System.Security.Cryptography.RSAOAEPKeyExchangeDeformatter”</p>
	<p>Optimal Asymmetric Encryption Padding. This feature is only available with Windows XP or later. Must be invoked with a specific implementation of RSA, as no default is defined.</p>
	<p>“System.Security.Cryptography.RSAPKCS1KeyExchangeFormatter”, “System.Security.Cryptography.RSAPKCS1KeyExchangeDeformatter”</p>
	<p>RSA PKCS #1 version 1.5 key exchange. Must be invoked with a specific implementation of RSA, as no default is defined.</p>

Table 12. Named cryptographic services and their default behavior in the .NET Framework version 1.1.

Modifications to the cryptographic settings should be based on carefully considered security policy decisions. Vulnerabilities may be introduced by changing defaults to weaker algorithms, or by attempting to implement custom cryptographic algorithm implementations that have not been thoroughly evaluated.

Custom cryptographic configuration settings are stored in XML on each host in the `machine.config` file in each .NET Framework version's `config` folder. These settings simply override on a setting-by-setting basis a default cryptographic configuration that is programmed into the cryptographic libraries. Each version of the .NET Framework must be configured separately by modifying the corresponding `machine.config` file. This file contains configuration settings for a variety of .NET Framework features including assembly binding (version redirection), assembly remoting (setting up communication channels that cross application domain, process, or computer boundaries), and ASP.NET. Some of these features are configurable through the .NET Framework configuration tool (`mscorcfg.msc`), but cryptography must be modified by hand. Because parts of this XML file must be modified by the user and parts are modified by an automated tool, changes should be made carefully to avoid leaving the file in an unusable state. In addition, the variety of configuration information stored in this file makes tailored deployment of host or domain-specific cryptography settings a challenge. The simplest deployment method, file replacement, may cause configuration data for other features to revert to an outdated or inappropriate state.

Cryptography settings are stored under the `<cryptographicSettings>` element under `<mscorlib>` or `<mscorlib version="{mscorlib assembly version}">`. The XML fragment in Figure 31 illustrates the relevant portion of the configuration file structure. Note that `<configuration>` is the root element of `machine.config` and the `<configSections>` element defines what constitute valid subsections (child elements) of the root. The `<mscorlib>...</mscorlib>` section will not be parsed unless there is a corresponding `<section>` entry under `<configsections>` as shown.

```
<configuration>
  <configSections>
    <section name="mscorlib"
      type="System.Configuration.IgnoreSectionHandler, System,
      Version=1.0.5000.0, Culture=neutral,
      PublicKeyToken=b77a5c561934e089" allowLocation="false" />
    ...
  </configSections>
  <mscorlib version="1.0.5000.0">
    <cryptographicSettings>
      ...
    </cryptographicSettings>
  </mscorlib>
  <mscorlib version="1.0.3000.0">
    <cryptographicSettings>
      ...
    </cryptographicSettings>
  </mscorlib>
```

```

<mscorlib>
  <cryptographySettings>
    ...
  </cryptographySettings>
</mscorlib>
</configuration>

```

Figure 31. XML Structure of the Cryptography Settings in `machine.config`.

Note that the figure shows multiple `<mscorlib>` elements. These are mutually exclusive – only one will be used. When the configuration file is parsed, only the settings under the first `<mscorlib>` element with a `version` attribute that matches the assembly version of `mscorlib.dll` that is currently loaded in the execution environment are used. Note that this is the version of the assembly itself, not the .NET Framework version. If such an element is not present, then the last occurrence of `<mscorlib>` with no `version` attribute is used. If neither of these cases hold, or there is no `machine.config` file, the built-in default configuration is used unmodified.

The cryptographic configuration consists of a dictionary of short class names for cryptographic classes (`cryptoClass` elements), a list of cryptographic service names (`nameEntry` elements), and a list of `oidEntry` elements. These are structured under various headings as shown in Figure 32:

```

<cryptographySettings>
  <cryptoNameMapping>
    <cryptoClasses>
      <cryptoClass .../>
      ...
      <cryptoClass .../>
    </cryptoClasses>
    <nameEntry .../>
    ...
    <nameEntry .../>
  </cryptoNameMapping>
  <oidMap>
    <oidEntry .../>
    ...
    <oidEntry .../>
  </oidMap>
</cryptographySettings>

```

Figure 32. XML Structure of the `<cryptographySettings>` Element.

The <cryptoClass> element

These elements have the form

```
<cryptoClass {short class name}="{fully-qualified class name}"/>
```

where a fully-qualified class name can have the form

```
"{full (namespace-qualified) class name}, {assembly name}, ↵
    Culture='{culture}', PublicKeyToken={hexadecimal token}, ↵
    Version={version a.b.c.d}"
```

A `cryptoClass` element associates a short class name with a specific class in a specific assembly. An unusual aspect of this element is that the user-defined short class name is an attribute label rather than a value. The set of these elements serves as a dictionary of class nicknames that can be used in subsequent `nameEntry` elements. The short class names defined by the `cryptoClass` elements in each configuration file should be consistent across the enterprise. These are simply abbreviated forms of the class name and shouldn't be used to describe a policy-driven use for these classes.



Recommendation: Use short class names as abbreviated forms of the algorithm classes. Do not use short class names to express policy-driven roles such as algorithm defaults or use conditions.

When creating `cryptoClass` elements, it is important to precisely specify each component of the fully-qualified class name. If any component is incorrect, it will fail to be applied, and the default mappings will be used instead. Correct component values may be obtained from any assembly installed in the GAC by viewing the GAC using the Assembly Cache Viewer Explorer shell extension, right-clicking the assembly, and viewing its properties.

The <nameEntry> element

These elements have the form

```
<nameEntry name="{friendly name}" class="{short class name}"/>
```

A `nameEntry` element associates a "friendly name" with a pre-defined short class name. More than one friendly name can be associated with the same class through multiple `nameEntry` elements. Each of the cryptographic service names in Table 12 is a friendly name by default. There are no default short class names.

Software that is written to depend on specific implementations of cryptographic services may not be portable to environments where those services aren't available due to policy or

incompatibility. `nameEntry` elements allow software to be flexible enough to respond to changes in security policy that affect an organization's cryptographic standards or guidelines. Software can adapt the use of cryptography to various local execution environments by referencing cryptographic resources by names that have locally defined referents. Policy-driven cryptographic localization is achieved by mapping custom policy-based friendly names to specific algorithms and overriding the default cryptographic configuration by remapping existing friendly names to substitute implementations.

A friendly name can express the policy-driven use of the algorithm in an organization or a host. For example, a `nameEntry` element can associate the friendly name "DefaultHashAlgorithm" with a specific hash algorithm implementation through a pre-defined short class name ("SHA1_MyCryptoLibrary", "MD5Scheme48", etc.). Software can then use the friendly name to invoke the cryptographic service through the library defined by the associated short class name, without knowing in advance what specific hash algorithm is the default in the local execution environment.



Recommendation: Use friendly names to express the policy-driven roles played by particular algorithm classes (identified by their short class names) in the local execution environment. These roles can be defaults (e.g., "DefaultHashAlgorithm") or use conditions (e.g., "FinancialDataEncryption").

Although the configuration of friendly names is chiefly designed to support the resolution of generic friendly names (i.e., "DSA") into specific library code, even implementation-based friendly names (i.e., names based on existing library code such as "System.Security.Cryptography.DSACryptoServiceProvider") can be configured to invoke a substitute implementation.

In short, `nameEntry` elements allow the administrator to specify local cryptographic standards. Software can then implement these standards by invoking cryptographic resources by name rather than requesting a specific class and software library. Note that software can still invoke any CLR-provided cryptographic class directly, so cryptographic localization can be achieved only through a partnership with trusted software developers.

The `<oidEntry>` element

These elements have the form

```
<oidEntry name="{friendly name}" OID="{object identifier}"/>
```

An `oidEntry` element maps friendly names to ASN.1 object identifiers (OIDs). These are globally recognized identifiers for cryptographic algorithms or formats. Some standards for exchanging cryptographic products such as digital certificates between parties allow the use of arbitrary cryptographic algorithms for hashing or encryption. In order for the receiving party to verify the data received, they must use the same algorithms that the sending party

used to create the data. By embedding the OIDs of the algorithms used in the data, the sending party can communicate this information in a standardized way.

An application may need to record the OID of a cryptographic service that it used through a friendly name. Since the purpose of the friendly name is to allow local cryptographic policy to determine the algorithms used, the application may not know what algorithm was invoked. A mapping between friendly names and OIDs must be used to determine what OID to record. The .NET Framework libraries for version 1.1 come with a default mapping shown in Table .

Friendly Name	OID
“SHA1”	1.3.14.3.2.26
“System.Security.Cryptography.SHA1”	1.3.14.3.2.26
“System.Security.Cryptography.SHA1CryptoServiceProvider”	1.3.14.3.2.26
“System.Security.Cryptography.SHA1Managed”	1.3.14.3.2.26
“SHA256”	2.16.840.1.101.3.4.1
“System.Security.Cryptography.SHA256”	2.16.840.1.101.3.4.1
“System.Security.Cryptography.SHA256Managed”	2.16.840.1.101.3.4.1
“SHA384”	2.16.840.1.101.3.4.2
“System.Security.Cryptography.SHA384”	2.16.840.1.101.3.4.2
“System.Security.Cryptography.SHA384Managed”	2.16.840.1.101.3.4.2
“SHA512”	2.16.840.1.101.3.4.3
“System.Security.Cryptography.SHA512”	2.16.840.1.101.3.4.3
“System.Security.Cryptography.SHA512Managed”	2.16.840.1.101.3.4.3
“MD5”	1.2.840.113549.2.5
“System.Security.Cryptography.MD5”	1.2.840.113549.2.5
“System.Security.Cryptography.MD5CryptoServiceProvider”	1.2.840.113549.2.5
“System.Security.Cryptography.MD5Managed”	1.2.840.113549.2.5
“TripleDESKeyWrap”	1.2.840.113549.1.9.16.3.6

Table 13. Default Friendly Names for OIDs in the .NET Framework version 1.1.

There are some inconsistencies: “SHA-256”, “SHA-384”, and “SHA-512” are default friendly names, but are not associated by default with the OIDs for their respective algorithms. Note also that “System.Security.Cryptography.MD5Managed” is associated with the OID for the MD5 algorithm, but this is not a default friendly name with respect to any implementation of MD5. In fact, there is no class named System.Security.Cryptography.MD5Managed in version 1.1.4322 of the .NET Framework. “TripleDESKeyWrap” is also not defined as a friendly name with respect to any existing implementation of the Triple-DES Key Wrap algorithm.

The `oidEntry` elements defined in the cryptographic configuration file can supplement these default mappings or override them if the friendly names are the same. New `oidEntry` elements should be created whenever one of the following is true:

- When a friendly name that has a default OID is overridden in a `nameEntry` element with a different algorithm. In this case, the `oidEntry` element must be created to override the default OID and associate the correct OID to the friendly name.
- When a new friendly name is defined, even if it is configured to refer to an existing cryptographic service that already has an associated OID. In this case, a corresponding `oidEntry` element should be created to explicitly map the new name to its appropriate identifier.
- When a new named cryptographic service is installed. Since the name of a cryptographic service (e.g., “ExampleCorporation.Cryptography.AES256CBC”) may serve as a default friendly name, it can be used by software to get the corresponding OID (“2.16.840.1.101.3.4.1.42”) from the mapping defined by the `oidEntry` element.

Cryptographic Localization Examples

Cryptographic Localization Example 1

The SHA-512 hash algorithm has been selected by an organization as the default version of SHA for use by managed code. However, division A has some critical interoperability requirements with external partners that dictate that SHA-1 be employed when communicating with those partners. The .NET Framework administrator can support both policies by configuring cryptography in each .NET Framework version on each host in the organization to specify the appropriate default algorithms by the names “DefaultSHA” and “DefaultSHAInterop”. This allows organization-wide software applications to localize their cryptographic features by explicitly invoking default algorithms by name.

Since the `cryptoClass` elements should simply define an organization-wide dictionary of cryptographic services, the administrator defines short class names “SHA512Managed_v1_0” and “SHA1Managed_v1_0” for implementations of both SHA-512 and SHA-1, respectively, in all configuration files.

In all configuration files, there is also a `nameEntry` element that associates the friendly name “DefaultSHA” with the short class name “SHA512Managed_v1_0”. In the configuration files for hosts in division A, there is an additional `nameEntry` element that associates the friendly name “DefaultSHAInterop” with “SHA1Managed_v1_0”.

Cryptographic configuration common to all hosts:

```
<cryptoClass SHA512Managed_v1_0="System.Security.Cryptography.
    SHA512Managed, mscorlib, Culture='',
    PublicKeyToken=b77a5c561934e089, Version=1.0.3300.0"/>
<cryptoClass SHA1Managed_v1_0="System.Security.Cryptography.SHA1Managed,
    mscorlib, Culture='', PublicKeyToken=b77a5c561934e089,

```

```

    Version=1.0.3300.0"/>
...
<nameEntry name="DefaultSHA" class="SHA512Managed_v1_0"/>
<nameEntry name="SHA" class="SHA512Managed_v1_0"/>

```

Figure 33. Cryptographic Localization Example 1.

Note that since “SHA” is a default friendly name, the administrator must also create a `nameEntry` element that overrides the default implementation. Hosts in division A have the additional `nameEntry` element:

```
<nameEntry name="DefaultSHAInterop" class="SHA1Managed_v1_0"/>
```

Cryptographic Localization Example 2

In order to support policy-driven rather than developer-driven use of cryptography for encryption algorithms, an organization has worked with a trusted software supplier to consistently invoke encryption algorithms through the default friendly names “System.Security.Cryptography.SymmetricAlgorithm” and “System.Security.Cryptography.AsymmetricAlgorithm”. A cryptographic configuration can then be developed to specify the algorithms employed. If policy is changed to specify a different set of algorithm, the software will apply the new policy as soon as its corresponding configuration is deployed.

Short class names are defined for all the algorithm implementations that are permitted by policy. In this example, the permitted algorithms are System.Security.Cryptography.RijndaelManaged and a custom implementation of DSA in the class Example.Security.DSAEncryption, as shown in Figure 34:

```

<cryptoClass RijndaelManaged="System.Security.Cryptography.
    RijndaelManaged, mscorlib, Culture='',
    PublicKeyToken=b77a5c561934e089, Version=1.0.3300.0"/>
<cryptoClass DSAEncryption="Example.Security.DSAEncryption,
    ExampleCryptLib, Culture='', PublicKeyToken=0011223344556677,
    Version=1.0.0.0"/>

```

Figure 34. Cryptographic Localization Example 2-A.

`nameEntry` elements will dictate the behavior invoked by the default friendly names so that System.Security.Cryptography.RijndaelManaged is invoked anytime a generic symmetric encryption algorithm is requested, and Example.Security.DSAEncryption is invoked whenever a generic asymmetric encryption algorithm is requested (Figure 35).

```

<nameEntry name="System.Security.Cryptography.SymmetricAlgorithm"
    class="RijndaelManaged"/>

```

```
<nameEntry name="System.Security.Cryptography.AsymmetricAlgorithm"
  class="DSAEncryption"/>
```

Figure 35. Cryptographic Localization Example 2-B.

Cryptographic Localization Example 3

An organizational policy directs that the 256-bit version of the Advanced Encryption Standard be implemented through a custom cryptographic library `CryptLib.dll`, and that this be the default symmetric encryption algorithm. Furthermore, any requests for the Rijndael algorithm should be redirected to this library. To accomplish this, the short name "AES256" is associated with a class of the same name in the library.

`nameEntry` elements are defined that associate the default friendly names "System.Security.Cryptography.SymmetricAlgorithm", "System.Security.Cryptography.Rijndael", "SymmetricAlgorithm", "Rijndael", and "System.Security.Cryptography.RijndaelManaged" with the short class name "AES256". In addition, the new friendly names "BlockCipher" and "AES" are defined (Figure 36).

```
<cryptoClass AES256="CryptLib.Algorithms.AES256, CryptLib Culture='',
  PublicKeyToken=fedcba9876543210, Version=1.0.0.0"/>
...
<nameEntry name="System.Security.Cryptography.SymmetricAlgorithm"
  class="AES256"/>
<nameEntry name="System.Security.Cryptography.Rijndael" class="AES256"/>
<nameEntry name="SymmetricAlgorithm" class="AES256"/>
<nameEntry name="Rijndael" class="AES256"/>
<nameEntry name="System.Security.Cryptography.RijndaelManaged"
  class="AES256"/>
<nameEntry name="BlockCipher" class="AES256"/>
<nameEntry name="AES" class="AES256"/>
```

Figure 36. Cryptographic Localization Example 3.

Summary

The .NET Framework provides a way to support local cryptographic standards through the use of a cryptographic configuration section in the machine configuration file for each .NET Framework version. Because this file contains configuration settings for several aspects of the .NET Framework, deployment is a challenge. Because the cryptography settings must be modified by hand, while other parts of the file are modified by automated tools, stability is also a challenge, and care must be taken to create valid XML. Nevertheless, the ability to localize the use of cryptography is a valuable feature of the .NET Framework.

The localization is supported administratively through the creation of a dictionary of short class names that are abbreviated implementation specifications. These are then used in a many-to-one mapping of friendly names for cryptographic services. Since managed code can always invoke any specific implementation, localization can only be achieved with the cooperation of software developers.

Recommendations in This Section

-  ***Recommendation: Use short class names as abbreviated forms of the algorithm classes. Do not use short class names to express policy-driven roles such as algorithm defaults or use conditions.***

-  ***Recommendation: Use friendly names to express the policy-driven roles played by particular algorithm classes (identified by their short class names) in the local execution environment. These roles can be defaults (e.g., “DefaultHashAlgorithm”) or use conditions (e.g., “FinancialDataEncryption”).***

Administrative Tasks and Tools

Administering the .NET Framework involves a number of security-related tasks that cannot all be accomplished with a single tool. Several tools are distributed with the .NET Framework or the .NET Framework Software Developer's Kit (SDK) to perform these tasks. This chapter will describe some key security-related administrative tasks and list the tools used to perform each task. Guidelines and procedures for the performance of each task are included under the description of each tool in Appendix A or B.

Security-Related Administrative Tasks Summary

Table 14 lists some security-related administrative tasks in the .NET Framework and the tools used to perform them:

Task	Tool
General .NET Framework tasks	
List the .NET Framework versions installed	migppol.exe
Global Assembly Cache tasks	
Enable or disable the Assembly Cache Viewer	regedit.exe
View cache contents	explorer.exe gacutil.exe mscorcfg.msc
Add an assembly to the GAC	explorer.exe gacutil.exe mscorcfg.msc
Delete an assembly from the GAC	explorer.exe gacutil.exe mscorcfg.msc
View properties of an assembly installed in the GAC	explorer.exe mscorcfg.msc
View or modify GAC properties	explorer.exe regedit.exe
Clear the Download Cache	gacutil.exe
Isolated Storage tasks	
List all local or roaming data stores associated with the current user	storeadm.exe
Remove all local or roaming data stores associated with the current user	storeadm.exe
Code Access Security policy tasks	

UNCLASSIFIED

Task	Tool
Migrate CAS policy from one .NET Framework version to another	migppol.exe
Create a CAS policy deployment package	mscorcfg.msc
Enable or disable CAS policy	caspol.exe
Enable or disable Execution permission checking	caspol.exe
Build a CAS policy cache file	caspol.exe
Reset all CAS policy levels to default settings	caspol.exe mscorcfg.msc
Recover the previous settings for a CAS policy level	caspol.exe
View Code Groups	caspol.exe mscorcfg.msc
Add or remove a Code Group	caspol.exe mscorcfg.msc
Rename a Code Group	caspol.exe mscorcfg.msc
Set or clear the Exclusive or Level Final attribute of a Code Group	caspol.exe mscorcfg.msc
Change a Code Group's Membership Condition	caspol.exe mscorcfg.msc
Change a Code Group's associated Named Permission Set	caspol.exe mscorcfg.msc
View Named Permission Sets	caspol.exe mscorcfg.msc
Add or remove a Named Permission Set	caspol.exe mscorcfg.msc
Modify a Named Permission Set	caspol.exe mscorcfg.msc
View Policy Assemblies	caspol.exe mscorcfg.msc
Enroll or withdraw a Policy Assembly	caspol.exe mscorcfg.msc
List Code Groups to which an assembly belongs	caspol.exe mscorcfg.msc
View an assembly's Allowed Permission Set	caspol.exe mscorcfg.msc
Adjust the Allowed Permission Set for an assembly	caspol.exe mscorcfg.msc
Create a tailored Code Group	caspol.exe mscorcfg.msc
Use the Trust an Assembly Wizard	mscorcfg.msc
System security tasks	
View publisher certificate verification settings	setreg.exe
Adjust publisher certificate verification settings	setreg.exe
Set the CSP used by the CLR when strong-naming assemblies	sn.exe
Assembly tasks	
Validate and verify an assembly	peverify.exe

Task	Tool
View an assembly's strong name	explorer.exe secutil.exe
View the public key token corresponding to the public key in an assembly's manifest	sn.exe
Strong name an assembly	sn.exe
Verify an assembly's strong name	sn.exe
Enroll an assembly for strong name simulation	sn.exe
Withdraw an assembly from strong name simulation	sn.exe
List assemblies enrolled for strong name simulation	sn.exe
View an assembly's publisher certificate	secutil.exe certmgr.exe
Verify the trust associated with an assembly's Authenticode digital signature	chktrust.exe
View an assembly's permission requests and declarative permission constraints	permview.exe
View the list of configured assemblies	mscorcfg.msc
Configure an assembly	mscorcfg.msc
Delete the configuration information for an assembly	mscorcfg.msc
Application configuration tasks	
Add an application to be configured	mscorcfg.msc
Configure application properties	mscorcfg.msc
View assembly dependencies for an application	mscorcfg.msc
View list of configured assemblies for an application	mscorcfg.msc
Configure an assembly for an application	mscorcfg.msc
Fix an application (roll back application Binding Policy)	mscorcfg.msc
Configure Remoting Services for an application	mscorcfg.msc

Table 14. Security-Related Administrative Tasks.

Administrative Tools Summary

The .NET Framework is distributed with a number of tools for performing configuration, deployment, debugging, security, and other types of tasks. The .NET Framework SDK provides some additional tools that are helpful for the administrator. Table 15 below lists tools used to perform security-related administrative tasks in the .NET Framework. Tools distributed with the .NET Framework SDK are indicated by (SDK) following the tool name.

All of the tools are console applications except for the Assembly Cache Viewer (`shfusion.dll`) which is a Windows shell extension, the .NET Framework Configuration Tool (`mscorcfg.msc`) which is a Microsoft Management Console snap-in, and `certmgr.exe` which presents a GUI interface when invoked from the console with no parameters. Appendix A contains guidelines and procedures for using these tools to perform the administrative tasks list in Table 14.

Tool	Filename
.NET Framework Code Access Security Policy Tool	caspol.exe
Microsoft Certificate Manager Tool	certmgr.exe (SDK)
Microsoft Authenticode Signature Verification Tool	chktrust.exe (SDK)
Assembly Cache Viewer (Windows shell extension)	explorer.exe (shfusion.dll)
.NET Global Assembly Cache Utility	gacutil.exe (SDK)
CAS Policy Migration Tool	migpol.exe
.NET Framework Configuration Tool	mscorcfg.msc
.NET Framework Permission Request Viewer	permview.exe (SDK)
.NET Framework PE Verifier	peverify.exe (SDK)
Registry Editor	regedit.exe
Microsoft .NET Framework Security Utility	secutil.exe (SDK)
Software Publishing State Tool	setreg.exe (SDK)
.NET Framework Strong Name Utility	sn.exe (SDK)
.NET Framework Isolated Storage Tool	storeadm.exe (SDK)

Table 15. Administrative Tools.

Task Descriptions

General .NET Framework Tasks

List the .NET Framework versions installed

Since multiple versions of the .NET Framework can be installed side-by-side, the administrator may need a means to determine which versions of the .NET Framework are installed on a particular host computer. This task can be performed by navigating to the folder %Windir%/Microsoft.NET/Framework. Each installed version of the .NET Framework has a subfolder of this folder. The names of the subfolders correspond to the .NET Framework version numbers. This task can also be performed in a scriptable manner using migpol.exe.

Global Assembly Cache tasks

Enable or disable the Assembly Cache Viewer

Because the file system uses only file names as an identifier, the GAC holds multiple versions of the same assembly in separate subfolders under the “%WINDIR%\assembly” folder. When the .NET Framework is installed, a Windows shell extension (shfusion.dll) is registered that provides a “fused” display in Windows Explorer of all the assemblies installed in the subfolders that comprise the GAC. When this shell extension is enabled, multiple assemblies with the same name appear in a unified list. This display can be enabled or disabled by setting the value of a registry key using the registry editor regedit.exe. For details, see: shfusion.dll: Enable or Disable the Assembly Cache

Viewer or `regedit.exe`: Enable or disable the Assembly Cache Viewer in Appendix B. Note that navigating the file system via the console will show the actual folders and their contents. The Assembly Cache Viewer shell extension applies only to file system navigation through Windows Explorer.

View cache contents

The GAC contents can be viewed by navigating to “%WINDIR%\assembly” in Windows Explorer or the console, through the .NET Framework configuration tool `mscorcfg.msc`, or through the GAC console utility `gacutil.exe`. Navigating to the GAC through the console is the least useful of these options, since it does not show a combined list of all assemblies installed in the GAC. Instead, it shows the underlying file system folder tree that enables side-by-side installation of multiple versions of the same assembly. To list the assemblies installed in the GAC, it is preferable to view the “fused” display.

This task can be performed using `explorer.exe` with the Assembly Cache Viewer enabled, `gacutil.exe`, or `mscorcfg.msc`.

Add an assembly to the GAC, Delete an assembly from the GAC, View properties of an assembly installed in the GAC

The GAC is intended to hold assemblies that will be shared by multiple applications. If an assembly is for the exclusive use of one application, it should only be stored in the same directory as the application. Unless granted expansive access to the file system, other applications and assemblies won't be able to access the assembly – it is private to its intended application. On the other hand, GAC assemblies are available for use by any assembly. Since the file name of an assembly could create naming conflicts, shared assemblies must be uniquely identified by a strong name before installation into the GAC. This does not provide any evidence of trustworthiness. It only solves the name collision problem. The default CAS policy grants unrestricted access to resources to assemblies originating from the local computer (the My Computer zone), which includes all assemblies installed in the GAC. However, assemblies in the GAC could run with varying levels of access if CAS policy is configured that way.

These tasks can be performed using `explorer.exe` with the Assembly Cache Viewer enabled, `gacutil.exe`, or `mscorcfg.msc`.

View or modify GAC properties, Clear the Download Cache

The only GAC property that is configurable is the maximum size of the Download Cache. The default size of the Download Cache is 4608KB. To modify this property, use `explorer.exe` with the Assembly Cache Viewer enabled.

The Download Cache holds assemblies that have been obtained from a remote host via a local UNC path or via a Web URL. These assemblies are temporarily stored on the local machine in this cache, but their association with their place of origin is maintained to use as evidence when applying CAS policy to grant access to resources. Since the Download Cache

has a configurable maximum size, its resource usage does not need to be frequently monitored. Clearing the Download Cache is more likely to be useful in a development environment where testing requires the most recent version of an assembly to be used, especially when Web applications are being developed. This task can be performed using `gacutil.exe`.

Isolated Storage tasks

List Isolated Storage File stores (including roaming) for the current user, Remove all Isolated Storage File stores (including roaming) for the current user

The Isolated Storage facility provides a way for partially trusted assemblies to save information for later use without directly accessing local machine resources such as the file system or registry. This facility is similar to “cookies” used by Web applications. Isolated Storage is a protected resource in the .NET Framework that is implemented through a folder tree within each user’s local and/or roaming profile folder. The Isolated Storage facility can be managed using `storeadm.exe`.

Code Access Security policy tasks

Migrate CAS policy from one .NET Framework version to another

Multiple versions of the .NET Framework can be installed on the same computer at the same time. Each version will have its own CAS policy. Applications will run with the .NET Framework version for which they were developed, and will be granted permissions based on the CAS policy of the version that is managing their execution. Keeping CAS policy settings consistent across multiple versions of the .NET Framework is a key security-related administrative task, as changes made to the policy for one version will not automatically be mirrored on other versions.

Custom software can be developed to extend the CAS policy system by defining new types of Code Groups, Membership Conditions, or Permissions. This is different from creating user-defined Code Groups and Named Permission Sets – it involves creating software libraries that define new types of policy components that behave differently from the built-in types. For example, a new Membership Condition called `WorkingJoeMembershipCondition` could be defined that is satisfied if the code being checked is executing between the hours of 9am and 5pm and is running in the context of a user account named “Joe.”

New CAS policy components are designed to run with specific versions of the .NET Framework. Each version of the .NET Framework would need to have its own version of these special software extensions. Thus, it is inappropriate to simply cut and paste references to these extensions in one CAS policy file into a CAS policy file for a different version of the .NET Framework. Currently, there is no automated tool that assists the administrator in migrating configuration file references to CAS policy extensions developed for one version of the .NET Framework to another version. Where CAS policy contains user-defined Code Groups and Named Permission Sets that rely only on the built-in policy components, the tool `migpol.exe` can be used to migrate policy.

Create a CAS policy deployment package

A Windows Installer package (.msi file) can be created for a particular CAS policy level and .NET Framework version using `mscorcfg.msc`. For a discussion of CAS policy deployment, see Chapter 3.

Enable or disable CAS policy

The entire CAS policy access control system can be disabled, allowing all assemblies to run as if they were unmanaged native executables. This is clearly a very dangerous thing to do, and should never be done while a computer is connected to a network, especially the Internet.



Recommendation: Never disable CAS policy on a computer connected to an untrusted network such as the Internet.

This task can be performed using `caspol.exe`.

Enable or disable Execution permission checking

A performance-enhancing feature checks assemblies for Execute permission early in the policy application process. If this permission is not granted, the assembly will fail to load. This feature can be disabled, although this would rarely, if ever, be done. This task can be done using `caspol.exe`.

Build a CAS policy cache file

The use of CAS policy cache files (.cch) is not documented. Cache files are automatically created by the CLR when a policy is first used, so administratively building a new cache file is generally unnecessary. This task can be performed using `caspol.exe`.

Reset all CAS policy levels to default settings

When CAS policy becomes corrupted or a host computer changes roles and needs its CAS policy rebuilt from the ground up, the default policy can be restored. This can be done through `mscorcfg.msc` or by simply deleting the existing CAS policy configuration files, but it can also be scripted using the console application `caspol.exe`.

Recover the previous settings for a CAS policy level

After making a change in the CAS policy configuration through `mscorcfg.msc` or `caspol.exe`, the previous configuration is automatically saved in a backup file. The previous configuration can be recovered using `caspol.exe`.

View Code Groups, Add or remove a Code Group, Rename a Code Group, Set or clear the Exclusive or Level Final attribute of a Code Group, Change a Code Group's Membership Condition, Change a Code Group's associated Named Permission Set, View Named Permission Sets, Add or remove a Named Permission Set, Modify a Named Permission Set

Code Groups and Named Permissions Sets can be viewed, added, removed, or modified using `mscorcfg.msc` or `caspol.exe`. Changing the Named Permission Set associated with one of the built-in Zone-based Code Groups can also be performed using `mscorcfg.msc` with the Security Adjustment Wizard. See the Adjust Zone Security task in `mscorcfg.msc` for details.

View Policy Assemblies, Enroll or withdraw a Policy Assembly

CAS policy is extensible by design, with the ability to recognize and apply custom software components such as new Code Group types, Membership Conditions, and resource permissions. The managed libraries that contain the definitions for these custom elements are called Policy Assemblies. Policy Assemblies are used by the CLR to apply CAS policy to executable code, and thus must be Fully Trusted themselves. All Policy Assemblies must be strong named, installed in the GAC, and enrolled in each CAS policy level configuration file in which their custom elements are used. This enrollment allows the CLR to find and use the definitions of security components they contain, as well as to automatically consider them Fully Trusted at the specified policy level. This means that they will automatically be granted unrestricted access to resources at that policy level, regardless of Code Group membership. This access is still subject to intersection with Enterprise, User, and Application Domain level policies.

If a Policy Assembly needs other assemblies to function, the other assemblies must first be added as Policy Assemblies. This is true even if the assemblies used are distributed with the .NET Framework or are from Microsoft.

Since the access control mechanism of the .NET Framework relies on the managed libraries, built-in or custom, that define the CAS policy elements, Policy Assemblies must be developed by trusted parties using the best practices for design and coding of security functions.

The presence of references to custom CAS policy components and their associated Policy Assemblies may affect the migration of CAS policy from one version of the .NET Framework to another. See the section on `migpol.exe` below for details.

Policy Assemblies can be viewed, enrolled, or withdrawn using `mscorcfg.msc` or `caspol.exe`.

List Code Groups to which an assembly belongs, View an assembly's Allowed Permission Set

Once created, the effect of a CAS policy configuration on a particular assembly can be partially tested by resolving Code Group Membership Conditions against an assembly's evidence. The resulting set of Code Group memberships or the set of associated permissions can give an administrator an idea of the actual permissions that would be granted to the assembly if executed. If the assembly is specified via a URL, this could take into account Site or URL evidence as well.

Only the administratively configured elements of CAS policy are included in the Allowed Permission Set test. Permission requests (including refused permissions) made by the assembly itself are not included in this computation, nor is any Application Domain level policy, as these elements are not administratively configured. Permission requests are configured by software developers, and Application Domain policy is set by a hosting application. However, the set of permissions that an assembly is allowed based on its evidence and the administratively configured policy will be an upper bound on the final granted set of permissions.

These tasks can be performed using `mscorcfg.msc` or `caspol.exe`.

Adjust the Allowed Permission Set for an assembly, Create a tailored Code Group, Use the Trust an Assembly Wizard

The Allowed Permission Set for an assembly can be modified by changing the Named Permission Sets for existing Code Groups to which it belongs, or by creating a new Code Group that will contribute permissions to its Allowed Permission Set. The Allowed Permission Set associated with the assembly from the existing Code Groups can be checked with the View an assembly's Allowed Permission Set task. If these permissions are insufficient and the assembly needs additional permissions, the Trust an Assembly Wizard in `mscorcfg.msc` can be used to create a new Code Group that allows permissions from one of the built-in Named Permission Sets. See the Adjust the Allowed Permission Set for an assembly task under `mscorcfg.msc` for more details. Alternatively, the CAS policy settings can be individually configured to allow the desired permissions. To do this, perform the following for each desired additional permission:

- For each existing Code Group that contains the targeted assembly, determine if the Membership Condition of the Code Group defines a class of assemblies that may appropriately receive the additional permissions. If so, modify the Named Permission Set associated with this existing Code Group to include the given permission.

If there are remaining permissions that cannot be supported for the targeted assembly by the existing Code Group hierarchy, then a new Code Group will have to be created that contains the targeted assembly and contributes the remaining desired permissions. This is discussed below under Creating a Tailored Code Group.

If the assembly's Allowed Permission Set is already too permissive, and the assembly needs to be restricted, perform the following steps for each unwanted permission:

- For each existing Code Group that contains the targeted assembly and allows the unwanted permission, determine if the permission can be removed from the associated Named Permission Set without affecting the functioning of other assemblies contained in the Code Group. If so, then modify the Named Permission Set (or create a new if the Named Permission Set is one of the built-in sets) to remove the unwanted permission. If this is not possible, move to the next step.
- For every existing Code Group that contains the targeted assembly and must allow the unwanted permission, determine if the Membership Condition can be modified to exclude the targeted assembly without excluding other assemblies that the Code Group was intended to configure. If so, modify the Membership Conditions of all the Code Groups to exclude the targeted assembly. This must be possible for all such Code Groups in order to be effective.

If there are remaining unwanted permissions that are allowed by the existing Code Groups, then create a new Code Group tailored to the targeted assembly and the appropriate permissions and set the Exclusive attribute. This could be done at either the Enterprise or Machine level. If this would create more than one Code Group marked Exclusive for this assembly, review all such Code Groups and refine each Membership Condition until this condition is removed. The Exclusive attribute should be placed only on Code Groups with narrowly tailored Membership Conditions.

Creating a Tailored Code Group

To create a Code Group tailored to a specified assembly and a specific set of permissions, perform the following steps in either `caspol.exe` or `mscorcfg.msc` (see the Create a tailored Code Group task for details):

- Create a Named Permission Set containing only the specified permissions, if one does not already exist.
- Create a Code Group whose Membership Condition will discriminate between the targeted assembly and other assemblies. Set the Membership Condition to discriminate as little as possible commensurate with operational requirements and organizational policy. At the same time, insist on cryptography-based identities unless prohibited by policy constraints. Associate the new Named Permission Set with this Code Group.

The Membership Conditions identify a class of assemblies based on a spectrum of identifying information of varying specificity. Table 16 lists the Membership Conditions in increasing order of discriminating power.

UNCLASSIFIED

Membership Condition	Description
Non-Cryptography-Based Identities	
All Code	This Membership Condition offers no discriminating power.
Internet Zone Local Intranet Zone My Computer Zone	“Security by Proximity.” The Zone Membership Conditions of Internet, Local Intranet, and My Computer define “Security by Proximity.” Code obtained locally is considered more trustworthy than code obtained from “far” away. This is the coarsest method of granting access to resources, and emphasizes extreme scalability over security.
Trusted and Untrusted Zones Site URL	Webpace Identity. These Membership Conditions define a subspace of the network environment of a host, including both local host resources as well as networked resources, including the Internet. Even a URL that specifies file name X is really specifying the class of all files that could be named or renamed ‘X’ at that path location, i.e., all files that could possibly be referenced through that URL.
Cryptography-Based Identities	
Publisher	Organizational Identity. This Membership Condition identifies a particular organization based on a digital certificate. If the organization develops and distributes multiple lines of software products, this does not discriminate between them.
Strong Name without version Strong Name with version	Functional Identity. These Membership Conditions identify a particular class of software. The public/private key pair used to strong name an assembly may identify a class of software products developed by an organization. The name of the assembly may further identify a particular product. The version number identifies a particular release of the product. This is a functional identity in that it is based on what the software developers consider to be functionally equivalent releases of software. A modified assembly could be released with the same key pair, name, and version, if the developers consider it to be the “same” assembly under their criteria. Different developers may have different strategies for deciding how much change constitutes a new version of an assembly. Functional identity indicates equivalence within the software developer’s versioning scheme. A strong name Membership Condition without the version information will apply to all versions of the assembly, and thus has less discriminating power than specifying an exact version.

Membership Condition	Description
Hash	Logical Identity. This Membership Condition identifies a specific sequence of bytes.

Table 16. Discriminating Power of Membership Conditions.

System security tasks

View publisher certificate verification settings, Adjust publisher certificate verification settings

A set of registry keys called the Software Publishing State keys govern how Authenticode digital signatures are verified. These settings affect how Publisher Membership Conditions are evaluated, which can affect CAS policy application. The ten Software Publishing State values are listed in Table 17:

Number	Description	Default Value
1	Trust Test Root certificates.	False
2	Check for expired certificates	True
3	Check for revoked certificates	False if Internet Explorer 3.x is installed True if Internet Explorer 4.0 or later is installed.
4	Offline revocation server OK (individual)	False
5	Offline revocation server OK (commercial)	False for Microsoft Windows versions prior to Windows 2000 True for Windows 2000 and later.
6	Java offline revocation server OK (individual)	False
7	Java offline revocation server OK (commercial)	False for Microsoft Windows versions prior to Windows 2000 True for Windows 2000 and later.
8	Invalidate version 1 signed objects	False
9	Check the revocation list on time stamp signer	False, Only applies if Internet Explorer 4.0 and later is installed.
10	Only trust items found in the Personal Trust Database	False, Only applies if Internet Explorer 4.0 and later is installed.

Table 17. Software Publishing State Settings.

Viewing and adjusting publisher certificate verification settings can be performed using `setreg.exe`. These settings are discussed in more detail below.

Setting 1: Trust Test Root Certificates

Certificates used for testing and debugging are signed by root certificates designated as being for test purposes. If this setting is enabled, certificates created for testing and debugging purposes will be considered trusted. This setting should only be enabled in protected development networks.



Recommendation: Disable trust of Test Root certificates in an operational environment.

This setting is only used in Windows versions prior to Windows XP SP1. Windows XP SP1 and Windows Server 2003 never automatically trust the test root. In these operating systems, the value of this setting is ignored – the system always behaves as if it were false.

Setting 2: Check for Expired Certificates

Virtually all digital certificates are issued with a moderately short lifespan (one year is common). This helps to decrease loss associated with undetected private key compromise and, in accord with the security principle of periodic risk assessment, provides a scheduled review of the need for the certificate and a reconsideration of the trust relationship between the certificate subject and the issuer. Operational environments should always check for expired certificates.



Recommendation: Enable checking for expired certificates.

Software that is signed by a publisher's certificate may be needed for far longer than the certificate's lifespan. Either the software must be resigned and redeployed each time the publisher is issued a new certificate, or the signing process must include a digitally signed time stamp that can demonstrate that the software was signed before the certificate expired. The certificate verification process in Authenticode 2.0 (available since Internet Explorer 4.0) supports the assignment of trust in the latter case. Code that has been verifiably signed within the lifespan of a publisher's certificate is considered to have a valid signature, even if the certificate has expired. Note that the time stamp must itself be signed by the time stamp provider with a non-expired certificate. Setting 9 deals with whether the time stamp provider's certificate is checked for revocation.

The .NET Framework is designed to facilitate distributed applications that download components as needed from Web servers. In this environment, it is not as great a burden on software publishers to periodically resign code, since deployment through a Web site is discretionary and "as-needed". Nevertheless, it is not possible to prevent code that is signed within the lifespan of a publisher's certificate from satisfying the Publisher Membership Condition for a Code Group. For this reason, the Publisher Membership Condition should only be used for publishers with a well-established history of trustworthiness.



Recommendation: Only use the Publisher Membership Condition for software publishers with whom your organization has a well-established history of trust. Access to resources may be granted to code that presents expired certificates, so the use of the Publisher Membership Condition assumes that the publisher was trustworthy in the past as well as the present.

Setting 3: Check for Revoked Certificates

Digital certificates used to sign code may be revoked prior to their expiration date. Among other reasons, this may happen because the certificate subject is no longer trusted, because information about the certificate subject contained in the certificate has changed, because private key data has been or may have been compromised, or simply because the certificate is no longer needed. In any case, a revoked certificate should not be used for identification and authentication, including checking for membership in any code group that will grant access to protected resources.

If this setting is enabled, Certificate Revocation Lists will be checked to verify that certificates presented have not been revoked. Revoked certificates will not be considered valid. This setting can be modified through Internet Explorer using the **Tools | Internet Options...** menu. Select the **Advanced** tab and check the box in the **Security** category labeled **Check for publisher's certificate revocation**.



Recommendation: Enable checking for revoked certificates.

Settings 4-7: Automatically Trust Certificates Whose Revocation Status Cannot Be Determined

Each certificate issuing entity (Certificate Authority/CA) is responsible for providing information on the revocation status of the certificates it issues. A common way to do this is to make available a list (Certificate Revocation List/CRL) of its issued certificates that have been revoked. These lists are published by the CA to CRL Distribution Points (CDPs) on the local network or Internet. Each certificate issued by the CA contains the URL of one or more of these CDPs so any application or host verifying the certificate can check the issuer's lists to determine if the certificate is still valid. This is typically done by periodically downloading an updated list from the CDPs and caching it locally. Since each CRL has an assigned lifespan, it is easy to know when an updated list is needed. Moreover, requesting a CRL refresh rather than requesting the status of a single certificate only indicates that one from the set of certificates served by that CRL may be being verified, providing a degree of anonymity to the request. One issue with using CRLs is that they can become very large for Certificate Authorities that serve a large community, such as the Department of Defense, or well-established certificate services vendors. The performance and anonymity benefits of caching CRLs must be balanced against the large amount of data that must be transferred regularly and the potential delay in identifying revoked certificates.

An alternative approach to checking for certificate revocation is the use of real-time certificate status servers (Online Certificate Status Protocol/OCSP). These are network services hosted by a CA or its designee that provide up-to-the-minute revocation status for

single certificate requests, making the download of large lists unnecessary. The benefits in performance and timeliness must be balanced against the requirement for reliable network connectivity. In addition, the disclosure of transaction information such as which certificate identifier is being verified and when to an external party such as the revocation status server may be unacceptable.

The use of CRLs or OCSP may be determined by installed custom cryptographic software, so the actual means of checking revocation may not be known to applications or even to the operating system. Windows will simply invoke the installed cryptographic software to perform the revocation check function when appropriate in the certificate verification process. This ability to install custom cryptographic software supports the implementation of policy-driven cryptographic services. However, the actual methods used may have an impact on the ability to provide access control over distributed software such as .NET Framework assemblies.

The use of the Publisher Membership Condition for managed code assumes the presence and integrity of a public key infrastructure, including a robust process for certificate verification. Since decisions about trust are made each time the .NET Framework loads an assembly, the reliability and availability of the certificate verification process may determine whether code is able to execute or not when it is needed. Clearly, an OCSP implementation is more vulnerable to loss of availability than cached CRLs, but even CRLs will periodically expire and require a refresh. Where connectivity is expected to be intermittent, manual distribution of updated CRLs may be an option in some cases. However, in general, volatile network environments that have a critical need for the availability of certain managed applications may want to grant access to such code based on characteristics that are verifiable on the local host, such as a strong name or hash.

Where publisher certificates are used as a basis for CAS policy decisions, certificates should not be trusted unless their revocation status can be determined. If settings 4-7 are enabled, code signed by a revoked certificate could be automatically trusted if the network connection is broken or unreliable.



Recommendation: Disable automatic trust for certificates whose revocation status cannot be determined.

Setting 8: Invalidate Version 1 Signed Objects

If this setting is enabled, software signed using a Windows version 1 certificate will not be trusted. Windows version 2 certificates are new to Windows Server 2003 (Enterprise and Data Center).

Setting 9: Check for Revoked Timestamp Signer Certificate

Code that is signed may (and should) contain a time stamp issued by a time stamp provider that certifies that the code was signed within the lifespan of the publisher's certificate. The time stamp itself is digitally signed with the time stamp provider's certificate. This process allows some basis for trust to be assigned to software that is still in useful service even

though the publisher's certificate has expired. This basis is only as good as the verifiability of the time stamp, which in turn relies on the integrity of the time stamp provider's certificate. This certificate should be valid, non-expired, and non-revoked.

 **Recommendation: Enable checking for revoked time stamp provider's certificate.**

Setting 10: Only Trust Items Found in the Personal Trust Database

If this setting is enabled, software that is not signed by a publisher whose identity is stored in the user's Trusted Publishers and Issuers of Credentials database (trust database) will fail the certificate verification process. The logged-on user's trust database can be viewed through Internet Explorer using the **Tools | Internet Options...** menu. Select the **Content** tab, and click the **Publishers...** button.

Signed managed code will satisfy a Publisher Membership Condition only if it successfully passes the certificate verification process, so if an assembly is signed by a publisher not found in the user's trust database, it will be treated as if the publisher's signature was not present. Because such code will not satisfy the Publisher Membership Condition, the permissions associated with such a Code Group will be ignored and Code Group flags such as Exclusive or Level Final will not take effect. This setting will only affect the behavior of the Publisher Membership Condition. Code signed by a software publisher may still receive permissions by satisfying other types of Membership Conditions.

Set the CSP used by the CLR when strong-naming assemblies

The Cryptographic Service Provider used by the CLR to perform the cryptographic functions needed to strong name an assembly can be administratively specified. By default, the CLR uses the operating system's default CSP. This task can be performed using `sn.exe`.

Assembly tasks

Validate and verify an assembly

Assembly validation and verification is a process that determines whether an assembly is provably safe to run, that is, whether it can be known via an automated process if it can be effectively managed by the CLR or not. The CLR will refuse to execute code that fails the validation and verification process, unless it has been given the Skip Verification permission. Code granted the Skip Verification permission will be allowed to execute as long as it passes validity checks.

An assembly may consist of syntactically correct components, but not be valid. An assembly is valid if it is internally consistent and conforms to the expected file, metadata, and code instruction formats. Assemblies must be valid to be allowed to execute, although validity in and of itself does not indicate that an assembly is safe to run.

Execution safety is determined by the verification process. The .NET Framework performs a standard set of verification checks that aim to prove an assembly is safe to run. If an

assembly passes all of these checks, it is presumed to be safe. If an assembly fails any of these checks, it is presumed to be unsafe, even though it may in fact be safe. This ambiguity is due to the fact that code verifiers may not rely on all the information available about the assembly's code, and may assume code is unsafe in the absence of information to the contrary.

Figure 37 shows the relationship between validation and verification. Code that is given the Skip Verification permission will be allowed to execute if it falls within the inner three rings. Code that is not given the Skip Verification permission must fall within the innermost ring. Thus, the Skip Verification permission may allow code to execute that is in fact unsafe. The Skip Verification permissions is intended for use in situations where the CLR's code verifier cannot certify the safety of an assembly, but where the assembly is known to be safe nonetheless, due to additional documentation provided by the developers.

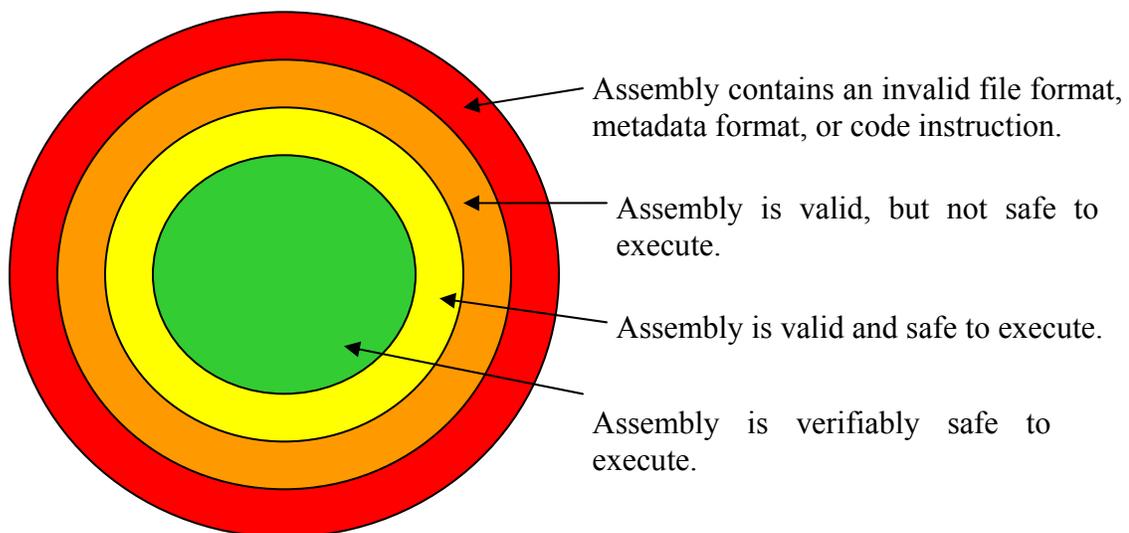


Figure 37. Relationship between validation and verification.

This task is performed automatically by the CLR when an assembly is loaded. It can also be performed on an assembly file with `peverify.exe`.

View an assembly's strong name, View the public key token corresponding to a public key

The full strong name of an assembly (name, version, culture, and public key) can be viewed using `explorer.exe` with the Assembly Cache Viewer enabled, or with `secutil.exe`. The public key token is an abbreviated form of the public key consisting of the last 8 bytes in reverse order of the SHA-1 hash of the public key data. This token can be viewed using `sn.exe`.

Strong name an assembly

Strong names enable the unique identification of shared assemblies that have been installed in the GAC. To base trust decisions, such as access to resources through CAS policy, on a strong name requires a known and verified relationship between the public key component of the strong name and a trustworthy party. The security of a strong name is based on the difficulty of determining the private key that corresponds to the public key. Although only the public key is needed to verify a strong name, the private key is necessary to create strong named assemblies. Thus, a decision to allow access to resources based on a strong name must also be based on trust that the public/private key owner has taken great care to limit access to their private key.



Recommendation: Base trust on a strong name only where the public key is verifiably associated with a trustworthy party, and the public key owner can be trusted to limit access to the corresponding private key.

To be most helpful to CAS policy configuration, a trusted software creator should use multiple key pairs that identify classes of assemblies, separate diverse product lines, or distinguish software from different business units. Through CAS policy, the administrator could then grant appropriate levels of access to resources based on the purpose of each class of assembly. For example, operating system components and administrative tools require broader access than office automation applications. Similarly, networking applications require access that embedded Web controls do not.

Because the Strong Name Membership Condition can be based on the public key and the name of the assembly only, while leaving the version variable, CAS policy can easily be configured to apply across upgrades to an assembly.

An assembly can be strong named using a public/private key pair using `sn.exe`. Using this tool, an assembly can also be re-signed with a different key pair, or re-signed to convert a delay signed assembly to a fully strong named assembly.

Verify an assembly's strong name

The integrity of an assembly can be checked by verifying its strong name. The strong name is used to digitally sign an assembly by encrypting a hash of an assembly using the private key of a public/private key pair. The corresponding public key is then stored with the

assembly and may be used to decrypt the hash value and check it against the bytes of the assembly as a test for corruptions or unauthorized modifications.

Unless the public key component of the strong name is known through some trusted means to be associated with a particular entity, this only verifies that an assembly has not been modified since it was created. It does not determine the origin or trustworthiness of the assembly's creators.

When a strong named assembly is loaded by the CLR, its strong name will be verified and it will not be permitted to execute unless it passes this check.

This task can be performed using `sn.exe`.

Enroll an assembly for strong name simulation, Withdraw an assembly from strong name simulation, List assemblies enrolled for strong name simulation

These tasks allow developers to work with delay signed assemblies as if they were strong named. An assembly is delay signed when it is not actually strong named, but a space is reserved for the digital signature component of a strong name and contains the public key representing the public/private key pair that is anticipated to be used in the future for the actual strong name. Such an assembly would not normally satisfy a strong name Membership Condition based on the public key, because it is not actually digitally signed by the corresponding private key. An entry on the strong name simulation list specifies a set of user account names and a class of assemblies that will successfully “pass” strong name verification when they are delay signed and executing in the context of one of the named user accounts. An assembly might not be fully strong named while it is being developed and undergoing frequent revision. Adding the assembly to this list allows software developers to simulate a fully strong-named environment without requiring repeated access to an organization's private keys. Anyone with access to an organization's public key can produce a delay signed assembly with that public key. When an assembly is delay signed, its origin cannot be derived from the public key alone, and any changes made to the assembly may be undetectable, so strong name verification should never be simulated in an operational environment.

These tasks can be performed using `sn.exe`.



Recommendation: Strong name verification should never be simulated in an operational environment.

View an assembly's publisher certificate

An assembly signed by a software publisher will contain an embedded Authenticode digital certificate. A digital certificate contains fields describing the name of the software publisher, the expiration date of the certificate, the cryptographic algorithms used in the digital signature process, and the name of the party that issued to certificate to the software publisher. To view the publisher's certificate as a sequence of byte values, use `secutil.exe`.

To view the certificate with its components identified, use `certmgr.exe`.

Verify the trust associated with an assembly's Authenticode digital signature

Authenticode digital signatures are the basis for the Publisher Membership Condition. An assembly will satisfy the Publisher Membership Condition for a specified software publisher, if it is digitally signed with the publisher's Authenticode software publisher's certificate, and the CLR can verify a chain of trust associated with that signature. Checking the chain of trust consists of verifying the validity of each digital signature in a chain of signatures beginning with the signature of the file itself (the lowest level), and continuing through the digital signatures of each certificate that asserts the identity of the lower-level signer. The chain is verified as trusted if all the signatures and certificates are valid, and the top-level signature is by a trusted root certification authority. Signature verification is subject to policy settings such as the Software Publishing State Keys managed by the `setreg.exe` tool, and thus trust decisions will be affected by host-specific policy. Furthermore, some verification steps require the availability of local or remote network interaction, and thus trust decisions may be affected by network conditions.

This task can be performed using `chktrust.exe`.

View an assembly's permission requests and declarative permission constraints

An assembly's permission request consists of three sets of permissions defined by the software developer: the minimum, optional, and refused sets of permissions. These sets are intended to guide administrators in determining the minimum level of privilege that code must have to perform its function. It also provides a guide to the maximum set of privileges that an assembly should have.

The Minimal Permission Set is a set of permissions required by the application in order to function adequately. If these permissions are not granted through CAS policy, the assembly will not execute. The Optional Permission Set consists of those permissions that the assembly does not need to adequately execute, but could use to provide additional features. If these permissions are not granted through CAS policy, the application should be able to fall back to a basic level of functionality. The Refused Permission Set are those permissions that the developer of the assembly has determined are not necessary for the proper functioning of any of the features of the assembly, and therefore should be denied to the assembly via CAS policy in accordance with the principle of least privilege. Note that an assembly that specifies an Optional Permission Set will have all permissions not in either its Minimal or Optional Permission Sets implicitly refused.

Permission requests are a claim made by the developer of an assembly and may or may not reflect the actual use of resources made by the assembly. The minimal set may be overstated, or it may not reflect actual access requirements added at a later stage of development. Displaying an assembly's permission requests is a check on the documented requirements of the assembly, but is not a substitute for up-to-date software documentation that fully describes the resources an assembly needs, and why it needs them. Permissions should not be granted simply because an assembly requests them or claims to require them. Granting

permissions is a trust decision, and an assembly's permission requests are only as trustworthy as the origin of the assembly.

In addition to the permission request sets, the authors of the assembly may insert programming instructions that specify dynamic security constraints such as permission requirements or explicit refusals of permissions at various points in the code in order to establish a known security environment in which portions of the code will run. These runtime environments can only be more restrictive than the permissions granted by CAS policy to the assembly itself. Thus, a dynamic permission requirement will only check that the required permission has already been granted before proceeding – it will never result in the granting of an additional permission.

Some trusted libraries are designed to access protected resources on behalf of less trusted code. Because the security permissions granted during execution of the library code are the intersection of the permission sets granted to the less trusted application and those of the trusted library, the default security environment in this case would be the restricted set of permissions granted to the less trusted code. However, trusted libraries may selectively relax these restrictions in order to mediate resource access to their callers.

There are two types of dynamic permission constraints that software developers can use: declarative constraints and imperative constraints. Declarative constraints are fully described in the assembly itself, while imperative constraints may be tailored to the current state of the program. For example, a declarative constraint could check that the File IO permission has been granted with All access to a application-specific log file. An imperative constraint could do the same thing, but could also check that the File IO permission has been granted with Read access to a file or folder previously specified on-the-fly by the user.

This task may be performed by `permview.exe`. Currently, there is no tool that displays the available static (not determined only at runtime) information about imperative permission constraints.

View the list of configured assemblies, Configure an assembly, Delete the configuration information for an assembly

A configured assembly is a versioned family of assemblies that has an associated administratively-configured Binding Policy.

Introduction to Binding Policy

The .NET Framework allows multiple versions of managed applications or libraries to be installed at the same time, so applications may request the specific version of a library that they were developed to use, even if newer versions have since become available. In many cases, a library is available from multiple sources, or an application does not request a specific version, or the version requested has been disallowed by policy or deprecated by its publisher. In this context, a facility called binding refers to the process of determining which library will satisfy an application's request. For example, when a software publisher releases a new version of a library that is backward-compatible with previous versions, requests for

older versions may be redirected to the new version through a Binding Policy created by the software publisher.

The .NET Framework provides a Binding Policy system that allows software developers and administrators to change assembly binding behavior through the use of XML configuration files. These XML files can redirect a reference from one version of an assembly to another, and indicate the location from which each version of an assembly is to be obtained. The three forms of Binding Policy are: application policy, publisher policy, and machine policy. The policy files are processed in the order of application, publisher, and machine to arrive at a final post-policy assembly reference that the CLR will use to satisfy the assembly request. The resulting output of each policy stage is used as input to the next policy stage until the final reference is obtained. A prerequisite for this process is that the assembly is strong named, as this provides precise version identification that is cryptographically bound to the assembly.

Application Policy

When an assembly is created, the versions of all libraries that it was developed to use are recorded in its manifest. When the assembly first tries to invoke one of these libraries, the CLR checks the application policy XML file to determine if the version of the requested library has been redirected to a different version. By specifying version redirection in the application policy, a developer or administrator may reconfigure which version of a strongly-named assembly should be used by the application without having to recreate the assembly with a new manifest. In addition, the application policy may specify that publisher binding policy should not be checked for any assembly references or for a specific assembly reference (when disabled for all assemblies, this is known as “safe mode”, because applications may break if the publisher’s claims of backward-compatibility of new versions are not correct). If publisher policy is checked, it could override the application policy, so this reverses the policy precedence for a specific dependent assembly.

Publisher Policy

If a publisher policy exists, the CLR examines it after first reviewing the Binding Policy in the application configuration file (and determining that the publisher policy has not been disabled). Publisher policies affect all applications using the publisher’s assembly and are used to deprecate faulty or outdated shared components by redirecting component references from the old version to a new version. Each assembly can have its own publisher policy that contains a Binding Policy in XML, and is embedded in a separate special assembly that is versioned, strong-named with the same key as the assembly, and installed in the GAC. Publisher policy assemblies not installed in the GAC will not be applied. The name of the publisher policy assembly must have the form

```
policy.<major version>.<minor version>.<assembly name>.
```

If multiple publisher policies are installed for the same assembly, the one with the highest version will take precedence.

Unless disabled, publisher policy is applied to the current version information that is in the assembly manifest or that is the result of processing the application binding policy from the application configuration file. If the application policy redirects the version specified in the assembly manifest to a new version, then the publisher policy may redirect the new version to yet a different version. If there is no application policy version redirection, then the publisher policy is applied to the version specified in the assembly manifest.

It may not always be desirable to redirect references to shared managed components to the newest available version. New version of software may be buggy or unstable, may not have been thoroughly tested, or may not be fully backward-compatible. New features introduced in the new version may introduce security vulnerabilities that did not exist in earlier versions. In these cases, application policy can use safe mode to disable automatic version redirection through a publisher policy created by a third-party.

Machine Policy

Versions of assemblies that are disallowed due to identified security flaws or other unwanted behavior may still be invoked by applications, since the application policy is under the control of the developer and publisher policy may be bypassed by using safe mode. The .NET Framework provides an administrator-controlled Binding Policy stored in the machine configuration file for each .NET Framework version that will override application and publisher policy to enforce consistent machine-wide binding behavior for any given version of any assembly.

The machine policy is applied last, cannot be bypassed, and affects all applications, so this is the appropriate place for version redirection settings driven by organizational security policy. The machine policy is a component of the machine configuration file `machine.config`, located in each .NET Framework version's `config` folder. Note that there is no Framework-wide configuration file. Each `machine.config` file will only affect the binding policy applied by the corresponding version of the CLR. In order for the administrator to create an effective machine-wide binding policy, all of the configuration files must be kept consistent.

Summary

The .NET Framework provides a method of deprecating or disallowing certain versions of an assembly. This is accomplished through three stages of policy application under the control of different users. Application policy is under the control of the developer of an application. Publisher policy is under the control of the publisher of a shared managed component. These two policies are designed to facilitate software distribution and maintenance. Machine policy is under the control of the host administrator, and is where security-relevant binding policy should be placed.

The creation of the machine Binding Policy is known as “configuring” an assembly, and can be performed using `mscorcfg.msc`.

Application configuration tasks

Add an application to be configured, Configure application properties

Configuring an application means creating application-specific Binding Policies for the assembly on which the application depends, and setting up communication channels for the use of Remoting Services. The first of these two types of tasks can be performed using `mscorcfg.msc`. The second can be performed using `mscorcfg.msc` and by creating or editing XML configuration files.

View assembly dependencies for an application, View list of assemblies configured for an application, Configure an assembly for an application, Fix an application (roll back application Binding Policy)

Application-specific Binding Policies can be created that determine what versions of an assembly will be used when the assembly is loaded in the context of the specified application. For a discussion of Binding Policies, see the View the list of configured assemblies task above. The tasks listed above can be performed using `mscorcfg.msc`.

Configure Remoting Services for an application

Configuration Remoting Services is best performed through the editing of the XML files that contain the desired settings. `mscorcfg.msc` can be used to make some modifications but in general, it is better suited for displaying some of the current Remoting Services settings for an application that it is at configuring Remoting Services. Application developers will typically create application configuration files containing Remoting Services settings that may be reviewed by the administrator using `mscorcfg.msc`. `mscorcfg.msc` will only display the contents of files named `{application file name}.config`, however, once granted the ability to actually use their application-specific Remoting Services settings, an application may load the settings from any file to which they have access.

To configure Remoting Services for an application, edit the appropriate XML file, using the discussion of Remoting Services settings in Appendix B: `mscorcfg.msc` as a general guide.

Summary

The common security-related administrative tasks for the .NET Framework are numerous and varied. Several tools have been provided, either with the .NET Framework or with the SDK to assist the administrator. The security-related background for each task is discussed in this chapter, with detailed procedures for the performance of each task with a particular tool is included Appendices A and B.

Recommendations in This Section



Recommendation: Never disable CAS policy on a computer connected to an untrusted network such as the Internet.

-  ***Recommendation: Disable trust of Test Root certificates in an operational environment.***
-  ***Recommendation: Enable checking for expired certificates.***
-  ***Recommendation: Only use the Publisher Membership Condition for software publishers with whom your organization has a well-established history of trust. Access to resources may be granted to code that presents expired certificates, so the use of the Publisher Membership Condition assumes that the publisher was trustworthy in the past as well as the present.***
-  ***Recommendation: Enable checking for revoked certificates.***
-  ***Recommendation: Disable automatic trust for certificates whose revocation status cannot be determined.***
-  ***Recommendation: Enable checking for revoked time stamp provider's certificate.***
-  ***Recommendation: Base trust on a strong name only where the public key is verifiably associated with a trustworthy party, and the public key owner can be trusted to limit access to the corresponding private key.***
-  ***Recommendation: Strong name verification should never be simulated in an operational environment.***

This page has been intentionally left blank.

Appendix

A

Administrative Tools Reference

This appendix contains descriptions of some tools used to administer the .NET Framework. Each tool's usage is described, as well as detailed procedures for performing administrative tasks. Not all tool options will be described in detail. Special attention will be given to options necessary to perform security-related tasks.

caspol.exe – .NET Framework Code Access Security Policy Tool

`caspol.exe` is distributed with the .NET Framework SDK. It is a managed console application that gives users and administrators the ability to modify .NET Framework CAS policy. For example, the administrator can configure Code Groups and Named Permission Sets, extend the CAS policy system by adding Policy Assemblies, and determine the permissions that would be granted to a specified assembly by the policy. `caspol.exe` provides much the same functionality as `mscorcfg.msc` with respect to CAS policy configuration.

Each version of `caspol.exe` is built for a specific version of the .NET Framework and is located in the corresponding .NET Framework version folder `%windir%\Microsoft.NET\Framework\. The CAS policy settings configuration through caspol.exe will apply only to the corresponding version of the .NET Framework.`

In order to function properly, `caspol.exe` must be granted the equivalent of the “Everything” Named Permission Set. The tool has a protective mechanism that prevents CAS policy changes from being made through the tool that would deny `caspol.exe` itself the necessary permissions to execute. When `caspol.exe` is asked to make such a change, it will display an error message and fail. This mechanism may be overridden using the `-force` option.

Syntax

```
caspol [options] [arguments]
```

A full list of supported options can be displayed using:

```
caspol -help
```

UNCLASSIFIED

Each option has a corresponding shortcut form (for example, the shortcut form for `-help` is `-?`). See the full list of supported options for details. `caspol.exe` can be configured to ask for confirmation whenever a command would result in a change in CAS policy. This setting is maintained in the registry, so it applies to all commands until it is changed, although it only affects a particular version of `caspol.exe`. To enable or disable confirmation for all subsequent commands, use the command

```
caspol -polchgprompt {on | off}
```

To disable confirmation only for the current command, use the option `-quiet` or `-q` in conjunction with the command.

Many `caspol.exe` tasks can be applied to all CAS policy levels or only to a single CAS policy level by preceding the option with `-all`, `-enterprise`, `-machine`, `-user`, `-customuser {file name}`, or `-customall {file name}`. The `{file name}` argument is intended to specify the path of a User level CAS policy file for a user other than the logged-on user, but it could be used to specify any CAS policy file, including an XML file not currently in use by the .NET Framework. Thus, this option could be used to administer policy for another user or to administer Enterprise or Machine level policy on a remote box via a UNC path if the .NET Framework configuration folder is shared. The `-customall` option applies any changes to the Enterprise and Machine level CAS policy files as well as to the file specified by the `{file name}` argument. When logged-on as an Administrator, the default behavior is to perform the indicated task on the Machine level policy only. Otherwise, the default is to perform the indicated task at the User level policy only.

Tasks

The following tasks can be performed using `caspol.exe`:

- Enable or disable CAS policy
- Enable or disable Execution permission checking
- Build a CAS policy cache file
- Reset all CAS policy levels to default settings
- Recover the previous settings for a CAS policy level
- View Code Groups
- Add or remove a Code Group
- Rename a Code Group
- Set or clear the Exclusive or Level Final attribute of a Code Group

- Change a Code Group's Membership Condition
- Change a Code Group's associated Named Permission Set
- View Named Permission Sets
- Add or remove a Named Permission Set
- Modify a Named Permission Set
- View Policy Assemblies
- Enroll or withdraw a Policy Assembly
- List Code Groups to which an assembly belongs
- View an assembly's Allowed Permission Set
- Create a tailored Code Group

Enable or disable CAS policy

CAS policy enforcement may be disabled for a host computer. Disabling CAS policy will prevent checks for any managed code for all users. This setting is stored in the registry and will apply to all versions of the .NET Framework. To enable or disable CAS policy enforcement, use the command:

```
caspol -security {on | off}
```

Enable or disable Execution permission checking

When asked to load an assembly, the CLR will normally check to see whether the assembly has the Enable Assembly Execution permission. If not, it will not load the assembly. This is a performance-enhancing feature that can be enabled or disabled by the command:

```
caspol -execution {on | off}
```

This setting is stored in the registry and will apply to all versions of the .NET Framework.

Build a CAS policy cache file

The command

```
caspol -buildcache
```

will create new CAS policy cache files (.cch) for policy levels modified since the last time the cache files were built. The policy cache files are used to enhance the performance of the CAS policy permission resolution process, and their use is not documented.

Reset all CAS policy levels to default settings

Each CAS policy level has a built-in default state that is used whenever the policy configuration files are missing. One way to revert to the default state is to simply delete the relevant configuration file. The next time the .NET Framework is invoked by any managed code, a new configuration file will be created containing the default policy. `caspol.exe` can be used to immediately reset CAS policy to its default state. For example,

```
caspol -enterprise -reset
```

resets the Enterprise level CAS policy for a particular .NET Framework version to its default state.

Recover the previous settings for a CAS policy level

When a change is made to CAS policy through `mscorcfg.msc` or `caspol.exe`, the previous CAS policy file for the level that was changed is saved in a backup file with the extension `.old`. CAS policy can be rolled back to this prior policy, if it exists. This will undo exactly one change, as each change made by `mscorcfg.msc` or `caspol.exe` is backed up immediately, overwriting the previous backup file. For example,

```
caspol -machine -recover
```

resets the Machine level CAS policy to the contents of the backup file. This will also save the rolled-back policy as the new “old” policy.

View Code Groups

The commands

```
caspol -list
```

```
caspol -listgroups
```

```
caspol -listdescription
```

display Code Group information for the Machine level CAS policy. The first command shown also displays all the Named Permission Sets and Policy Assemblies for the given CAS policy level.

The second command shown above lists the Code Groups by a hierarchical numeric label, Membership Condition, and associated Named Permission Set. The numeric label identifies nodes in the Code Group tree. For example, the root Code Group is always `All_Code` and has the numeric label 1. Child Code Groups of `All_Code` have the numeric labels 1.1, 1.2, etc. When making changes to a Code Group, the targeted Code Group can be identified by either its name or its numeric label. If more than one Code Group has the same name, the

results may be unpredictable. For consistent results, the numeric labels may be used to uniquely identify a Code Group.

The third command lists Code Groups by numeric label, name, and description.

Add or remove a Code Group

The command

```
caspol -addgroup {parent Code Group identifier} {Membership
Condition} {Named Permission Set} [-exclusive {on | off}] [-
levelfinal {on | off}] [-name {"name"}] [-description
{"description"}]
```

will add a Code Group with the specified properties. The parent Code Group, a Membership Condition, and an associated Named Permission Set must be specified. Optionally, the new Code Group may be marked as Exclusive or Level Final, and a name and description provided. The name and description must be in double quotes. Unlike `mscorcfg.msc`, `caspol.exe` will allow the addition of multiple Code Groups with the same name (or no name). However, the results can be unpredictable when attempting to remove a Code Group whose name occurs multiple times. For stability, Code Group names should be unique across the entire Code Group tree for any given CAS policy level.



Recommendation: Make Code Group names unique across the entire Code Group tree for any given CAS policy level.

The form of the Membership Condition argument varies depending on the Membership Condition specified (Table 18):

Membership Condition	Argument forms
All Code	-all
Zone	-zone {MyComputer Intranet Internet Trusted Untrusted}
Site	-site {domain name of Web site}
Publisher	-pub -cert {certificate file} -pub -file {signed file containing an embedded certificate} -pub -hex {hexadecimal representation of an X.509 certificate}
Strong Name	-strong -file {strong-named assembly file} {{name} -noname} {{version} -noversion}
Hash	-hash {MD5 SHA1} -hex {hash value} -hash {MD5 SHA1} -file {assembly file to hash}
Application Directory	-appdir
Custom	-custom {XML file}

Table 18. `caspol.exe` Membership Condition Arguments.

For example, the command:

```
caspol -machine -addgroup All_Code -zone Internet - MyPermissions -  
name NewCodeGroup
```

adds a child Code Group to the All_Code Code Group at the Machine level. The new Code Group will be named “NewCodeGroup,” and will have the Internet Zone Membership Condition and the Named Permission Set “MyPermissions.”

The command

```
caspol -remgroup {Code Group identifier}
```

will remove the specified Code Group and all of its child Code Groups. For example, the command

```
caspol -enterprise -remgroup TestCodeGroup
```

will remove the named group from the Enterprise level CAS policy. If more than one Code Group exists with the specified name, the results may be unpredictable. To guarantee consistency when multiple Code Groups exist with the same name, use the numeric labels to uniquely identify Code Groups within the Code Group tree.

Rename a Code Group

The command

```
caspol -chggroup MyCodeGroup -name YourCodeGroup
```

will change the name of the Code Group from MyCodeGroup to YourCodeGroup. A Code Group can be renamed in conjunction with any other property change. See the Change a Code Group’s associated Named Permission Set task below for an example.

The description of a Code Group can also be changed using

```
caspol -chggroup MyCodeGroup -description "What-a-group!"
```

Set or clear the Exclusive or Level Final attribute of a Code Group

The commands

```
caspol -chggroup MyCodeGroup -exclusive on
```

```
caspol -enterprise -chggroup 1.2.4 -levelfinal off
```

will set the Code Group MyCodeGroup at the default CAS policy level to be Exclusive and remove the Level Final attribute from Code Group 1.2.4 at the Enterprise level policy.

Change a Code Group's Membership Condition

A Code Group's Membership Condition can be changed using the command

```
caspol -chggroup {Code Group identifier} {membership ↵  
condition}
```

The form of the Membership Condition arguments depends on the Membership Condition specified. See the task Add or remove a Code Group above for details. For example, the command

```
caspol -chggroup MyGroup -hash SHA1 ↵  
-hex 31f63c465a058c0684f7cd8877eb85c8093dea7a6
```

will change the Membership Condition of the named Code Group at the Machine level policy to a Hash Membership Condition with the specified SHA-1 hash value. The command

```
caspol -user -chggroup TheLocalNetwork -zone Intranet
```

will set a Zone Membership Condition for the Local Intranet. A Code Group's Membership Condition can be changed in conjunction with any other property change. See the Change a Code Group's associated Named Permission Set task below for an example.

Change a Code Group's associated Named Permission Set

The command

```
caspol -user -chggroup MyCodeGroup FullTrust
```

will change the associated Named Permission Set associated with the Code Group MyCodeGroup in the User level CAS policy to FullTrust. A Code Group's associated Named Permission Set can be changed in conjunction with other Code Group property changes. For example, the command

```
caspol -enterprise -chggroup MyCodeGroup -hash SHA1 ↵  
-hex 31f63c465a058c0684f7cd8877eb85c8093dea7a6 ↵  
OfficeApps -name "YourCodeGroup"
```

applies to the Code Group MyCodeGroup in the Enterprise level CAS policy. It will change the Membership Condition to the specified Hash Membership Condition, change the associated Named Permission Set to OfficeApps, and rename the Code Group to "YourCodeGroup."

View Named Permission Sets

The command

```
caspol -listpset
```

will display the Named Permission Sets defined for a CAS policy level.

Add or remove a Named Permission Set

The command

```
caspol -machine -addpset "c:\perms\webaccess.xml" WebAccessPermissions
```

will add the Named Permission Set WebAccessPermissions to the Machine level CAS policy. The permissions contained within WebAccessPermissions are defined in the XML file c:\perms\webaccess.xml. The XML file must contain a <PermissionSet> element as the root. If no name is supplied for the new Named Permission Set, the Name attribute of the <PermissionSet> element will be used as the default name of the Named Permission Set. If the Name attribute is not present, a name must be supplied on the command line.

The command

```
caspol -enterprise -rempset WebAccessPermissions
```

will remove the given Named Permission Set from the Enterprise level CAS policy. If a Named Permission Set is currently associated with a Code Group, caspol.exe will not remove it. caspol.exe will also not remove any of the built-in Named Permission Sets that are part of the default CAS policy.

Modify a Named Permission Set

The permissions associated with a Named Permission Set can be modified by removing the Named Permission Set with the -rempset option and adding it again with a different XML source file with the -addpset option. If the Named Permission Set is currently associated with a Code Group, caspol.exe will not allow it to be removed. However, caspol.exe will allow a Named Permission Set to be redefined from an XML file, while retaining all the associations between the Named Permission Set and Code Groups. For example,

```
caspol -user -chgpset newdbperms.xml DatabasePermissions
```

will change the permissions contained in the Named Permission Set DatabasePermissions to those listed in the XML file newdbperms.xml, without modifying any associations between the Named Permission Set and Code Groups. The built-in Named Permission Sets that are

part of the default CAS policy cannot be changed using `caspol.exe` except for the Everything Named Permission Set.

View Policy Assemblies

The command

```
caspol -listfulltrust
```

displays the Policy Assemblies for the default CAS policy level.

Enroll or withdraw a Policy Assembly

The command

```
caspol -machine -addfulltrust NewMembershipCondition.dll
```

will enroll the assembly `NewMembershipCondition.dll` in the Machine level CAS policy configuration file for the current .NET Framework version. Once enrolled, custom CAS policy components (in this case, a custom Membership Condition) defined in the assembly can be referenced by the policy elements of the configuration file. The command

```
caspol -enterprise -remfulltrust CustomPermission.dll
```

will remove the assembly `CustomPermission.dll` from the list of Policy Assemblies configured for the Enterprise policy level.

List Code Groups to which an assembly belongs

`caspol.exe` can determine the Code Groups in one or all policy levels for which a given assembly satisfies the associated Membership Conditions. Unlike other commands, the default scope is all three policy levels of Enterprise, Machine, and User. Use `-enterprise`, `-machine`, `-user`, or `-customuser {file name}` to limit the scope to one policy level. For example,

```
caspol -machine -resolvegroup c:\managedcode\myapp.exe
```

will display the Code Groups at the Machine policy level that contain the specified assembly. Assemblies may be specified by UNC or local file system paths, but URLs with protocols other than “file:///” are not supported. To display Code Group membership for assemblies specified by URLs with other protocols, use `mscorcfg.msc`.

If the specified assembly satisfies the Membership Condition of more than one Code Group marked Exclusive at any policy level, the task will fail. On the other hand, if the specified assembly belongs to an Exclusive Code Group as well as other Code Groups, those other Code Groups are listed along with the Exclusive Code Group, even though they would not contribute permissions to the assembly’s Allowed Permission Set (although they could still

affect access if they were marked with the Level Final attribute). If the specified assembly satisfies the Membership Condition of a Code Group marked Level Final, then lower policy levels will not be evaluated.

View an assembly's Allowed Permission Set

`caspol.exe` can compute the Allowed Permission Set that the current CAS policy will associate with an assembly. This set may be different from the set of permissions actually granted to the assembly when it is executing (its Granted Permission Set). The Allowed Permission Set will take into account Exclusive and Level Final Code Groups, but will not include any Application Domain policy, as this policy is determined at runtime. The Allowed Permission Set also does not include permission requests (Minimum, Optional, or Refused permissions) made by the assembly itself. Furthermore, the Allowed Permission Set is based on the evidence presented by the specified assembly, and the actual evidence presented by the assembly at runtime may be different if the assembly is obtained from a different source. The default scope is the three policy levels of Enterprise, Machine, and User. To limit the scope to a single level, use the options `-enterprise`, `-machine`, `-user`, or `-customuser {file name}`. For example, the command

```
caspol -resolveperm mylibrary.dll
```

will determine the Allowed Permission Set associated with the assembly `mylibrary.dll` by the current CAS policy.

If the specified assembly satisfies the Membership Condition of more than one Code Group marked Exclusive at any policy level, the task will fail.

Create a tailored Code Group

To create a Code Group that will associate a particular set of permissions to a particular assembly, perform the following steps:

- Create a Named Permission Set that contains the desired permissions. See the Add or remove a Named Permission Set task above for details.
- Make a Code Group whose Membership Condition will discriminate between the targeted assembly and other assemblies. Set the Membership Condition to discriminate as little as possible commensurate with operational requirements and organizational policy. At the same time, insist on cryptography-based identities unless prohibited by policy constraints. Associate the new Named Permission Set with this Code Group. See the Add or remove a Code Group task above for details.

certmgr.exe – Microsoft Certificate Manager Tool

`certmgr.exe` is distributed with the .NET Framework SDK. It is both a console and a GUI application that manages certificates stored in files or system stores.

Syntax

```
certmgr [{action}] [{options}] {items} [{destination}]
```

The above is an abbreviated form of the full syntax specification. The action can be to add the specified items to the destination store or file (-add), to delete the items (-del), to copy the items from a store to a destination file (-put), or to display the items (no action parameter). Items can be certificates, certificate revocation lists (CRLs), or certificate trust lists (CTLs) from a specified store or file. A full list of supported options can be displayed using

```
certmgr -?
```

To launch a GUI application that performs some certificate management tasks, use

```
certmgr
```

with no parameters. Note that the single `certmgr.exe` task included in this guide, namely, viewing a publisher's certificate embedded in a digitally signed assembly, cannot be performed using the GUI interface. The command line syntax to perform this task is shown below.

Tasks

The following tasks can be performed using `certmgr.exe`:

- View an assembly's publisher certificate

View an assembly's publisher certificate

The commands

```
certmgr app.exe
```

```
certmgr -v app.exe
```

will display the Authenticode certificate for the publisher who digitally signed the assembly `app.exe`, or an error message if the assembly file does not contain a digital certificate. The second command will display a more verbose output message.

chktrust.exe – Microsoft Authenticode Signature Verification Tool

`chktrust.exe` is distributed with the .NET Framework SDK. It is a console application that determines the trust associated with an Authenticode digital signature. Trust verification

of digital signatures is dependent on the values of the Software Publishing State Keys that are managed by the `setreg.exe` tool. See `setreg.exe` below for details.

Syntax

```
chktrust [-q] [-v] {assembly file}
```

Tasks

The following tasks can be performed using `chktrust.exe`:

- Verify the trust associated with an assembly's Authenticode digital signature

Verify the trust associated with an assembly's Authenticode digital signature

The command

```
chktrust app.exe
```

will verify the chain of trust associated with a software publisher's Authenticode digital signature of `app.exe`. Checking the chain of trust consists of verifying the validity of each digital signature in a chain of signatures beginning with the signature of the file itself (the lowest level), and continuing through the digital signatures of each certificate that asserts the identity of the lower-level signer. The chain is verified as trusted if all the signatures are valid, and the top-level signature is by a trusted root certification authority. Unless the `-q` option is used, `chktrust.exe` reports its results in a dialog box.

```
chktrust -q app.exe
```

The command shown above will not pop up a dialog box that allows the user to declare trust in the assembly signer. Instead, a single-line output will report success or failure. If the assembly is not Authenticode signed, or the signature cannot be verified as trusted, `chktrust.exe` will report failure. If the assembly is signed and the signature is verified as trusted, `chktrust.exe` will report success.

explorer.exe/shfusion.dll – Windows Explorer/ Assembly Cache Viewer

The Assembly Cache Viewer (`shfusion.dll`) is a Windows shell extension that is installed and enabled when the .NET Framework is installed. It presents a special view of the GAC and Zap Cache (`%WINDIR%\assembly`) in Windows Explorer (`explorer.exe`) that combines all versions of assemblies installed in the GAC in a single list, even though they reside in different subfolders of the underlying file system tree.

To enable or disable the Assembly Cache Viewer, a registry value must be modified. See the task Enable or disable the Assembly Cache Viewer under `regedit.exe` below for details.

Tasks

The following tasks can be performed using `explorer.exe`:

- View cache contents
- View or modify GAC properties
- Add an assembly to the GAC
- Delete an assembly from the GAC
- View assembly properties
- Reset all CAS policy levels to default settings

View cache contents

When the Assembly Cache Viewer is enabled, the GAC and Zap Cache are shown as a virtual folder with all installed assemblies in a combined list. To view the contents of the GAC and Zap Cache, simply navigate in Windows Explorer to `%WINDIR%\assembly`. If the Assembly Cache Viewer is disabled, the underlying file system hierarchy that implements the GAC will be displayed. Since the “raw” file system details are subject to change, this display is of limited value to the administrator.

Although the GAC and Zap Caches are implemented by a file system tree in which each version of an assembly is stored in a different folder, the Assembly Cache Viewer presents a combined view of the GAC and the Zap Cache as a virtual folder with all the assemblies in a single list. To view the contents of the GAC and the Zap Cache, navigate in Windows Explorer to the directory `%windir%\assembly`. Figure 38 shows the presentation of the GAC and Zap Cache as a virtual folder.

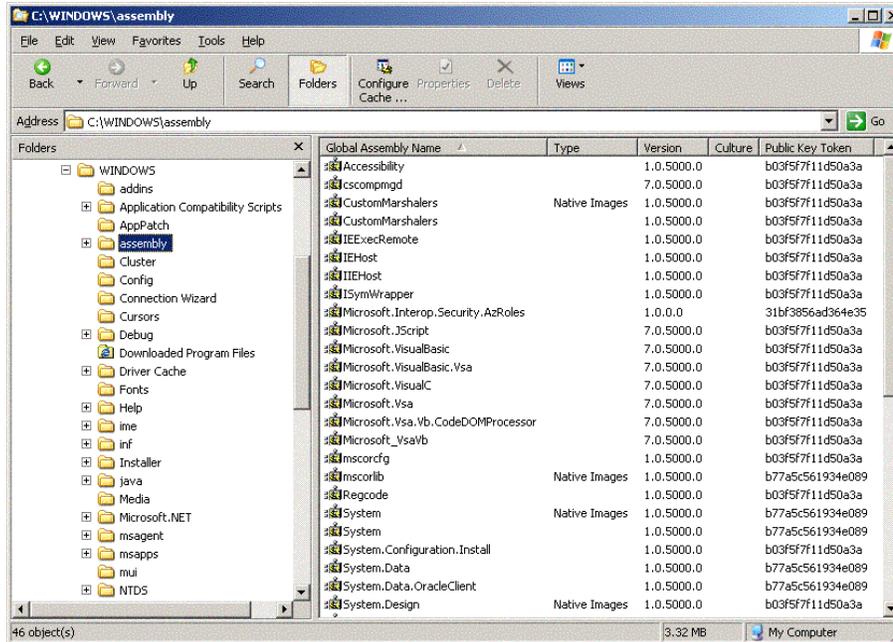


Figure 38. Viewing the GAC and Zap Cache in Windows Explorer using the Assembly Cache Viewer.

Add an assembly to the GAC

When the Assembly Cache Viewer is enabled, an assembly may be added to the GAC through Windows Explorer by dragging its icon or by copying and pasting the assembly file into the GAC folder %WINDIR%\assembly.

An assembly must be strong-named before it can be added to the GAC. The strong name of an assembly provides a unique identity to a particular version of an assembly that cannot be confused with other versions (it is cryptographically tied to the actual sequence of bytes of the file). This allows the installation of multiple versions of an assembly, all of which have the same file name. Managed code that uses an assembly in the GAC may request the assembly by its full strong name, thereby ensuring that the referenced code is the exact version expected.

The GAC is a repository for managed libraries that may be shared by multiple applications. Once installed in the GAC, an assembly may be used by any managed application. Assemblies in the GAC are not necessarily granted unrestricted permissions. The strong name requirement serves to resolve naming ambiguities, and ensure that an assembly has not been corrupted or modified during distribution. The trust-worthiness of an assembly and its access to resources through CAS policy is a function of the trust assigned to the parties and processes involved in its creation and any organizational security policy. Once the degree of trust has been assessed, the strong name provides a means of ensuring that the code employed is actually the same code that was assessed.

Delete an assembly from the GAC

When the Assembly Cache Viewer is enabled, an assembly may be deleted from the GAC through Windows Explorer by right-clicking the assembly entry in the cache folder (see the View cache contents task above) and choosing **Delete** from the context menu. Alternatively, select the assembly entry and select **File, Delete** from the drop-down menus.

View properties of an assembly installed in the GAC

When the Assembly Cache Viewer is enabled, properties of an assembly may be displayed through Windows Explorer by right-clicking the assembly entry in the cache folder (see the View cache contents task above) and choosing **Properties** from the context menu. Alternatively, select the assembly entry and select **File, Properties** from the drop-down menus. This will display the Properties dialog box for the assembly (see Figure 39).

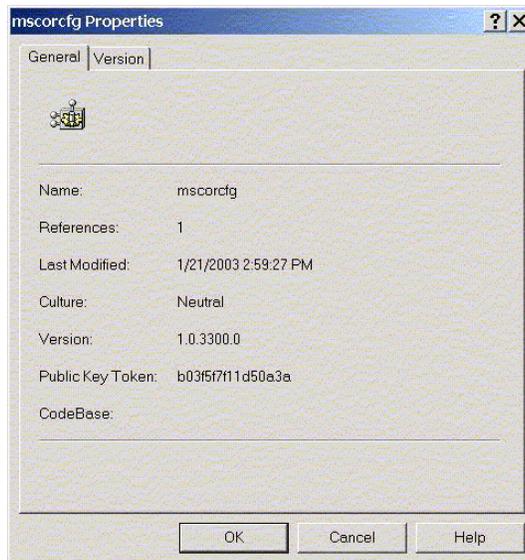


Figure 39. Assembly Properties Dialog Box.

Table 19 describes the assembly properties viewable through the Assembly Properties dialog box. The strong name of an assembly consists of the four properties: assembly name, version, culture, and public key token.

Property	Description
Assembly name	Part of the assembly's strong name.
Number of references	Number of references associated with this assembly. References record dependency information. See <code>gacutil.exe</code> for more information.
Last modified date	The date the assembly's file was last modified.
Culture	Part of the assembly's strong name.
Version	Part of the assembly's strong name.

Property	Description
Public key token	Part of the assembly's strong name. The public key token is a cryptographically-derived short form of the public key information.
CodeBase	The full path of the assembly's main file (the file containing the assembly manifest).
File version	Version information about the PE file that contains the assembly.
Description	Software publisher-supplied data.
Copyright	Software publisher-supplied data.
Comments	Software publisher-supplied data.
Company name	Name of the software publisher.
Internal name	Software publisher-supplied data.
Language	Language identification.
Legal trademarks	Software publisher-supplied data.
Original filename	The name of the file corresponding to this assembly.
Product name	The name of the software product that installed this assembly.
Product version	The version of the software product that installed this assembly.

Table 19. Assembly Properties Viewable Through the Assembly Cache Viewer.

View or modify GAC properties

View the GAC using Windows Explorer with the Assembly Cache Viewer enabled (see the View cache contents task above). Click the **Configure Cache Settings** button on the toolbar or select **Tools, Cache Settings...** from the menus. This will open the Cache Properties dialog box (see Figure 40). The only GAC property modifiable through the Assembly Cache Viewer is the size of the cache for downloaded assemblies (labeled **Store limits**).

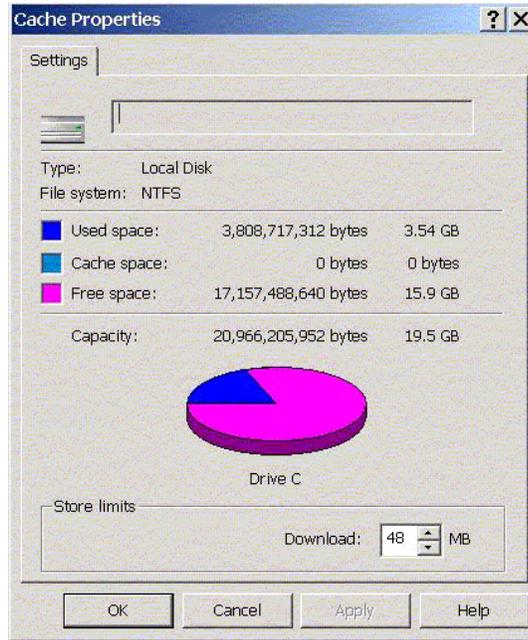


Figure 40. Cache Properties Dialog Box.

The default size of the download cache is 4608KB, and although the GUI allows the value to be changed in units of MB, it is configurable in units of KB through the registry key HKLM/Software/Microsoft/Fusion/DownloadCacheQuotaInKB.

Reset all CAS policy levels to default settings

To reset CAS policy for a given level, simply delete the XML file that contains the CAS policy settings. The next time the CLR is started by any process, a new policy file will be created containing the default settings. Table 20 shows the XML files that correspond to the three configurable policy levels. There is no default CAS policy for the Application Domain policy level. Each version of the .NET Framework has its own CAS policy configuration files. To specify a particular version of the .NET Framework, replace “{version}” with the desired version data (for example, “1.1.4322”) in the paths shown.

CAS Policy Level	XML File
Enterprise	%WINDIR%\Microsoft.NET\Framework\v{version}\config\enterprisesec.config
Machine	%WINDIR%\Microsoft.NET\Framework\v{version}\config\security.config
User	%USERPROFILE%\Application Data\Microsoft\CLR Security Config\v{version}\security.config

Table 20. XML Files for .NET Framework CAS Policy.

gacutil.exe – .NET Global Assembly Cache Utility

The Global Assembly Cache Tool is a console application distributed with the .NET Framework SDK that allows users and administrators to view and manipulate the contents of the three .NET Framework caches, the Global Assembly Cache (GAC), the Zap Cache, and the Download Cache. The GAC holds assemblies installed on the local host that are available for shared use by other code. The Zap Cache is a sub-cache of the GAC that holds assemblies that have been precompiled into native machine code. Zap Cache assemblies are typically Fully Trusted libraries that are used frequently by the CLR itself, so precompilation boosts performance. The `ngen.exe` tool can create a native image from an assembly and install it into the Zap Cache. For more information on this tool, see the .NET Framework SDK. The Download Cache holds assemblies downloaded from remote sites. When an assembly is invoked via a URL, a copy is downloaded and temporarily stored in the Download Cache while it is executing. Although it resides temporarily on the local machine in this cache, it is granted permissions based on the URL from which it was obtained.

Syntax

```
gacutil [/silent | /nologo] {option} [{parameters}]
```

A full list of supported options and their parameters can be displayed using:

```
gacutil
```

Tasks

The following tasks can be performed using `gacutil.exe`:

- View cache contents
- Add an assembly to the GAC
- Delete an assembly from the GAC
- Clear the Download Cache

View cache contents

The commands

```
gacutil /l [{assembly name without file extension}]
```

```
gacutil /lr [{assembly name without file extension}]
```

```
gacutil /ldl
```

list the contents of one or more caches in various forms. The `/l` argument will display all of the GAC assemblies and Zap Cache executables, or optionally only those with the specified name. Zap Cache executables will be listed under “The cache of ngen files”. The `/lr` argument will also include reference information in the list. GAC references are used to keep track of assembly dependencies, so that assemblies are not uninstalled while they are still needed for use by some application. The `/ldl` argument lists the contents of the Download Cache.

Add an assembly to the GAC

The commands

```
gacutil /i myassembly.dll
```

```
gacutil /if myassembly.dll
```

```
gacutil /ir myassembly.dll {reference scheme} {reference ID} ↵
    {reference description}
```

install the strong named assembly contained in the file `myassembly.dll` into the GAC. For multi-file assemblies, the specified file must be the one that contains the assembly manifest. `gacutil.exe` requires the assembly name to be the same as the name of the file (without the file extension). In order for the GAC to make multiple versions of the same assembly available at the same time, each assembly installed in the GAC must be strong-named. This provides a unique extended name that is cryptographically tied to the exact contents of the file and allows assemblies with the same file name to be distinguished. Since the Windows file system does not recognize strong names, the GAC is currently implemented by storing similarly named files in different folders. These files are presented in a merged list through the Assembly Cache Viewer and programming interfaces as if they were in the same folder.

If the same version of the assembly already exists in the GAC, the installation will not overwrite it unless the `/if` argument is used to force the installation.

The `/ir` option installs the assembly and adds the specified reference information to the references list in the registry under `HKLM\Software\Microsoft\Fusion\References`. If the assembly is already installed in the GAC, it simply adds an additional reference. References record dependency information by pointing to other assemblies that need to use the referencing assembly: if assembly `app.exe` depends on assembly `lib.dll`, then `lib.dll` should be installed with a reference to `app.exe`. This is useful when installing a managed application that depends on other shared assemblies. The shared assemblies can be installed with a reference back to the managed application. Table 21 provides a description of the reference data values.

Scheme	ID	Description
FILEPATH	The full path to the assembly	Any text

	that depends on the installed assembly	
UNINSTALL_KEY	Add/Remove Programs token	Any text
OPAQUE	Any text	Any text
WINDOWS_INSTALLER	MSI	Windows Installer

Table 21. `gacutil.exe` Reference Data Values.

The reference scheme must be one of the values shown. The meaning of the reference ID value depends on the specified scheme. A `FILEPATH` reference can be used to create references manually or through scripting. The `FILEPATH` ID value should be the full path to the referenced assembly. If a partial path is given, then `gacutil.exe` constructs a full path as follows:

- A value that does not begin with “\” will be appended to the current directory path. For example, “`dir\x`” may become “`c:\what\example\dir\x`”.
- A value that begins with “\” will be appended to the current volume. For example, “`\dir\x`” may become “`c:\dir\x`”.
- A value that begins with a volume letter will not be changed.

An `UNINSTALL_KEY` ID value should be the name of a registry key under `HKLM\Software\Microsoft\Windows\CurrentVersion\Uninstall`. This key holds information about software or data installed through the Add/Remove Programs Control Panel extension. This type of reference indicates that this assembly should be removed through a software uninstall via the Add/Remove Programs Control Panel extension.

`OPAQUE` ID and description values can be any text. These references are not used by the .NET Framework, but provide a means of storing application-dependent information. Because the reference records are intended to reflect assembly dependencies, and the behavior of `gacutil.exe` is based on this expectation, `OPAQUE` reference records should not be used to store other types of metadata.

The `WINDOWS_INSTALLER` scheme is used by the .NET Framework installation process to denote assemblies that were added to the GAC when the .NET Framework was installed. These assemblies should only be removed from the GAC by the .NET Framework’s uninstall process.

Although the reference description is a required parameter, it can usually take any value. Reference information cannot differ only by the description field. If a second reference is specified that matches an existing reference in the scheme and ID values, the description for the existing reference will be overwritten with the new value.

Installation of a managed application should include adding shared managed libraries to the GAC with references. This is especially true if the shared libraries may have been added to the GAC by a previous application install. In such a case, an additional reference record will

be added for the existing library, protecting the new application should the previous application be uninstalled. When an application is uninstalled, the uninstall of its dependent libraries should provide the reference data that was specified at install-time. The reference record will be deleted, but the library will only be removed from the GAC if there are no more “active” reference records. See the Delete an assembly from the GAC task below for more details.

Delete an assembly from the GAC

The commands

```
gacutil /u myassembly,Version=1.2.0.0,Culture=neutral,↓
    PublicKeyToken=1a2b3c4d5e6f7089,Custom=null

gacutil /ur myassembly,Version=2.0.0.0 FILEPATH ↓
    "c:\apps\myapp.exe" "myapp"

gacutil /uf myassembly
```

attempt to remove the assembly `myassembly` from the GAC. Removal of an assembly from the GAC is affected by the dependency information recorded in its reference records. References that contain valid data called “active.” In general, a `FILEPATH` reference is active if the `ID` parameter is the path of an existing file. An `UNINSTALL_KEY` reference is active if the `ID` is the name of an existing subkey of `HKLM\Software\Microsoft\Windows\CurrentVersion\Uninstall`. `OPAQUE` and `WINDOWS_INSTALLER` references are always considered active.

Active references protect an application from being broken by the removal of dependent libraries. When an assembly has an active reference record, an attempt to remove it from the GAC will fail (a “force” option can override this). Since assemblies may be uninstalled without deleting corresponding reference records, these records may become inactive. Whenever an uninstall is attempted, reference records are cleaned up by the deletion of inactive references.

The first command shown above provides a full specification of an assembly strong name. If the specified assembly has been installed in the GAC and has no active references, it will be removed. If there are active references, it will not be removed, and the active references will be reported (unless `/silent` is used).

The second command shown above provides only a partial assembly name. Since multiple versions of an assembly or assemblies signed with different strong names may coexist in the GAC, the combination of the simple name `myassembly` and the version number `2.0.0.0` could match more than one assembly installed in the GAC. For example, several different software vendors could distribute helper assemblies named `Util`. Since each version will be strong named with the vendor’s unique private key, they can coexist in the GAC. When partial assembly name information is provided, the command will apply to all assemblies

matching the specified information. As with adding an assembly to the GAC, the reference description value (in this case “myapp”) is required, but not used. Any reference that matches the scheme and ID values will be removed even if the description does not match the specified value.

The `/ur` option attempts to remove the indicated reference. If this is successful and there are no more active references, the assembly is removed from the GAC as well. `WINDOWS_INSTALLER` references cannot be removed by `gacutil.exe`. Assemblies with this type of reference will be removed during an uninstall of the .NET Framework version with which they were distributed.

The third command shown above also provides only a partial assembly name. The command shown will attempt to remove all assemblies that have the name `myassembly`. The `/uf` option (force) attempts to remove even those with active references. This will fail for assemblies with `WINDOWS_INSTALLER` references, as these references cannot be removed.

The command

```
gacutil /ungen myassembly
```

removes any native images for assemblies with the name `myassembly` from the Zap Cache. The `ngen.exe` tool that comes with each .NET Framework version must be used to add assemblies to the Zap Cache. See the .NET Framework SDK for more information on this tool.

Example

```
gacutil /i WebCommerce.exe
```

```
gacutil /ir ShoppingCart.dll FILEPATH ␣  
    "c:\www.example.com\WebCommerce.exe" WebCommerce
```

```
gacutil /ir Calculator.dll FILEPATH ␣  
    "c:\www.example.com\WebCommerce.exe" WebCommerce
```

```
gacutil /i BudgetMaker.exe
```

```
gacutil /ir Calculator.dll FILEPATH ␣  
    "c:\www.example.com\BudgetMaker.exe" BudgetMaker
```

A managed application `WebCommerce.exe` depends on `ShoppingCart.dll` and `Calculator.dll`. After `WebCommerce.exe` is installed in the GAC, `ShoppingCart.dll` and `Calculator.dll` are also installed with references to `WebCommerce.exe`. Another managed application `BudgetMaker.exe` also depends on `Calculator.dll`. After `BudgetMaker.exe` is installed in the GAC, `Calculator.dll`

is installed with a reference to BudgetMaker.exe. Since Calculator.dll is already installed in the GAC, a second reference is recorded for this library.

The following commands will uninstall the application WebCommerce.exe:

```
gacutil /ur Calculator.dll FILEPATH ↓
    "c:\www.example.com\WebCommerce.exe" WebCommerce

gacutil /ur ShoppingCart.dll FILEPATH ↓
    "c:\www.example.com\WebCommerce.exe" WebCommerce

gacutil /u WebCommerce.exe
```

The first command will remove the reference to WebCommerce.exe from Calculator.dll. Since this is not the only reference (there is also the reference to BudgetMaker.exe), Calculator.dll will not be removed from the GAC. Next, the reference will be removed from ShoppingCart.dll. Since this was the only reference, ShoppingCart.dll will then be removed from the GAC. Lastly, the application WebCommerce.exe will be removed from the GAC.

By keeping track of references that record assembly dependencies, the GAC supports the use of shared assemblies in a more robust way. Assemblies do not have to be aware of the dependency requirements of other assemblies.

Clear the Download Cache

The command

```
gacutil /cdl
```

will remove all the assemblies from the Download Cache.

migpol.exe – CAS Policy Migration Tool

migpol.exe is a console application available in .NET Framework version 1.1. This tool allows the administrator to migrate Enterprise and Machine level CAS policy from one version of the .NET Framework to another.

Syntax

```
migpol {-migrate {toVersion} [{fromVersion}] | -listversions}
```

A full list of supported options can be displayed using:

```
migpol
```

Tasks

The following tasks can be performed using migpol.exe:

- List the .NET Framework versions installed
- Migrate CAS policy from one .NET Framework version to another

List the .NET Framework versions installed

The command

```
migpol -listversions
```

displays the versions of the .NET Framework installed on the current host.

Migrate CAS policy from one .NET Framework version to another

The commands

```
migpol -migrate 1.0.3705 1.1.4322
```

```
migpol -migrate 1.1.4322 1.0.3705
```

```
migpol -migrate 1.1.4322
```

attempt to migrate Enterprise and Machine CAS policy from one .NET Framework version to another. The first command shown above attempts to migrate policy from the newer version (1.1.4322) to the older version (1.0.3705), while the second command attempts to migrate CAS policy from the older version to the newer version. The third command assumes that only two versions of the .NET Framework have been installed. The CAS policy of the specified version will be updated to match the policy of the unspecified version. If more than two versions of the .NET Framework are installed, the source version must be specified or migpol.exe will fail and report an error message. Because migpol.exe will overwrite the existing CAS policy files, both the source and the target policies should be backed up before running migpol.exe.



Recommendation: Back up both the source and the target CAS policies at the Enterprise and Machine levels before running migpol.exe.

The CAS policy system can be extended by the development of custom Code Group, Membership Condition, or Permission types. This is not the same as defining a new Code Group or Named Permission Set, but involves the use of custom software. For example, an assembly could be developed that defines a new type of Code Group beyond the built-in Union, First Match, File, and Net Code Group types. The assemblies that define these types must be strong-named and recorded as policy extension assemblies (see the Add a policy assembly task under mscorcfg.msc or caspol.exe for details). Extending the CAS

policy system may prevent policy migration through `migppol.exe`. If either the Enterprise or Machine level policy of either the source .NET Framework version or the target .NET Framework version contains a custom Code Group type or a standard Code Group with a custom Membership Condition, `migppol.exe` will not migrate the policy (although it will still report “success”). In contrast, if the only custom CAS policy extensions are custom Permission types, they will be simply discarded from the migrated policy. Named Permission Sets that contain the custom Permission types will be migrated without the custom types.

After running `migppol.exe`, check the resulting migrated policy to ensure that it conforms to any organizational security policy and that the migrated policy represents an appropriate configuration for the target version of the .NET Framework. Do not assume that an organizational policy-conforming CAS policy migrated via `migppol.exe` will remain conforming after migration. `migppol.exe` is an aid to maintaining policy consistency across versions of the .NET Framework that should be used in conjunction with administrative review.



Recommendation: Review any policy migrated using `migppol.exe` to ensure that it conforms to organizational security policy.

When version 1.1 of the .NET Framework is installed, `migppol.exe` will be executed silently as part of this installation process to migrate from version 1.0.3705, if possible. Results will be as described above – if custom policy extensions are found, migration will not be possible.

`mscorcfg.msc` – .NET Framework Configuration Tool

`mscorcfg.msc` is a Microsoft Management Console snap-in. A wide variety of administrative tasks may be performed using this tool. This is the primary means of administering the .NET Framework. Each version of the .NET Framework comes with a version of `mscorcfg.msc` built to administer that version of the .NET Framework. When using this tool to perform security-related tasks, changes will only be made to a single version of the .NET Framework.

See Appendix B for a detailed discussion of the tasks available using `mscorcfg.msc`.

`permview.exe` – .NET Framework Permission Request Viewer

`permview.exe` is distributed with the .NET Framework SDK. It is a console application that displays the permission requests made by an assembly, as well as the declarative permission constraints that ensure a specific security environment at various points in the code’s execution.

Syntax

```
permview [/output {output file}] [/decl] {assembly file name}
```

A full list of supported options can be displayed using:

```
permview
```

For multi-file assemblies, the file name given must be the name of the file containing the assembly manifest. The `/decl` argument will also include the declarative permission constraints embedded in the code.

Tasks

The following tasks can be performed using `permview.exe`:

- View an assembly's permission requests and declarative permission constraints

View an assembly's permission requests and declarative permission constraints

```
permview myapp.exe
```

```
permview /decl myapp.exe
```

The first example shown above displays the minimal, optional, and refused permission requests contained in the metadata for the assembly `myapp.exe`. The second example displays the permission request sets and also the declarative permission constraints at various points in the program. Each permission constraint is identified by the code component that declares the constraint, the permission involved, and what type of security environment is being checked or imposed. Some constraints (`Assert`, `Demand`, `LinkDemand`, and `InheritanceDemand`) seek to guarantee access to a resource. Of these, `Assert` is the only one that can actually modify the security environment. The others simply check for the existence of a prior constraint. Other constraints (`Deny` and `PermitOnly`) seek to guarantee that a resource cannot be accessed and can modify the security environment. See the .NET Framework SDK for details on these declarative security constraints.

peverify.exe – .NET Framework PE Verifier

`peverify.exe` is distributed with the .NET Framework SDK. It is a console application that determines whether an assembly is safe to run, that is, whether it can be effectively managed by the CLR or not. The CLR will refuse to execute code that fails the verification process, unless it has been given the `Skip Verification` permission. This tool can be used to determine whether an assembly's safety can be verified. An assembly may in fact be safe, even though the CLR (or `peverify.exe`) cannot verify that this is true.



Recommendation: *Although an assembly may be safe even though it fails `peverify.exe`, do not allow unverifiable code to execute in an operational environment by granting the `Skip Verification` permission unless the code is from a highly trusted source.*

Syntax

```
peverify {assembly file} [{option1}] [{option2}] ...
```

A full list of supported options can be displayed using:

```
peverify
```

The `/quiet` option will suppress all output except a one-line pass/fail message.

Tasks

The following tasks can be performed using `peverify.exe`:

- Validate and verify an assembly

Validate and verify an assembly

```
peverify lib.dll /md
```

```
peverify lib.dll /il
```

The first command shown above validates the file and metadata structures of the assembly `lib.dll`. The second command validates the file structure and verifies the code in `lib.dll` for safety.

```
peverify lib.dll
```

```
peverify lib.dll /md/il
```

The first command shown above validates the file and metadata structures of the assembly (as with the `/md` option) and, if no errors are found, verifies the code (as with the `/il` option). To force both checks to be done, use the second command shown above.

regedit.exe – Registry Editor

`regedit.exe` is useful as a console application to modify .NET Framework registry settings.

Syntax

```
regedit /s {registry file}
```

Tasks

The following tasks can be performed with `regedit.exe`:

- Change a registry setting from the console
- Enable or disable the Assembly Cache Viewer

Change a registry setting from the console

The command

```
regedit /s settings.reg
```

will merge the registry settings in `settings.reg` with the Windows registry. The specified file must be a registry script such as those created using the **Registry | Export Registry File...** pull-down menu option in the graphical interface to `regedit.exe`.

Enable or disable the Assembly Cache Viewer

To disable the Assembly Cache Viewer, and display the Windows folder tree when viewing the GAC, merge the following registry script using `regedit.exe`:

```
Windows Registry Editor Version 5.00
[HKEY_LOCAL_MACHINE\Software\Microsoft\Fusion]
"DisableCacheViewer"=dword:00000001
```

To enable the Assembly Cache View, replace the script value with `00000000` and merge the script.

secutil.exe – Microsoft .NET Framework Security Utility

`secutil.exe` is distributed with the .NET Framework SDK. It is a console application that displays properties of the strong name or Authenticode digital signature of an assembly.

Syntax

```
secutil [{"-c | -v | -hex"}] {"-s | -x"} {assembly file}
```

The above syntax description is the effective result of the full syntax specification. Some alternative parameter names exist. The list of valid parameters can be displayed on the command line using:

```
secutil
```

The optional output formats are designed to facilitate cutting and pasting the output into program source code. Available formats are C/C++/C# array initialization syntax (-c), Visual Basic array initialization syntax (-v), and hexadecimal (-hex). Hexadecimal format is the most compact and readable for administrative purposes. The default format is C/C++/C# array initialization syntax.

Tasks

The following tasks can be performed using `secutil.exe`:

- View an assembly's strong name
- View an assembly's publisher certificate

View an assembly's strong name

The command

```
secutil -hex -s assembly.dll
```

displays the strong name information for `assembly.dll`, or an error message if the specified assembly is not strong-named or its strong name cannot be verified. Since the `-hex` format is specified, the public key token will be displayed as a hexadecimal string.

`secutil.exe` will display the strong name information for a delay-signed assembly only if the assembly has been registered for strong name simulation. For more information, see the Register an assembly for strong name simulation task under `sn.exe` below.

View an assembly's publisher certificate

The command

```
secutil -hex -x assembly.dll
```

displays the X.509 software publisher's certificate for `assembly.dll`, or an error message if the specified assembly is not Authenticode signed. The certificate will be displayed as a hexadecimal string.

setreg.exe – Software Publishing State Tool

setreg.exe is distributed with the .NET Framework SDK. It is also available in most server versions of the Windows operating system. It is a console application that modifies registry settings (Software Publishing State keys) that govern how Authenticode digital signatures are verified. These settings affect how Publisher Membership Conditions are evaluated, which can affect CAS policy application.

Syntax

```
setreg [-q] [{<setting number> {true | false}} ...]
```

A full list of supported setting numbers can be displayed using the command

```
setreg -?
```

Every time setreg.exe is executed, it will display the Software Publishing State settings that are current after it has made any specified modifications, unless the -q option is used. An individual setting may be modified by specifying a setting number followed by the argument true or false (case-insensitive). Multiple settings may be changed with a single command by including more than one set of these values. For example, the command

```
setreg 9 true 10 false
```

will set setting 9 (Check the revocation list on time stamp signer) to true and setting 10 (Only trust items found in the Personal Trust Database) to false.

Tasks

The following tasks can be performed using setreg.exe:

- View publisher certificate verification settings
- Adjust publisher certificate verification settings

View publisher certificate verification settings

The command

```
setreg
```

will display the current Software Publishing State settings and take no other action.

Adjust publisher certificate verification settings

Table 22 illustrates the use of setreg.exe to modify Software Publishing State settings:

Task	Command
Disable trust for the Test Root	setreg 1 false
Enable certificate expiration checking	setreg 2 true
Enable certificate revocation checking	setreg 3 true
Disable revocation server fallback for individual and commercial publisher's certificates	setreg 4 false 5 false
Disable Java revocation server fallback for individual and commercial publisher's certificates	setreg 6 false 7 false
Disable trust for "Version 1 Signed Objects"	setreg 8 true
Enable time stamp signature revocation checking	setreg 9 true
Enable trust only for software publishers in the Personal Trust Database	setreg 10 true

Table 22. `setreg.exe` Command Examples.

sn.exe – .NET Framework Strong Name Utility

`sn.exe` is distributed with the .NET Framework SDK. It is a console application that is used to strong name an assembly, enroll a delay-signed assembly to simulate a full strong name in the context of a development environment, or perform other strong-name related tasks.

Syntax

```
sn [-q] {option} [{option parameters}]
```

The list of valid parameters can be displayed on the command line using:

```
sn
```

The `-q` option turns off all output except error messages.

Tasks

The following tasks can be performed using `sn.exe`:

- Enroll an assembly for strong name simulation
- Withdraw an assembly from strong name simulation
- List assemblies enrolled for strong name simulation
- Verify an assembly's strong name

- View the public key token corresponding to the public key in an assembly's manifest
- Strong name an assembly
- Set the CSP used by the CLR when strong-naming assemblies

Enroll an assembly for strong name simulation

```
sn -Vr {{assembly filename} or * or *,{public key token}}   
    [{user1,user2,...}]
```

This command adds an entry to the list of assemblies enrolled for strong name simulation. The list is maintained as a set of registry keys under HKLM\Software\Microsoft\StrongName\Verification. The assembly filename must be the name of a file that contains an assembly manifest, typically, the assembly file itself. The assembly name that is enrolled is the name of the assembly as contained in the manifest, which is not necessarily the name of the file. The named file must be able to be opened by `sn.exe`, as it reads the assembly name from the manifest. Multiple entries may be created for the same assembly name as long as the public key tokens are different.

Unless a list of users is specified, an entry will be applied to all logged on users. Once an entry is added for an assembly, additional users cannot be incrementally added. Each time an entry is created for an assembly and public key token pair, it overwrites any previously enrolled entry with the same parameters, including the list of targeted users.

```
sn -Vr lib.dll alice,bob
```

The example shown above will create an entry that corresponds to the assembly name and the public key token contained in the assembly's manifest. Strong name verification will be simulated only when alice or bob is logged on.

```
sn -Vr *,1a2b3c4d5e6f7890 alice,bob
```

```
sn -Vr *
```

The first example shown above will create an entry that applies to any assembly delay signed with the given public key (as before, only when alice or bob is logged on). The second example applies to all assemblies for all users. All delay signed assemblies will be considered fully strong named for the purposes of CAS policy.

Withdraw an assembly from strong name simulation

```
sn -Vu {{assembly filename} or * or *,{public key token}}
```

This command removes an entry from the list of assemblies enrolled for strong name simulation. If more than one entry exists for an assembly filename (with different public key

tokens), the entry that corresponds to the public key token contained in the manifest in the given file will be removed.

Withdrawal cannot be applied to individual users. To withdraw an assembly for a specific subset of the users listed in an entry, use the Enroll an assembly for strong name simulation task (-vr) to overwrite the entry with one that omits the desired set of users.

List assemblies enrolled for strong name simulation

```
sn -vl
```

This command lists the assemblies and/or keys enrolled for strong name simulation, and the user accounts in which the list entries will be applied.

Verify an assembly's strong name

```
sn -v lib.dll
```

```
sn -vf lib.dll
```

The first command shown above verifies the strong name of the assembly whose manifest is contained in the file lib.dll. If the assembly has been delay signed, this will only succeed if the assembly has been registered for strong name verification skipping for the currently logged-on user. If the assembly has been strong named and the contents of the assembly have been corrupted or modified, this will not succeed.

The second command shown above will verify the strong name of the assembly even if it has been registered for strong name verification skipping for the logged-on user. In this case, a delay signed assembly will fail the verification check.

Note that in quiet mode (-q), there is no output on success.

View the public key token corresponding to the public key in an assembly's manifest

```
sn -T lib.dll
```

The public key token is the last 8 bytes in reverse order of the SHA-1 hash of the public key data.

Strong name an assembly

```
sn -R control.dll KeyPair_WebControls.snk
```

```
sn -Rc spreadsheet.exe StrongNameKeys_OfficeAutomationApps
```

The first command shown above strong names the assembly contained in the file control.dll using the public/private key pair contained in the file

KeyPair_WebControls.snk. If the assembly has already been strong-named, or has been delay signed, the strong name is replaced by one using the specified keys.

The second command shown above strong names the assembly contained in the file `spreadsheet.exe` using the public/private key pair contained in the key container named “StrongNameKeys_OfficeAutomationApps.”

Set the CSP used by the CLR when strong-naming assemblies

```
sn -c "Microsoft Enhanced Cryptographic Provider v1.0"
```

This command sets the CSP used to strong name assemblies to the Enhanced Provider. The text argument is a cryptographic service provider friendly name stored in the registry under `HKLM\Software\Microsoft\StrongName\CSP`. Possible values for this registry key are the friendly names associated with each installed cryptographic service provider, for example, “Microsoft Base Cryptographic Provider v1.0,” “Microsoft Enhanced Cryptographic Provider v1.0,” or “Microsoft Strong Cryptographic Provider.” If the registry key does not exist, the default CSP is used. This default can vary from system to system. If no CSP name is specified, the registry key is deleted, and subsequent attempts to strong name assemblies will use the default CSP.

storeadm.exe – .NET Framework Isolated Storage Tool

`storeadm.exe` is a console application distributed with the .NET Framework SDK that can be used to display or remove the Isolated Storage data stores that have been created by managed code running in the current user’s process.

Syntax

```
storeadm [/quiet] [/roaming] [/list] [/remove]
```

The list of valid parameters can be displayed on the command line using:

```
storeadm
```

The `/quiet` parameter turns off all output except error messages.

Tasks

The following tasks can be performed using `storeadm.exe`:

- List all local or roaming data stores associated with the current user
- Remove all local or roaming data stores associated with the current user

List all local or roaming data stores associated with the current user

The commands

```
storeadm /list
```

```
storeadm /roaming /list
```

list all local and roaming data stores for the current user, respectively.

Remove all local or roaming data stores associated with the current user

The commands

```
storeadm /remove
```

```
storeadm /roaming /remove
```

removes all local and roaming data stores for the current user, respectively. `storeadm.exe` cannot be used to selectively remove individual data stores. Note that if `/list` and `/remove` are both used in the same command, the tasks will be performed in the order in which the parameters appear.

Summary

This presentation of guidelines for the performance of common security-related tasks using specific tools is intended to serve as a brief summary in one location of the administrative use of the diverse set of .NET Framework tools. In addition, some additional administrative guidance was included where relevant.

Recommendations in This Section



Recommendation: Make Code Group names unique across the entire Code Group tree for any given CAS policy level.



Recommendation: Back up both the source and the target CAS policies at the Enterprise and Machine levels before running `migpol.exe`.



Recommendation: Review any policy migrated using `migpol.exe` to ensure that it conforms to organizational security policy.



Recommendation: Although an assembly may be safe even though it fails `pverify.exe`, do not allow unverifiable code to execute in an operational environment by granting the Skip Verification permission unless the code is from a highly trusted source.

Appendix

B

mscorcfg.msc – The .NET Framework Configuration Tool

The .NET Framework Configuration Tool (`mscorcfg.msc`) is a Microsoft Management Console snap-in that allows an administrator to perform common tasks associated with configuring and managing .NET applications and their components. `mscorcfg.msc` automates the modification of select portions of various XML files, helping to maintain the internal consistency of the XML data. When making configuration changes through this tool, it is not necessary to explicitly save changes; modifications are written immediately to the XML files with no rollback or undo function available.



Recommendation: Create frequent backups of configuration files administered using `mscorcfg.msc`. This can be done by making a copy of the `CONFIG` folder for each installed version of the .NET Framework. For hosts with specialized policy-driven configurations, copies of these files should be stored away from the host to facilitate recovery and restoration of host operation.

`mscorcfg.msc` only assists with a select subset of the configuration data. The tasks available through the tool are organized under five console tree nodes, as shown in Figure 41:

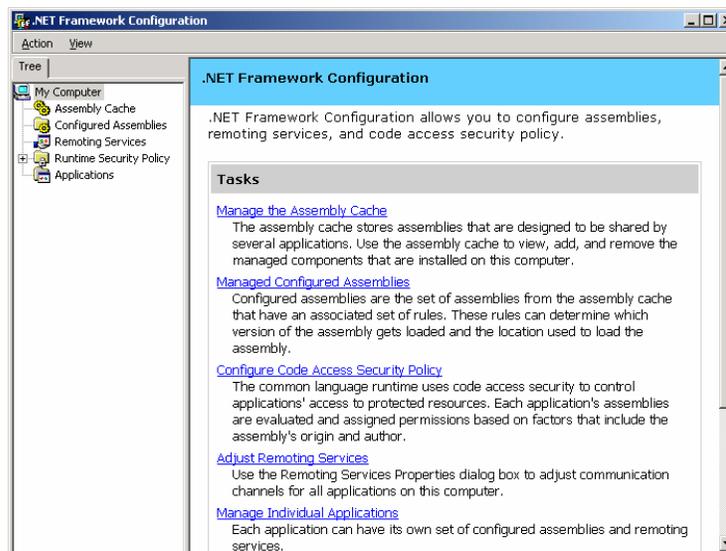


Figure 41. The .NET Framework Configuration Console (`mscorcfg.msc`).

The following list summarizes the content of the five console tree nodes. Each node is described more fully later.

-  Assembly Cache – The assembly cache is a virtual folder that contains assemblies shared by multiple .NET applications. Unlike the System32 folder for DLLs in Microsoft Windows, this virtual folder allows the presentation of different versions of the same assembly. In order to provide assurance of the integrity of shared assemblies as well as to uniquely identify assemblies that have the same library name, only strong-named assemblies may be installed in the cache. This console tree node contains tasks to view and modify the contents of the cache. The assembly cache is also known as the Global Assembly Cache (GAC).
-  Configured Assemblies – Configured assemblies are assemblies within the GAC that have an associated Binding Policy and CodeBase. A Binding Policy redirects requests for one version of an assembly to a different version. The CodeBase determines where each specified version of an assembly is to be obtained. This console tree node contains tasks related to viewing and modifying the Binding Policy and CodeBase properties of assemblies.
-  Remoting Services – The .NET Framework isolates applications in application domains, analogous to the process isolation enforced by the Windows operating system. Each operating system process that is providing an execution environment for managed code may contain multiple application domains. Remoting Services provides support for authorized communications between .NET Framework applications running in different application domains, whether those application domains are hosted by the same operating system process, by different processes, or even on different computers in the local network or Internet. This console tree node contains tasks relating to configuring channels for inter-application domain communication. Some additional Remoting Services tasks are available through the Applications node described below.
-  Runtime Security Policy – Code running in the .NET Framework execution environment is subject to a fine grained security policy that controls access to protected resources. The policy defines an access control matrix of code subjects administratively defined by evidence and resource objects and associated permissions defined by the extensible .NET Framework policy infrastructure. This console tree node contains tasks related to policy creation and deployment, and extension of the policy infrastructure through the specification of custom policy enforcement components. Runtime Security Policy is more commonly known as Code Access Security (CAS) policy.
-  Applications – .NET Framework applications are built from one or more assemblies. Most applications depend on several of the Common Language Runtime library assemblies provided with the .NET Framework. The component and dependent assemblies of an application can be viewed through this console tree node. The tasks described in the Configured Assemblies console tree node above can be performed on each component assembly, including shared assemblies. In addition, application-specific Remoting Services tasks can be performed to specify the application components that will handle communications across application domain boundaries.

In addition to the above tasks, the root console tree node “My Computer” contains the .NET Framework version’s Garbage Collection Mode property. This specifies a default mode for applications that do not specify their own preference in an application configuration file. See the .NET Framework SDK for information about this performance-related setting.

Each version of the .NET Framework comes with a version of `mscorcfg.msc` hard-coded to configure that version. In order to prevent inadvertent misconfiguration when multiple versions of the .NET Framework are installed, it is recommended that any shortcuts to `mscorcfg.msc` be renamed to indicate which .NET Framework version they are used to administer.



Recommendation: *Rename any shortcut to `mscorcfg.msc` to reflect the version of the .NET Framework it is designed to configure. Example: “`mscorcfg v1.1.4322`”*



Assembly Cache

The Assembly Cache node manages all assemblies located in the GAC. The administrator can add or remove assemblies from the GAC, or view a list of all assemblies in the GAC.

Assembly Cache Tasks

The following tasks can be performed under the Assembly Cache node:

- View cache contents
- Add an assembly to the GAC
- Delete an assembly from the GAC
- View properties of an assembly installed in the GAC

View cache contents

To view the contents of the GAC:

- Select the **Assembly Cache** node in the console tree pane and click on **View List of Assemblies in the Assembly Cache** in the details pane on the right

or

Right-click on the **Assembly Cache** node and select **View, Assemblies** from the context menu

or

Select **View, Assemblies** from the menu bar.

The details pane on the right will display the assemblies in the GAC. The columns displayed may be modified by selecting **View, Choose Columns...** from the menu bar. Figure 42 shows the list of assemblies in the GAC as displayed in `mscorcfg.msc`. The four displayable columns are the components of a strong name. Each assembly in the GAC is uniquely identified by the combination of all four values:

- **Assembly Name.** Each assembly is represented by an icon indicating whether the assembly is processor-independent CIL (Common Intermediate Language, a.k.a., MSIL/Microsoft Intermediate Language) or has been pre-compiled into native executable code. Managed code is typically in CIL which is compiled into native executable code only as needed by the CLR's Just-In-Time (JIT) compiler. Native code assemblies are in a special cache called the Zap cache, which is displayed with the GAC.



Assembly contains CIL code.



Assembly is pre-compiled into native executable code.

- **Version.** The four components of an assembly's version are the major version, minor version, build number, and revision number.
- **Locale.** This is also known as the assembly's culture. This refers to language, date/time formatting, and other aspects of the user interface that are not content-related. A neutral culture will use the default culture information for the local .NET Framework installation.
- **Public Key Token.** This is a 64-bit (8-byte) hash of the public key, generated to save space. The actual public key may be much larger – a 1,024-bit/128-byte RSA key is not uncommon.

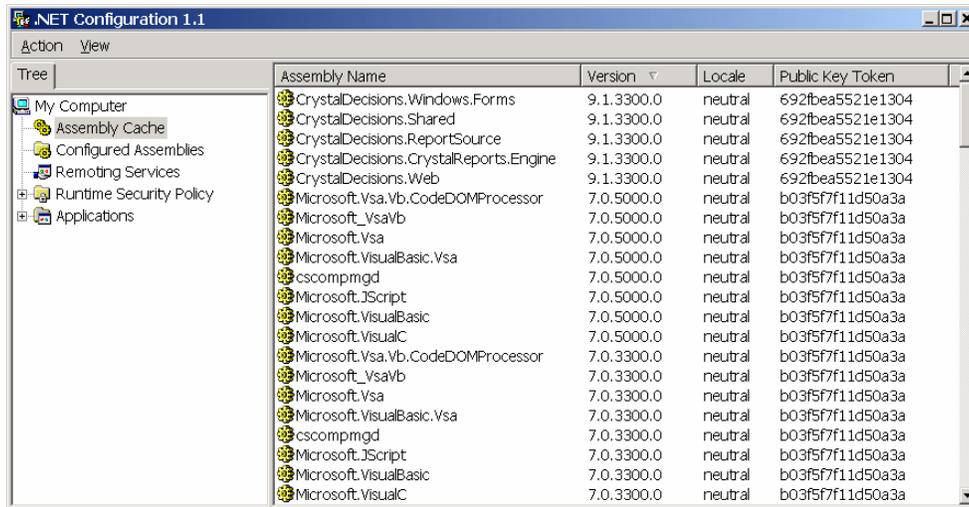


Figure 42. GAC Display in mscorcfg.msc.

Add an assembly to the GAC

Administrators (either local, domain, or enterprise) may add strong-named assemblies to the GAC. Since the GAC is intended to hold multiple versions of the same assembly as well as shared assemblies from a variety of sources, the strong name requirement ensures that different versions of the same assembly and assemblies with the same name (e.g., “helper.dll” or “util.dll”) from different sources will be distinguished. To add an assembly to the GAC:

- Select the **Assembly Cache** node in the console tree pane and click on **Add an Assembly to the Assembly Cache** in the details pane on the right

or

Right-click the **Assembly Cache** node and select **Add...**

or

Select **Action, Add...** from the menu bar.

- Browse in the Open File dialog box to the assembly to add to the GAC and click Open. An error will be produced if the selected file is not a strong-named assembly or the logged on user does not have administrative privileges.

Delete an assembly from the GAC

Although the details pane for the Assembly Cache node does not display the option to remove an assembly from the GAC, this task may be performed when the list of assemblies in the GAC is being displayed. Administrative privilege (either local, domain, or enterprise) is needed to remove an assembly from the GAC. To remove an assembly from the GAC:

- View the list of assemblies in the GAC.
- Right-click the assembly to be removed and select **Delete** from the context menu

or

Highlight the assembly to be removed and select **Action, Delete** from the menu bar.

Assemblies, such as the CLR libraries, installed in the GAC when the .NET Framework was installed cannot be removed. Only assemblies installed by the administrator can be removed. Assemblies removed from the GAC's virtual directory have not been deleted from the file system. They reside in their original installation directory. To completely remove the assembly from the host computer, simply delete the assembly's files from its installation directory.

View properties of an assembly installed in the GAC

- View the list of assemblies in the GAC.
- Right-click on the assembly and select **Properties**.

The Properties dialog box will display the following details about an assembly:

- Assembly's strong name components: Name, Version, Culture, and Public key token.
- Date the assembly was last modified. Some assemblies do not display this data.
- Codebase: This is the assembly's installation directory. This path can be used to completely remove an assembly from the local machine. Removing an assembly from the GAC will not delete it from this directory. Some of the CLR libraries do not display Codebase data.
- Cache type: "Gac" or "Zap". The GAC is the virtual repository for managed code. The Zap Cache is the virtual repository for assemblies pre-compiled into native executables.

Configured Assemblies

Configured assemblies are assembly version families that have an associated Binding Policy and CodeBase. A Binding Policy allows the administrator to redirect references to one version of an assembly to a different version of the same assembly. The CodeBase is a list of assembly locations that allows the administrator to specify the URLs from which particular redirected versions of an assembly are to be obtained. These locations could be local or remote, including the Internet.

The Binding Policies and CodeBases for assembly families are stored in the .NET Framework configuration file `machine.config`. Each version of the .NET Framework has its own configuration file and must be configured separately. The default installation of the .NET Framework does not include any assembly configurations.

In order to identify versioning families of assemblies, the assemblies should be strong-named. Binding Policy and CodeBase settings will apply to all versions of an assembly having the same assembly name and public key.

Configured Assemblies Tasks

The following tasks can be performed under the Configured Assemblies node:

- View the list of configured assemblies
- Configure an assembly
- Delete the configuration information for an assembly

View the list of configured assemblies

To view the list of configured assemblies:

- Select the **Configured Assemblies** node in the console tree and click on **View List of Configured Assemblies** in the details pane on the right

or

Right-click on the **Configured Assemblies** node and select **View, Assemblies** from the context menu

or

Select the **Configured Assemblies** node in the console tree and select **View, Assemblies** from the menu bar.

The details pane on the right will display the configured assemblies. The columns displayed may be modified by selecting **View, Choose Columns...** from the menu bar.

Configure an assembly

Before an assembly can be configured, it must be added to the list of configured assemblies. This will create an entry in the .NET Framework version's `machine.config` file. Once this is done, the Binding Policy and CodeBase settings may be configured through the Properties dialog box.

To add an assembly to the list of configured assemblies:

- Select the **Configured Assemblies** node in the console tree and click on **Configure an Assembly** in the details pane on the right.

or

Right-click on the **Configured Assemblies** node and select **Add...** from the context menu.

or

Select the **Configured Assemblies** node and select **Action, Add...** from the menu bar.

- Select the assembly to configure using the Configure an Assembly dialog box (Figure 43) and click **Finish**. The assembly may be selected from the GAC (by clicking **Choose Assembly...**) or the information can be entered manually.

Once the assembly has been identified, the Properties dialog box is displayed, and the assembly may be immediately configured. See Using the Properties dialog box below for details.

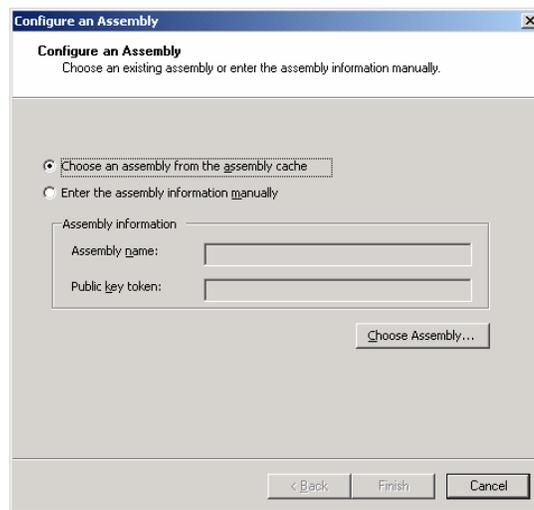


Figure 43. Adding a Configured Assembly.

To invoke the Properties dialog box for an assembly that is already on the list of configured assemblies:

- Display the list of configured assemblies. See the View the list of configured assemblies task above for more details.
- Right-click on the desired assembly and select **Properties** from the context menu

or

Select the desired assembly and select **Action, Properties** from the menu bar.

Using the Properties Dialog Box

The Binding Policy and CodeBase settings can be configured through the Properties dialog box. The **General** tab (Figure 44) simply shows the assembly name and public key token, and does not contain configurable information:



Figure 44. Assembly Properties Dialog Box – General Tab.

The **Binding Policy** tab (Figure 45) allows the configuration of a set of version redirection entries. A request for a version specified in the **Requested Version** column will be redirected to the version specified in the **New Version** column:

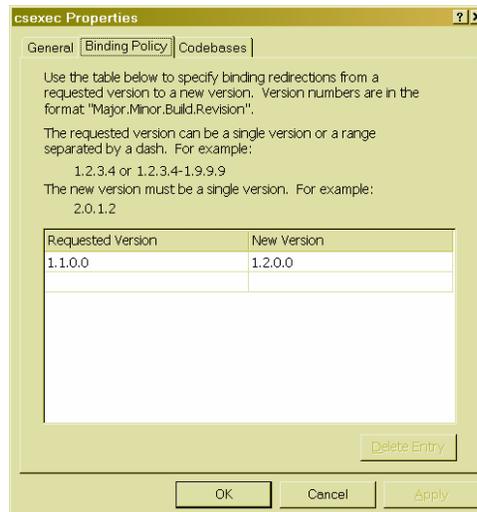


Figure 45. Assembly Properties Dialog Box – Binding Policy Tab.

For example, if version 1.2.0.0 of the assembly is compatible with any application that used version 1.1.0.0, and the use of version 1.1.0.0 has been disallowed by policy, then a Binding Policy could be configured to redirect all references to version 1.1.0.0 of the assembly to version 1.2.0.0.

Binding Redirection is not chained. If one entry redirects version 1.0.0.0 to version 2.0.0.0, and another entry redirects version 2.0.0.0 to version 3.0.0.0, then the entries will not be chained to redirect version 1.0.0.0 to version 3.0.0.0.

The **CodeBases** tab (Figure 46) allows the specification of a preferred location from which to obtain specified versions of the assembly. The location of a strong-named assembly can be anywhere on the local machine, the local intranet, or the Internet:

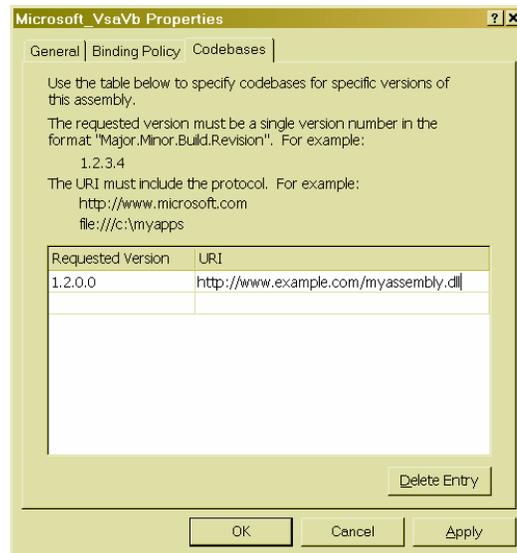


Figure 46. Assembly Properties Dialog Box – Codebases Tab.

The CodeBase settings for a version of an assembly will not be applied unless the version has been the target of a prior version redirection (and this cannot be an “identity” redirection from version X to version X).

Delete the configuration information for an assembly

To remove an assembly’s configuration data from `machine.config`, perform the following steps:

- View the list of configured assemblies.
- Right-click the configured assembly, select **Delete** from the context menu, and click **Yes** in the confirmation dialog box.

Deleting the configuration information for an assembly will remove it from the list of configured assemblies and remove the configuration information pertaining to the assembly

from the `machine.config` file corresponding to the .NET Framework version administered by the current instance of `mscorcfg.msc`. It will not delete the assembly or remove it from the GAC.

Remoting Services

Remoting Services is a feature of the .NET Framework that supports communication between managed applications, where an application is defined as all the .NET Framework code components that reside within the same Application Domain. Since the Application Domain is the basic unit of execution isolation for the .NET Framework, code running in one Application Domain cannot directly communicate with code running in another Application Domain, even when the two Application Domains reside within the same Windows operating system process. Through the Remoting Services facility, the CLR performs the vast majority of the work required to set up inter-Domain communication, allowing managed code components in different Application Domains to communicate with minimal effort whether the Application Domains reside in the same process, or across the Internet. Since the Application Domain defines the basic boundary at which Remoting Services becomes necessary, it is appropriate to talk about applications communicating via Remoting Services rather than assemblies, and the Remoting Services configuration reflects this approach. Administratively configuring communication between applications is essentially the same whether the applications are both local or are on different host computers across the Internet.

Remoting Services does not provide any built-in authentication or encryption. Without any customization, the built-in HTTP and TCP channels will transmit data unencrypted and unauthenticated. The use of cryptography to enable secure communications must be implemented using one of the following:

- The use of custom software that extends the Remoting infrastructure
- The security features of ASP.NET when hosting service applications in Microsoft IIS (Internet Information Services)
- Security features employed at the IP or lower protocol layers

The configuration of such security implementations is not covered in this document.

Terminology

In the discussion that follows, the term “object” will refer to managed software contained in an assembly that can execute a set of specific named functions. The “type” of an object defines the set of functions it is designed to perform. Any managed application consists of multiple interacting objects. Using Remoting Services, an application (the client) can request the execution of a function by an object of a particular type that is part of another application (the service). The CLR managing the execution of the client application uses that application’s Remoting Services configuration to determine how to pass request messages to the service application. The CLR managing the execution of the service application uses the

service application's Remoting Services configuration to determine how to listen for and respond to messages from client applications. Communication can only take place if the configurations are compatible, and the service application is actively listening for messages from the client.

We will refer to an object as having been “remoted” by its containing service application when it is configured to respond to function requests from client applications. Note that remoted objects are only “remote” from the point of view of the Application Domain – the client and service application may both reside in the same Windows process. In fact, an application could talk to itself using Remoting Services.

Remoting Services Tasks

The following discussion of the Remoting Services configuration elements serves as a guide to the use of those settings, but does not include specific procedures for individual tasks.

Remoting Services Configuration Files

Configuring Remoting Services for an application consists primarily of identifying a remoted object and setting some parameters for how communication with that object will take place. For a client application, the parameters tell the CLR how the client intends to communicate with the remoted object. An application may be a client to a wide variety of remoted objects located on the local host, the local Intranet, or the Internet. Note that the Remoting Services configuration does not specify which assemblies or code components of an application will actually perform the communication – the unit of configuration is the application and identifying a remoted object allows any assembly loaded in the application's Application Domain to communicate with the object, subject to authorization via CAS policy, of course. For a service application that is hosting a remoted object, the parameters tell the CLR how the object must be communicated with. The same assembly may function as both a service and a client.

The system-wide configuration file for each .NET Framework version, `machine.config`, can contain a default Remoting Services application configuration. This default configuration will apply to all applications that are not granted the Security permission `Enable Remoting Configuration`, which allows an assembly to configure the use of Remoting Services in its Application Domain. Applications may have their own Remoting Services configuration in an XML file (typically named `{application file name}.config`). Application-specific settings take precedence over the default settings in `machine.config`.

Only the machine configuration is automatically loaded and applied by the CLR. Note that `mscorlib.config` can only be used to configure settings in an application configuration file whose file name has the format `{application file name}.config`. An arbitrary file name cannot be specified. No matter which configuration file is used, the XML structure will be the same. Application settings will be stored under the `<application>` element in the `<system.runtime.remoting>` section. The XML structure of the Remoting Services configuration is illustrated in Figure 47:

```

<configuration>
  <configSections>
    <section name="system.runtime.remoting"
type="System.Configuration.IgnoreSectionHandler, System,
Version=1.0.5000.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
allowLocation="false"/>
  </configSections>
  ...
  <system.runtime.remoting>
    <application>...</application>
    <channels>
      <channel>...</channel>
      ...
      <channel>...</channel>
    </channels>
    <channelSinkProviders>
      <clientProviders>
        <provider .../>
        ...
        <provider .../>
        <formatter .../>
        ...
        <formatter .../>
      </clientProviders>
      <serverProviders>
        <provider .../>
        ...
        <provider .../>
        <formatter .../>
        ...
        <formatter .../>
      </serverProviders>
    </channelSinkProviders>
  </system.runtime.remoting>
</configuration>

```

Figure 47. XML Structure of the Remoting Services Configuration.

The `<application>` element will be discussed at length below. The `<channel>`, `<provider>`, and `<formatter>` elements are the other basic building blocks of the Remoting Services configuration. When specified outside of an `<application>` element,

these elements constitute stock definitions that can be referenced multiple times inside of the <application> element. Their use and format will be discussed where they occur within the <application> element.

The <application> element

An application's Remoting Services settings consist of five components:

- Identification of remoted objects in other applications that this application wants to talk to. When talking with these objects the application will be in the role of client.
- Identification of remoted objects that are part of this application that other applications can talk to. When other applications talk with these objects, this application will be in the role of service.
- Identification of the ways that this application intends to communicate. When a client application wants to talk to a remoted object in a service application, both applications must be configured in compatible ways or communication cannot take place.
- How long a service application should keep a remoted object around to allow a client application to talk to it repeatedly.
- A dictionary of XML elements and types used in SOAP messages and corresponding software components that will process those elements.

These settings correspond to five XML elements under the <application> element:

```
<application name="...">
  <client>...</client>
  ...
  <client>...</client>
  <service>...</service>
  ...
  <service>...</service>
  <channels>
    <channel>...</channel>
    ...
    <channel>...</channel>
  </channels>
  <lifetime/>
  <soapInterop>...</soapInterop>
</application>
```

Figure 48. XML Structure of the <application> Element.

The <channel> elements may appear within the <application> element and/or as a peer element. Both forms are discussed under the <channel> element below.

The behavior of a remoted object is determined by its configuration as either a well-known or client-activated object. This configuration is the prerogative of the service application – a client application that wishes to communicate with a remoted object must be configured the same way as the service.

Well-known (Service-activated) objects

Well-known objects are managed by their service application in two ways. Either there is one object that all client applications talk to (Singleton mode), or a new object is created each time any function is requested (SingleCall mode). In either case, well-known objects don't store separate information for each client. Either data is stored in common, or it isn't stored at all. In Singleton mode, data stored by the remote object could potentially be shared by other clients. In SingleCall mode, no data is stored – each time the object is accessed, it begins in its initial state and is deleted after it has performed its function.

Well-known objects are also known as server-activated objects, because their creation and lifetime is determined solely by the service configuration.

Client-activated objects

A client-activated object is created when a client application asks the service to create it. Client applications can talk repeatedly to the same object, and it can store data specific to that conversation. Moreover, a client application can create and talk to multiple instances of the same type of object at the same time.

The lifetime of a client-activated object is based on a lease concept. A client-activated object is leased by the client application when it is created. This initial lease specifies a lifetime for the object. When the lease expires, the object is terminated, and subsequent communication attempts by the client application will fail. Each time a client application communicates with the object, its lease may be extended. The leasing settings can be administratively configured for the service application using the <lifetime> element. Client applications cannot control the leasing settings.

The <client> element

The <client> elements identify remoted objects that an application can talk to as a client. It has the following XML structure (Figure 49):

```
<client url="..." displayName="...">
  <activated type="..." />
  ...
</client>
```

```

<activated type="..."/>
<wellknown type="..." url="..." displayName="..."/>
...
<wellknown type="..." url="..." displayName="..."/>
</client>

```

Figure 49. XML Structure of the <client> Element.

From the client perspective, a remoted object is identified by an object type and a URL. An object's type is given by the identification of an assembly and one of the object types defined in that assembly. The URL identifies the location of a service application that will host objects of this type. Each <wellknown> child element defines its own type and URL pair. The `url` attribute of the <client> element provides the URL for all of the <activated> child elements. Because of this scheme, the <wellknown> elements may appear as child elements of any <client> element, but the <activated> elements must be grouped by URL under corresponding <client> elements.

The `displayName` attribute of the <client> element is used only by `mscorcfg.msc` to display a descriptive name of the remote service application whose objects are described by the child elements. These names are shown in the **Select the remote application to configure** drop-down box in the **Remote Applications** tab of the Remoting Services Properties dialog box (Figure 50). Once the name of the service application has been selected, the information for the corresponding <client> element will be displayed. The `url` attribute will be displayed in the **The selected remote application is located at the following URL** text box. This property is editable through the dialog box. Each server-activated object (corresponding to the <wellknown> elements) will be shown in the table, with the `type` and `url` attributes shown in the **Object Name** and **URL** columns, respectively. The URLs may be edited through the dialog box. Client-activated objects (corresponding to the <activated> elements) are not shown.

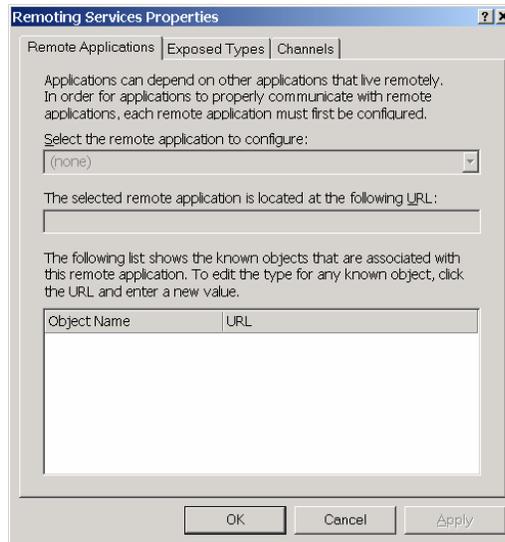


Figure 50. Remoting Services Properties Dialog Box – Remote Applications Tab.

The `displayName` attributes of the `<wellknown>` elements are not displayed; in fact, these attributes will be discarded by `mscorcfg.msc` if any changes are made to the `<client>` element through the Remote Applications tab of the Remoting Services Properties dialog box.

The <service> element

The `<service>` elements identify remoted objects that the application will host as a service for others to talk to. It has the following XML structure (Figure 51):

```
<service>
  <activated type="..." />
  ...
  <activated type="..." />
  <wellknown type="..." objectUri="..." mode="..." displayName="..." />
  ...
  <wellknown type="..." objectUri="..." mode="..." displayName="..." />
</service>
```

Figure 51. XML Structure of the `<service>` Element.

From the service perspective, a remoted object is simply identified by an object type. In addition, some hosted objects may be given a “well-known” nickname using the `objectUri` attribute. This nickname (also called the object’s “endpoint”) is used as the last component of the URL of the remoted object. The complete URL for objects hosted by the service application will include:

- The protocol and port as determined by the channel configured for this application

- The network domain name of the server host machine
- The name of the service application, from the `name` attribute of the `<application>` element. Services hosted by IIS do not use this attribute.
- The `objectUri` attribute

The URL will have the form:

```
{protocol}://{domain name}:{port}/{service application name}/{objectUri}
```

The `mode` attribute of a well-known type specifies how the object's creation will be handled by the server. The object creation behavior can be either "Singleton" (all clients talk to the same object) or "SingleCall" (every message from every client is handled by a new object of the specified type).

The `displayName` attribute of a `<wellknown>` element is used only by `mscorcfg.msc` to further describe the remoted object for ease of configuration. This text will appear in the **Object Name** column of the **Exposed Types** tab of the Remoting Services properties dialog box (Figure 52). The **URI** column will show the corresponding `objectUri` attribute value.

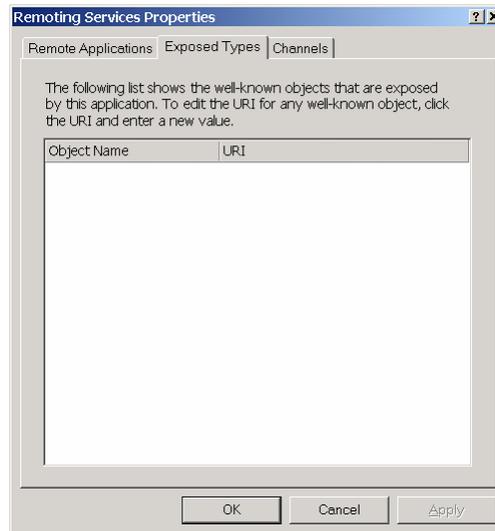


Figure 52. Remoting Services Properties Dialog Box – Exposed Types Tab.

The `<activated>` elements are not shown, nor are the `mode` or `type` attributes of the `<wellknown>` elements.

Remoting Services Configuration Example 1

An object hosted through IIS on the website `remoting.example.com` is available through the URL `http://remoting.example.com/{objectUri}`. The assembly that contains

the remoted object must be located in the `bin` subdirectory of the virtual root directory associated with `remoting.example.com`. The client application's Remoting Services configuration should contain a `<client>` element with a `<wellknown>` child element containing the attributes `url="http://remoting.example.com/{objectUri}"` and `type="{object name, assembly name}"`.

Remoting Services Configuration Example 2

An object called `examples.remobjects.objectx` defined in the assembly `myobjects.dll` is hosted by a service application called `objserver.exe`, which resides on the network host `remoting.example.com`. If the object is to be registered as a well-known object, the service and client Remoting Services configurations may look like the following (Figure 53):

Service

```
<application name="objserver">
  <service>
    <wellknown type="example.remobjects.objectx, myobjects"
objectUri="abc" mode="SingleCall" displayName="Object X"/>
  </service>
  <channels>
    <channel ref="http server" port="12345"/>
  </channels>
</application>
```

Client

```
<application name="clientapp">
  <client displayName="Object Server">
    <wellknown type="example.remobjects.objectx, myobjects"
url="http://remoting.example.com:12345/objserver/abc"/>
  </client>
  <channels>
    <channel ref="http client"/>
  </channels>
</application>
```

Figure 53. Remoting Services Configuration Example 2.

Both the client and the service configurations agree on the protocol, the port, the remote service application name, and the object “endpoint” used by the service, in this case, the value “abc.”

Remoting Services Configuration Example 3

If the same object as in example 2 above is to be client-activated, the service and client Remoting Services configurations would look like the following (Figure 54):

Service

```
<application name="objserver">
  <service>
    <activated type="example.remobjects.objectx, myobjects"/>
  </service>
  <channels>
    <channel ref="http server" port="12345"/>
  </channels>
</application>
```

Client

```
<application name="clientapp">
  <client url="http://remoting.example.com:12345/objserver"
  displayName="Object Server">
    <activated type="example.remobjects.objectx, myobjects"/>
  </client>
  <channels>
    <channel ref="http client"/>
  </channels>
</application>
```

Figure 54. Remoting Services Configuration Example 3.

The client application may attempt to use an object of type `examples.remobjects.objectx` and the CLR will handle the communication behind the scenes. To the client application, it will look like the software is local, except that it may silently disappear if the lease expires.

The <channel> element

The `<channel>` element identifies the networking resources that will be used to send and receive messages. This includes the communication protocols and port numbers, the basic encoding type that the message will use, as well as any special message processing that is desired before and after transmission.

The basic type of message that .NET Remoting is designed to handle is to invoke a program function in a remote software object, passing parameters to that function and getting a return

value that is the output of the function. It is not necessary to configure a server channel to receive the response to such a message. The message to invoke the remote function and the response message containing the output of the function will both be handled by the client channel. Server channels are used for objects that expose their own functions for use by remote clients.

Since multiple applications may use very similar `<channel>` configurations, common `<channel>` elements (“templates”) may be defined to save configuration time, promote consistent usage, or maintain conformance to a policy. Templates are `<channel>` elements that are not child elements of any `<application>` element, but appear grouped in a `<channels>` element directly under the `<system.runtime.remoting>` element in a configuration file. In the machine configuration file, several different application configurations may refer to the same `<channel>` template element. Application configuration files may refer to `<channel>` template elements defined in the same file or any `<channel>` template defined in the `machine.config` file of the .NET Framework version that the application will use.

`<channel>` elements have two basic forms. The shorter form contains a reference to an existing `<channel>` template element. The full form is used to define `<channel>` templates or to configure a channel for an application without a reference to a predefined template. The short form has the following XML structure:

```
<channel ref="..." port="..." displayName="..." {name}="{value}"/>
```

The value of the `ref` attribute must match the value of the `id` attribute of some channel template.

The `port` attribute for a client application is used only to receive response messages from a service application. The destination port for messages to service applications will be determined by the service object’s URL, not by this attribute. A client application may set the `port` attribute to 0 to indicate that the Remoting Services system should select any appropriate port number. Service applications must specify a port or use a channel that defines a default port.

The software that handles the channel will be fully identified in the referenced `<channel>` template element. This software may support additional custom parameters, which may be specified with zero or more attributes defining custom (name, value) pairs.

The `displayName` attribute is used by `mscorcfg.msc` in the **Select the channel to configure** drop-down box in the **Channels** tab of the Remoting Services Properties dialog box (see Figure 55). The `port` and any custom attributes will be shown in the parameters list in the same tab.

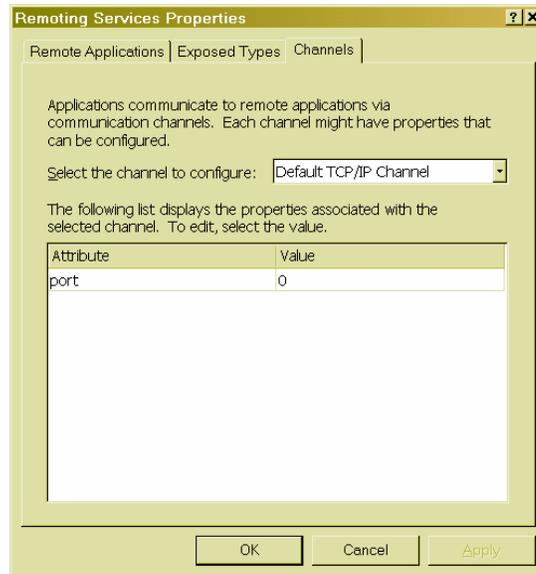


Figure 55. Remoting Services Properties Dialog Box – Channels Tab.

The full form of a `<channel>` element has the following XML structure (Figure 56):

```
<channel id="..." type="..." displayName="..." ␣
    delayLoadAsClientChannel="{true | false}" {name}="{value}">
  <clientProviders>
    <provider/>
    ...
    <provider/>
    <formatter/>
    ...
    <formatter/>
  </clientProviders>
  <serverProviders>
    <provider/>
    ...
    <provider/>
    <formatter/>
    ...
    <formatter/>
  </serverProviders>
</channel>
```

Figure 56. XML Structure of the `<channel>` Element.

The `id` attribute is only used for a template element. Its sole purpose is to provide an identifier that can be used in `ref` attributes of other `<channel>` elements.

The `type` attribute specifies the managed object that handles the channel communication, as well as the assembly that contains the object definition. The object type that is identified by the `type` attribute may support additional channel parameters. These may be specified by the custom property attributes. When the same custom property is specified in a template and in a `<channel>` element that references the template, the property value in the referencing (application-specific) `<channel>` element will replace the template value.

The `delayLoadAsClientChannel` attribute can be used for client applications that may use different networking protocols in different situations. By setting this attribute to true, a channel is identified as a delay load channel. This means that the channel will not be set up for communications until actually needed and actually supports the URL of the object to which the application wants to connect. For example, a (custom) FTP channel, a TCP channel and an HTTP channel are included in a client application's configuration file as `<channel>` elements, with the TCP and HTTP channels marked as `delayLoadAsClientChannel`. When the application is configured for Remoting, only the FTP channel will be set up for communications right away. If the client application wants to connect to a URL using the HTTP protocol, the Remoting Services facility will check each loaded channel to see if it supports communication to the indicated URL. In this case, the FTP channel is the only registered channel, and it does not accept the URL beginning with the "http://". Since no supporting channel was found, the delay load channels will be considered in turn until one is found that supports communication with the desired URL. In this case, the TCP channel is checked but not loaded since it does not support the desired protocol. The HTTP channel is then checked. Since it supports the desired communications, the networking infrastructure underlying the HTTP channel is now set up and initialized. The delay load option allows multiple channels to be specified as usable by a client application, but only the channels actually needed will be loaded. This attribute is not used for service applications.

As with the short form, any number of additional custom properties used by the channel object may be specified as attributes.

The `displayName` attribute of a `<channel>` template is used by `mscorcfg.msc` to display the channel templates defined in `machine.config`. This attribute plays no role in the Remoting Services functioning.

The `<clientProviders>` and `<serverProviders>` elements

Special pre- and post-transmission processing can be included by registering client and server side sinks. A sink is a software component that transforms a message before it is sent out (client sink) or after it has been received (server sink). An example might be a channel in which messages to be sent are first encrypted using a client sink and messages that are received must be decrypted using a server sink. Sinks can be chained together to make up a sequence of processing steps.

The client sinks used locally to prepare a message for transmission must be compatible with the server sinks used remotely to receive the message. Any transformation performed on the message before it is sent out must be handled by the server sinks. Likewise, any transformation that the server sinks performs on the response message before it is sent back to the client must be “undone” by the client sinks.

Processing of messages before transmission is performed by client sinks registered in the `<clientProviders>` element. Processing of messages received is performed by server sinks registered in the `<serverProviders>` element.

Sink chains are defined by sequences of `<provider>` and `<formatter>` child elements within the `<clientProviders>` and `<serverProviders>` elements. The order of processing will be the order in which the child elements appear. If no sinks are defined for a channel, default sinks will be used. Default sinks cannot be selectively overridden. If sinks are provided for a channel by specifying `<provider>` and/or `<formatter>` elements, none of the default sinks will not be used – they will be replaced by the specified sinks.

The `<provider>` and `<formatter>` elements

Sinks are actually defined by identifying a sink provider, an object that creates sinks. Each type of sink provider will create a sink of a particular type. For example, the `SdlChannelSinkProvider` object creates server sinks of type `SdlChannelSink`. There may be any number of sinks defined for a communications channel.

A formatter is a special kind of sink provider that determines the encapsulating format for the message data. For example, the `SoapClientFormatterSinkProvider` object creates sinks for client channels that encapsulate a function call in a SOAP message. There can be at most one formatter defined for each channel, although custom sinks may be defined both before and after the formatter sink in a sink chain.

Any `<provider>` or `<formatter>` element appearing in a `<channel>` template may be given an `id` attribute, and be referenced by a `<provider>` or `<formatter>` element, respectively, in an application-specific configuration using the `ref` attribute. The XML structures of `<provider>` and `<formatter>` elements are shown in Figure 57. The optional `id` or `ref` attributes are not shown. If a `ref` attribute is used, the `type` attribute should not be used.

```
<provider type="..." {name}="{value}"/>
<formatter type="..." includeVersions="{true | false}" ⌵
    strictBinding="{true | false}" typeFilterLevel="{Low | Full}" ⌵
    {name}="{value}"/>
```

Figure 57. XML Structure of the `<provider>` and `<formatter>` Elements.

Both `<provider>` and `<formatter>` elements may have any number of custom properties specified as attributes. The `<formatter>` elements have some additional options:

includeVersions

This attribute is for client formatter sinks only. When the `includeVersions` attribute is `“true”`, a precise identification of the type of remote object is sent with the message. This allows the client application to specify a preferred version of the remote object. The information sent includes that type name, assembly name, and assembly version, and, if strong named, also the assembly culture and public key token. The formatter sink on the server channel can use this information to determine which type and version of object will handle the message. If this attribute is `“false”`, only the type name and the assembly name will be sent.

strictBinding

This attribute is for server formatter sinks only, and is complementary to the `includeVersions` attribute for client channels. When the `strictBinding` attribute is `“true”`, the server formatter sink will attempt to use the most precise type and version of object specified. If the full version information has been transmitted (say, if the `includeVersions` attribute was `“true”` on the client channel), then the specific version of the object must be available or the message will not be handled. If this attribute is `“false”`, then full version information will be used if possible, otherwise, any available version will be used. If only the type name and assembly name were transmitted (say, if the `includeVersions` attribute was `“false”`), then the first available version of the object type will be used. These relationships are shown in Table 23:

<code>includeVersions</code> (client formatter)	<code>strictBinding</code> (server formatter)	Object type used
<code>true</code>	<code>True</code>	Use an object of the exact specification (type name, assembly name, assembly version, and possible assembly culture and public key token), or fail.
<code>true</code>	<code>False</code>	Use an object of the exact specification if available. If not available, use any object of the indicated type name and assembly name, or fail.
<code>false</code>	<code>Any</code>	Only the type name and assembly name are sent. Use any object of the specified type and assembly, or fail.

Table 23. Formatter Sink Attributes.

typeFilterLevel

The value of the `typeFilterLevel` attribute will allow or disallow passing some references to custom object types as parameters to remote function calls. Passing references to custom object types may pose a security risk to service applications. When the Remoting infrastructure includes a reference to a client object in a message, it is really passing information to the service application that tells it how to create a proxy object that can

communicate with the client and represent the client object as if it were in the service application's own Application Domain. The integrity of this information, which includes the location of the client object on the network, is critical to the correct behavior of the service application.

This attribute can be set to "Low" or "Full". A low filter level is used to indicate a higher degree of filtering and a lower filter acceptance level. A full filter level indicates that no filtering should be performed, and consequently a high filter acceptance level.

If the `typeFilterLevel` attribute is set to "Full", all references to custom client objects can be passed as parameters. If the `typeFilterLevel` attribute is set to "Low", most references to custom object types cannot be sent as parameters to service applications. This attribute is only available in the .NET Framework version 1.1. Version 1.0 of the .NET Framework does not do any filtering.

 ***Recommendation: Use authentication and encryption for all remoting channels when `typeFilterLevel` attribute of the channel's formatter sink is set to "full". Use authentication and encryption for all remoting channels in version 1.0 of the .NET Framework.***

Default channels in the Remoting Services configuration

Since the default Remoting Services configuration does not specify any actual remote objects, applications cannot communicate using Remoting Services unless they are granted the Security permission Enable Remoting Configuration. Those applications that are permitted to use Remoting Services can use the channels defined in the default Remoting Services configuration (Figure 58). These are not channel templates – they are actually available for use by applications that may use Remoting Services.

```
<application>
  <channels>
    <channel ref="http client"
      displayName="http client (delay loaded)"
      delayLoadAsClientChannel="true"/>
    <channel ref="tcp client"
      displayName="tcp client (delay loaded)"
      delayLoadAsClientChannel="true"/>
  </channels>
</application>
```

Figure 58. Default Remoting Services Channels.

The `ref` attribute of each `<channel>` element refers to a channel template defined elsewhere in `machine.config`. Properties for default channels can be changed using the Remoting Services Properties dialog box (Figure 59):

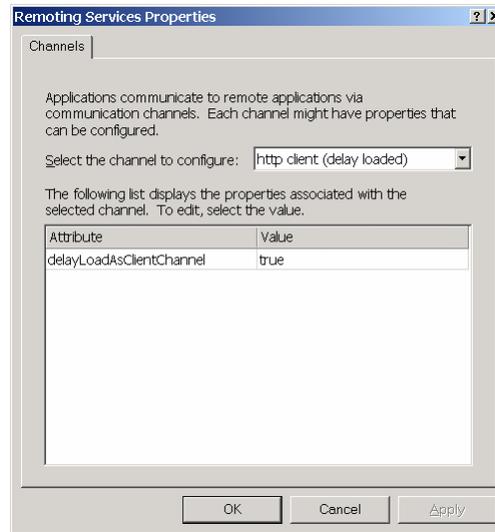


Figure 59. Remoting Services Properties Dialog Box – Channels Tab.

Additional channels must first be added to the XML file `machine.config` before they can be configured through `mscorcfg.msc`.

The <lifetime> element

The lifetime of a remoted object is controlled by its lease settings and the settings for the lease manager component of a service. The `<lifetime>` element has the following XML structure (Figure 60):

```
<lifetime leasetime="..." sponsorshipTimeout="..." renewOnCallTime="..."
    leaseManagerPollTime="..." />
```

Figure 60. XML Structure of the `<lifetime>` Element.

The use of each of the four attributes is described below. The value for each attribute is in the form “{n}{units}”, e.g., “4M” or “25H”, where {units} denotes the units of time and must be one of “D”, “H”, “M”, “S”, or “MS”, for days, hours, minutes, seconds, and milliseconds, respectively. If no unit of time is specified, seconds are assumed as a default.

leaseTime

This is the initial duration of the lease for all objects hosted by this service application. The default value is 5 minutes.

leaseManagerPollTime

The lease manager is the software that checks for expired leases. The `leaseManagerPollTime` attribute is the time interval between checks. The default value is 10 seconds.

renewOnCallTime

This is the length of time a lease is extended each time the object is used. A lease will not expire for at least this amount of time after an object is used. The default value is 2 minutes.

sponsorshipTimeout

When a client application may wait a long time between messages sent to the same object, but doesn't want the object's lease to expire, the client can specify sponsor software that the service lease manager will query when it is checking for expired leases. Sponsors are not administratively configurable – the client application must name its own sponsor, if any. The service's lease manager will check the sponsor when a lease has expired to see if the lifetime of the object should be extended even though the client application hasn't communicated with the object for a while. The sponsor can decide to keep the object alive by extending the lease, but if it does not respond within the amount of time given by the `sponsorshipTimeout` attribute, the object is terminated. The default time is 2 minutes.

The <soapInterop> element

When a message is passed between a client and a service, it may consist in part of objects whose complex data structures must be written out as a stream of bytes for transmission. If SOAP (or any similar XML-based format) is used as the message format, the data structures of the object will be represented by XML elements. Both the sender and the receiver need to understand the correspondence between the XML elements of the message and the types of object that the XML elements represent. The `<soapInterop>` elements are a means for an application to register these mappings.

The mappings can also be registered by the application in software, but this requires the Security permission `Extend Infrastructure`. When `<soapInterop>` elements are used to register mappings by means of the configuration file, only the Security permission `Enable Remoting Configuration` is needed. Thus, the SOAP XML-element-to-object-type mappings may be configured with either the `Enable Remoting Configuration` or the `Extend Infrastructure` permission.

The `<soapInterop>` element has the following XML structure (Figure 61):

```
<soapInterop>
  <interopXmlElement xml="..." clr="..."/>
  ...
  <interopXmlElement xml="..." clr="..."/>
  <interopXmlType xml="..." clr="..."/>
  ...
  <interopXmlType xml="..." clr="..."/>
  <preLoad type="..." assembly="..."/>
  ...
</soapInterop>
```

```

    <preLoad assembly="..." />
</soapInterop>

```

Figure 61. XML Structure of the <soapInterop> Element.

interopXmlElement

Each `interopXmlElement` element defines one mapping between an XML element and a managed object type. The `xml` attribute must be in the form "{element name}, {namespace}", and the `clr` attribute must identify an object type using the form "{object type}, {assembly name}" where the named assembly contains the definition of the specified type.

interopXmlType

Each `interopXmlType` element defines one mapping between an XML Schema type and a type of managed object. This type of mapping is needed when a "derived" XML Schema type is being transmitted under the name of its base type, perhaps to ensure validation against a schema that only knows about the base type. In such a case, the base type is used as the element name, while the actual content of the element corresponds to a type identified with the `xsi:type` attribute:

```

<{base type} xsi:type="{derived type}">...</{base type}>

```

For information on derived types see the XML Schema documentation [W3C, 2001].

preLoad

Software developers can embed pre-defined XML mappings with the object type definition in an assembly. This allows the CLR to simply read the object type definition from its containing assembly, and register the recorded mappings. This can be done either in software (requiring the Extend Infrastructure permission), or by the `preLoad` element of the Remoting Services configuration (requiring the Enable Remoting Configuration permission). The `preLoad` element specifies the object type by giving the type name using the `type` attribute and the assembly in which it is defined using the `assembly` attribute:

```

<preLoad type="My.Types.Structure" assembly="MyTypes, Version=1.0.0.0,
    Culture=neutral, PublicKeyToken=ba342a3fdd33d701" />

```

When an assembly contains a number of object type definitions that have pre-defined XML-element-to-object-type mappings, they can all be registered at once by the CLR using the `preLoad` element without specifying each individual type.

```

<preLoad assembly="MyTypes, Version=1.0.0.0, Culture=neutral,
    PublicKeyToken=ba342a3fdd33d701" />

```

This is equivalent to having a set of `preLoad` elements for each specific object type defined in the given assembly.

Runtime Security Policy

The Common Language Runtime depends on the Code Access Security (CAS) policy mechanism to determine an assembly's permission grant. The CLR ships with a default CAS policy that can be modified by the administrator. There are three configurable policy levels: Enterprise, Machine, and User. The administrator may modify each policy level's Code Groups, Named Permission Sets, and Policy Assemblies. There is an additional CAS policy level associated with the Application Domain in which the assembly is loaded. The Application Domain level policy is not configurable by the administrator.

The CAS policy configuration for each level is stored in an XML file. `mscorcfg.msc` assists the administrator in modifying these XML files. Table 24 lists the location of each configuration file.

Policy Level	Configuration File	Location
Enterprise	<code>enterprisesec.config</code>	<code>%FrameworkDirectory%\<version>\CONFIG</code>
Machine	<code>security.config</code>	<code>%FrameworkDirectory%\<version>\CONFIG</code>
User	<code>security.config</code>	<code>%SystemDrive%\Documents And Settings\<username>\Application Data\Microsoft\CLR Security Config\<>version></username></code>

Table 24. CAS Policy File Locations.

Figure 62 shows the Runtime Security Policy node expanded. Under each policy level, the administrator can configure Code Groups, Named Permission Set, and Policy Assemblies.

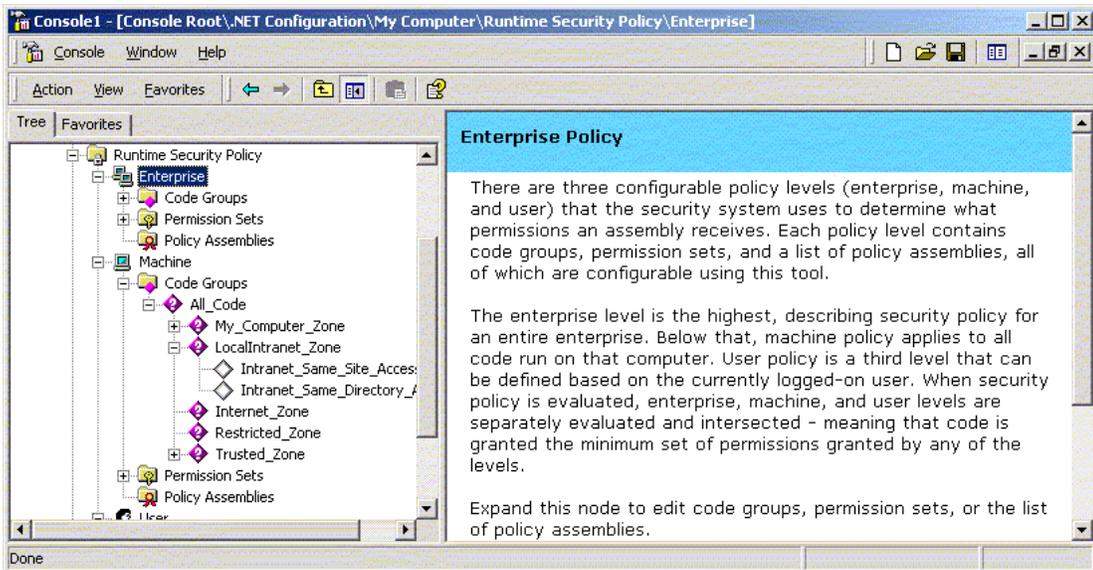


Figure 62. Runtime Security Policy Node.

- Code Groups – displays the Code Group tree for the corresponding policy level. Different icons identify different types of Code Groups:
 - ◆ The purple diamond represents a Union Code Group. Only Union Code Groups can be created using `mscorcfg.msc`. All other Code Group types must be created either programmatically or by manually editing the appropriate configuration file.
 - ◇ The white diamond icon represents any Code Group other than a Union Code Group. The default CAS policy includes some Code Groups that are not Union Code Groups.
- Permission Sets – lists all Named Permission Sets defined for a particular policy level. Named Permission Sets offer an easy way to assign permissions as a group.
- Policy Assemblies – lists all assemblies used for evaluating policy for the selected Policy Level. This list may contain custom software libraries that define new types of Code Groups, Membership Conditions, or resource permissions.

Runtime Security Policy tasks

The following tasks can be performed under the Runtime Security Policy node:

- Create a CAS policy deployment package
- Reset all CAS policy levels to default settings
- View Code Groups

- Add or remove a Code Group
- Rename a Code Group
- Set or clear the Exclusive or Level Final attribute of a Code Group
- Change a Code Group's Membership Condition
- Change a Code Group's associated Named Permission Set
- Adjust Zone Security
- View Named Permission Sets
- Add or remove a Named Permission Set
- Modify a Named Permission Set
- View Policy Assemblies
- Enroll or withdraw a Policy Assembly
- List Code Groups to which an assembly belongs
- View an assembly's Allowed Permission Set
- Create a tailored Code Group
- Use the Trust an Assembly Wizard

Create a CAS policy deployment package

CAS policy deployment via Group Policy is discussed in more detail in chapter 3. To create a Windows Installer package for a specified CAS policy level of the current version of the .NET Framework, perform the following steps. These summarize the more detailed discussion in chapter 3.

- Select the **Runtime Security Policy** node in the console tree.
- Select **Create Deployment Package** in the tasks pane on the right to open the Deployment Package Wizard.
- Select a policy level to deploy by selecting one of the radio buttons: **Enterprise**, **Machine**, or **User**.
- Enter a folder and file name (or browse to a location) for the new Windows Installer package.

- Select **Next** and then **Finish** and the Wizard will create the package in the specified location.

Reset all CAS policy levels to default settings

Resetting all policy levels will return CAS policy to its default configuration. All administratively-defined Code Groups and Named Permission Sets will be lost.

- Select the **Runtime Security Policy** node.
- Right-click and select **Reset All...** from the context menu or click **Reset All Policy Levels** in the Help Topic pane on the right.
- Select **Yes**.

View Code Groups

To view the Code Group tree for a policy level, simply expand the Code Groups node under the Runtime Security Policy node in the console tree.

Add or remove a Code Group

To add a Code Group, perform the following steps:

- In the left hand pane, expand the **Code Groups** node under the policy level in which the new Code Group will be created.
- Expand nodes in the Code Group tree as needed to select the parent Code Group for the new Code Group. The **All_Code** Code Group is the root of the Code Group tree.
- Click **Add a Child Code Group** in the task pane on the right

or

Right-click the parent Code Group and select **New...** from the context menu

or

Select **Action, New...** from the menu bar.

At this point, the task can be performed interactively through the Create Code Group dialog box (see Figure 63), or by importing a Code Group definition from an XML file. These two methods are described separately below.

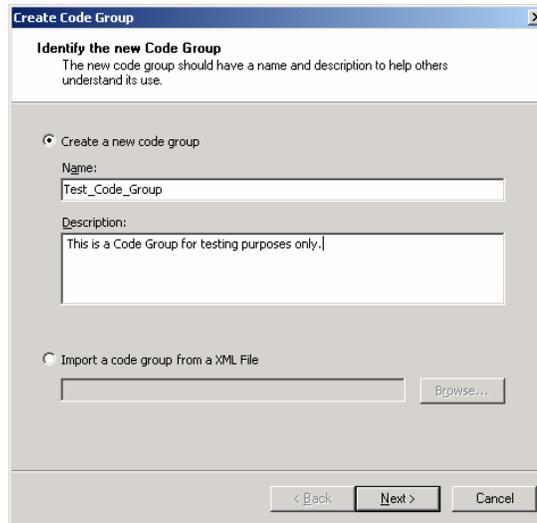


Figure 63. Create Code Group Dialog Box.

Interactively defining a new Code Group

- Select the **Create a new code group** radio button. This is the default selection.
- Enter a name and description for the new Code Group and click **Next**.
- Choose the Membership Condition for the custom Code Group and click **Next**. The administrator may choose among the default Membership Conditions shipped with the .NET Framework or import a custom Membership Condition defined in an XML file. The XML file should contain only an <IMembershipCondition> element, as in Figure 64 below:

```
<IMembershipCondition
  class="AllMembershipCondition, mscorlib, Version=1.0.5000.0,
    Culture=neutral, PublicKeyToken=b77a5c561934e089"
  version="1"/>
```

Figure 64. Example of a Membership Condition To Be Imported.

In the .NET Framework CAS policy files, the `class` attribute of an <IMembershipCondition> element may refer to a short name defined earlier in a <SecurityClass> element, but in the imported XML, the full class name must be used.

- Assign a Permission Set to the Code Group. The administrator has two options:
 - Use existing permission set** – select a Named Permission Set that has already been defined. This includes Named Permission Sets previously created by the administrator. See the Add or remove a Named Permission Set task for details about creating new Named Permission Sets.

Create a new permission set – use the Create New Permission Set dialog box to interactively select permissions to include in a new Named Permission Set or import a Named Permission Set definition from an XML file. The use of this dialog box is discussed below in the Add or remove a Named Permission Set task.

- Select **Next** and then **Finish**.

Importing a Code Group definition from an XML file

- Select the **Import a code group from a XML file** radio button.
- Enter the complete path to the XML file in the text box, or click Browse and select the file through the Import XML file dialog box.
- Click **Finish**. A message box will pop up if there are any problems importing the XML. The following XML shows a valid Code Group definition (Figure 65):

```
<CodeGroup
  class="System.Security.Policy.UnionCodeGroup, mscorlib,
    Version=1.0.5000.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089"
  version="1"
  Name="testxml"
  Description="testxmldescription">
  <IMembershipCondition
    class="System.Security.Policy.AllMembershipCondition,
    mscorlib, Version=1.0.5000.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089"
    version="1"/>
  <PermissionSet
    class="NamedPermissionSet"
    version="1"
    Name="Execution1"
    Description="Permits execution"/>
</CodeGroup>
```

Figure 65. Example of a Code Group To Be Imported.

In the .NET Framework CAS policy files, the `class` attribute of a `<CodeGroup>` or `<IMembershipCondition>` element may refer to a short name defined earlier in a `<SecurityClass>` element, but in the imported XML, the full class name must be used. The `PermissionSetName` attribute of a `<CodeGroup>` element should not be used. Instead, include a `<PermissionSet>` child element with the desired `Name` attribute. The name used may refer to a pre-defined Named Permission Set, or it could be a new set. A new

set must be created as a separate task – the import process for a Code Group will not create it if it does not exist. In any case, any permissions included as child `<IPermission>` elements of the `<PermissionSet>` element will be ignored.

To remove a Code Group:

- In the left hand pane, expand the **Code Groups** node under the policy level in which the Code Group will be removed.
- Expand nodes in the Code Group tree as needed to select the targeted Code Group.
- Right-click the targeted Code Group and select **Delete** from the context menu

or

Select the targeted Code Group and select **Action, Delete** from the menu bar.

Rename a Code Group

To rename a Code Group, perform the following steps:

- In the left hand pane, expand the **Code Groups** node under the policy level that contains the targeted Code Group.
- Expand nodes in the Code Group tree as needed to select the targeted Code Group.
- Click **Edit Code Group Properties** in the task pane on the right to display the Properties dialog box. The name and description of the Code Group can be changed through the first panel of this dialog box. Click **Apply** to save the changes.

or

Right-click the Code Group and select **Rename** from the context menu. This allows editing of the Code Group name in the Code Group tree.

or

Select the targeted Code Group and select **Action, Rename** from the menu bar. This allows editing of the Code Group name in the Code Group tree.

Set or clear the Exclusive or Level Final attribute of a Code Group

If an assembly will satisfy the Membership Conditions of two or more Code Groups marked Exclusive, the CLR will report an error, as it cannot unambiguously determine which permissions to grant. `mscorcfg.msc` will also report an error if the administrator attempts to set the Exclusive attribute through `mscorcfg.msc` on two or more Code Groups to which an assembly may belong. This determination is not perfect – `mscorcfg.msc` may allow multiple Exclusive Code Groups whose Membership Conditions are later satisfied by the

right combination of evidence. Thus, the Exclusive attribute should be set on Code Groups with narrowly tailored Membership Conditions.

To set Code Group attributes:

- In the left hand pane, expand the **Code Groups** node under the policy level that contains the targeted Code Group.
- Expand nodes in the Code Group tree as needed to select the targeted Code Group.
- Click **Edit Code Group Properties** in the task pane on the right.

or

Right-click the Code Group and select **Properties** from the context menu.

or

Select the targeted Code Group and select **Action, Properties** from the menu bar.

- All three of the methods in the previous step will display the Properties dialog box (see Figure 66).

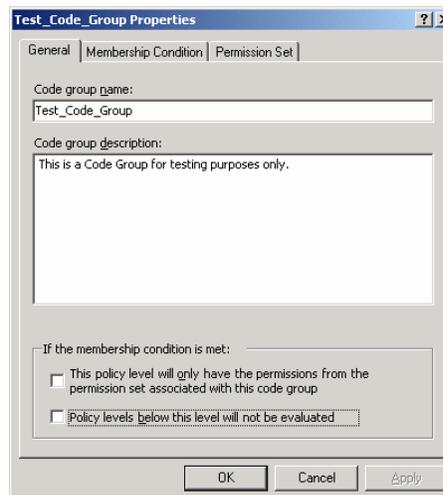


Figure 66. Code Group Properties Dialog Box: General Tab.

To set or clear the Exclusive attribute, check or uncheck the first check box: **This policy level will only have the permissions from the permission set associated with this code group.**

To set or clear the Level Final attribute, check or uncheck the second check box: **Policy levels below this level will not be evaluated.**

- Click **OK** to apply the changes.

Change a Code Group's Membership Condition

To change a Code Group's Membership Condition, perform the following steps:

- In the left hand pane, expand the **Code Groups** node under the policy level that contains the targeted Code Group.
- Expand nodes in the Code Group tree as needed to select the targeted Code Group.
- Click **Edit Code Group Properties** in the task pane on the right.

or

Right-click the Code Group and select **Properties** from the context menu.

or

Select the targeted Code Group and select **Action, Properties** from the menu bar.

- All three of the methods in the previous step will display the Properties dialog box. Select the **Membership Condition** tab (see Figure 67).

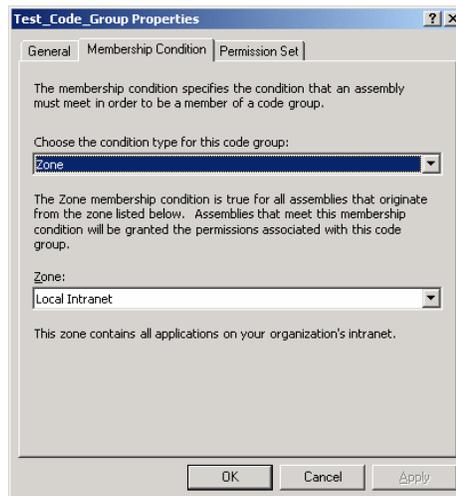


Figure 67. Code Group Properties Dialog Box: Membership Condition Tab.

- Select the Membership Condition for the Code Group from the pull-down menu. If additional parameters are necessary to define the Membership Condition, additional instructions will appear below the pull-down menu.
- Click **OK** to apply the change.

Change a Code Group's associated Named Permission Set

To change a Code Group's associated Named Permission Set, perform the following steps:

- In the left hand pane, expand the **Code Groups** node under the policy level that contains the targeted Code Group.
- Expand nodes in the Code Group tree as needed to select the targeted Code Group.
- Click **Edit Code Group Properties** in the task pane on the right.

or

Right-click the Code Group and select **Properties** from the context menu.

or

Select the targeted Code Group and select **Action, Properties** from the menu bar.

- All three of the methods in the previous step will display the Properties dialog box. Select the **Permission Set** tab (see Figure 68).

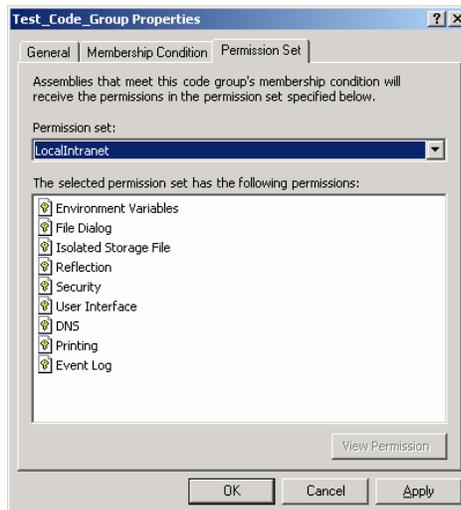


Figure 68. Code Group Properties Dialog Box: Permission Set Tab.

- Select the Named Permission Set to be associated with this Code Group from the pull-down menu. Only previously defined Named Permission Sets can be selected. To create a new Named Permission Set, see the Add or remove a Named Permission Set task below.
- Click **OK** to apply the change.

Adjust Zone Security

To modify the Named Permission Set associated with any of the built-in Zone-based Code Groups, perform the following steps:

- Select the **Runtime Security Policy** node in the console tree and click the **Adjust Zone Security** task in the detail pane on the right.

or

Right-click the **Runtime Security Policy** node in the console tree and select **Adjust Security...** from the context menu

or

Select the **Runtime Security Policy** node in the console tree and select **Action, Adjust Security...** from the pull-down menu.

- In the Security Adjustment Wizard (Figure 69), select the CAS policy level to modify and click **Next**. Selecting **Make changes to this computer** will modify the Machine level policy. Selecting **Make changes for the current user only** will modify the User level policy.

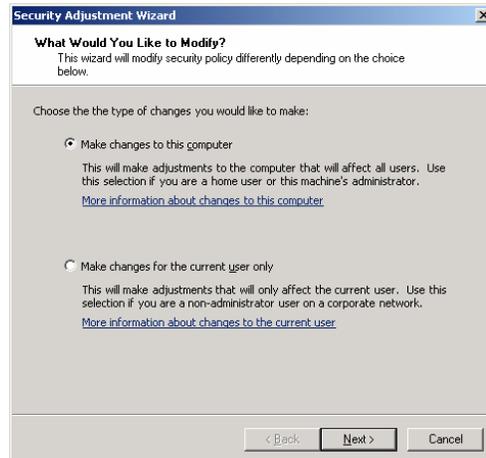


Figure 69. Security Adjustment Wizard.

- The Security Adjustment Wizard will display all the URL Security Zones with a slide control to set the “level of trust” associated with each zone (see Figure 70).



Figure 70. Adjusting the Security Level for Each Zone.

Each level of trust is mapped a built-in Named Permission Set, as shown in Table 25:

“Level of Trust”	Named Permission Set
Full Trust	FullTrust
Medium Trust	LocalIntranet
Low Trust	Internet
No Trust	Nothing

Table 25. Named Permission Sets Associated with “Levels of Trust.”

When configuring Machine level policy, the Security Adjustment Wizard will allow any “level of trust” to be associated with each zone. When configuring User level policy, the Wizard will restrict the maximum “level of trust” that can be associated with some zones.

- Make the desired modification and click **Next** to view a summary of the new settings.
- Click **Finish** to apply the changes.

View Named Permission Sets

To view the Named Permission Sets defined for a policy level, simply expand the Permission Sets node under the Runtime Security Policy node in the console tree.

Add or remove a Named Permission Set

To add a new Named Permission Set, perform the following steps:

- Select the **Permission Sets** node within a Policy Level.
- Right-click and select **New...** from the context menu.

or

Select **Action, New...** from the menu bar.

At this point, the task can be performed interactively through the Create Permission Set dialog box (see Figure 71), or by importing a Named Permission Set definition from an XML file. These two methods are described separately below.

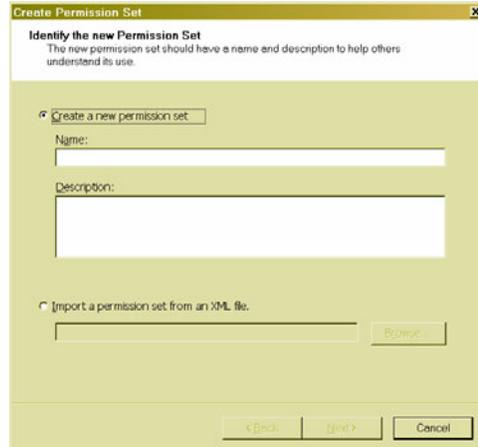


Figure 71. Create Permission Set Dialog Box.

Interactively defining a new Named Permission Set

- Select the **Create a new permission set** radio button. This is the default selection.
- Enter a name and description for the new Named Permission Set and click **Next**.
- Select the individual permissions to include in the new Named Permission Set (see Figure 72). Use the **Add >>** button to move permissions from the **Available Permissions** list to the **Assigned Permissions** list. This will add permissions to the Named Permission Set. Use the **<< Remove** button to remove permissions from the **Assigned Permissions** list.

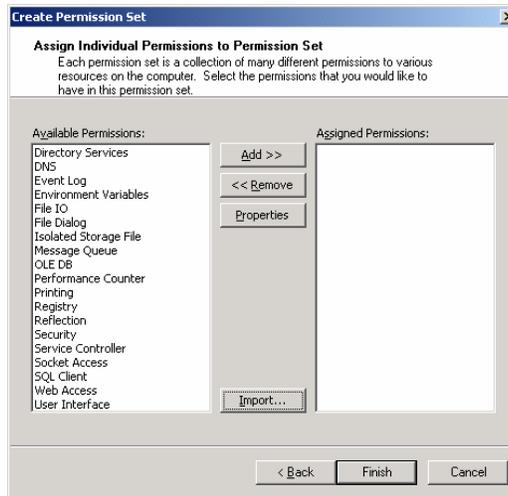


Figure 72. Assign Permissions to a Named Permission Set.

Individual permissions will have configurable settings that identify specific resources or access types. As each permission is assigned using the **Add >>** button, a Permission Settings dialog box will be displayed that allows some permission settings to be configured. These settings can also be changed later by selecting a permission in the **Assigned Permissions** list and clicking **Properties**. Each Permission Settings dialog box contains a pair of radio buttons:

Grant assemblies unrestricted access to <resource> – Selecting this radio button will allow full access to all instances of the protected resource. This level of access should be granted only where necessary in keeping with the principle of least privilege.

Grant assemblies access to <resource> – Access to resources is generally configured by identifying a specific resource of the type protected with the permission, and a type of access. If no resources are specified, no access will be permitted. That is, including a permission in a Named Permission Set without specifying any access will be the same as not including it – no access will be granted. The default setting when adding a new Named Permission Set is to restrict all access.

The DNS permission has no configurable access settings: access is either unrestricted or prohibited. In this case the radio button is labeled **Grant assemblies no access to DNS**.

To import the definition of a permission, click the **Import...** button and select an XML file in the Import a Permission dialog box. Figure 73 shows the contents of an XML file containing a permission:

```
<IPermission
```

```

class="System.Security.Permissions.FileDialogPermission, mscorlib,
    Version=1.0.5000.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089"
version="1"
Access="Open"/>

```

Figure 73. Example of a Permission To Be Imported.

In the .NET Framework CAS policy files, the `class` attribute of an `<IPermission>` element may refer to a short name defined earlier in a `<SecurityClass>` element, but in the imported XML, the full class name must be used.

- Select **Finish** once all desired permissions have been assigned to the new Named Permission Set and their access settings have been configured.

Importing a Named Permission Set definition from an XML file

- Select the **Import a permission set from an XML file** radio button.
- Enter the complete path to the XML file in the text box, or click Browse and select the file through the Import XML file dialog box.
- Click **Finish**. A message box will pop up if there are any problems importing the XML. The following XML shows a valid Named Permission Set definition (Figure 74):

```

<PermissionSet
  class="System.Security.NamedPermissionSet"
  version="1"
  Name="Internety"
  Description="Default rights given to internet applications">
  <IPermission
    class="System.Security.Permissions.FileDialogPermission,
    mscorlib, Version=1.0.5000.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089"
    version="1"
    Access="Open"/>
  <IPermission
    class="System.Security.Permissions.SecurityPermission,
    mscorlib, Version=1.0.5000.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089"
    version="1"

```

```
Flags="Execution"/>
</PermissionSet>
```

Figure 74. Example of a Named Permission Set To Be Imported.

In the .NET Framework CAS policy files, the `class` attribute of a `<PermissionSet>` or `<IPermission>` element may refer to a short name defined earlier in a `<SecurityClass>` element, but in the imported XML, the full class name must be used. If the name of the new Named Permission Set conflicts with an existing name, the name of the new Named Permission Set will be prefixed with “new”, “new (2)”, new (3)”, etc.

To remove a Named Permission Set:

- In the left hand pane, expand the **Permission Sets** node under the policy level in which the Named Permission Set will be removed.
- Right-click the targeted Named Permission Set and select **Delete** from the context menu

or

Select the targeted Named Permission Set and select **Action, Delete** from the menu bar.

Modify a Named Permission Set

Except for the Everything Named Permission Set, the built-in Named Permission Sets cannot be changed. To modify a Named Permission Set, perform the following steps:

- Expand the **Permission Sets** node within a Policy Level and select the targeted Named Permission Set.
- If the Help Topic is displayed in the tasks pane on the right, click **Change Permissions**.

or

Right-click and select **Change Permissions...** from the context menu.

or

Select **Action, Change Permissions...** from the menu bar.

- The Create Permission Set dialog box will be displayed. The permissions currently included in this Named Permission Set will be listed in the **Assigned Permissions** list on the right.

- Use the **Add >>** and **<< Remove** buttons to change the permissions included in the Named Permission Set as desired.

Use the **Properties** button to change access settings for individual permissions in the **Assigned Permissions** list.

- Select **Finish** to apply the changes.

View Policy Assemblies

To view the Policy Assemblies currently configured for a policy level, perform the following steps:

- Select the **Policy Assemblies** node within a policy level. If the Policy Assemblies view is currently set as the default, the list of Policy Assemblies will be displayed in the right hand tasks pane. Otherwise the Help Topic will be displayed and the following step should be performed:

- Select **View Policy Assemblies**.

or

Right-click the **Policy Assemblies** node and select **View | Assemblies** from the context menu.

or

Select **View, Assemblies** from the menu bar.

Enroll or withdraw a Policy Assembly

To enroll a Policy Assembly, perform the following steps:

- Select the **Policy Assemblies** node within a policy level.

- Right-click and select **Add...** from the context menu.

or

Select **Action, Add...** from the menu bar.

- In the **Choose Assembly from Assembly Cache** dialog box, select an assembly installed in the GAC, and click **Select**.

To withdraw a Policy Assembly, perform the following steps:

- View the Policy Assemblies configured for the desired policy level. See the View Policy Assemblies task for more details.

- Select the targeted assembly from the list of Policy Assemblies.
- Right-click the targeted assembly and select **Delete** from the context menu.

or

Select **Action, Delete** from the menu bar.

List Code Groups to which an assembly belongs

To list the Code Groups for which an assembly satisfies the corresponding Membership Condition, perform the following steps:

- Right-click the **Runtime Security Policy** node in the console tree and select **Evaluate Assembly...** from the context menu.

or

Select the **Runtime Security Policy** node and select **Action, Evaluate Assembly...** from the menu bar.

- In the Evaluate an Assembly dialog box (Figure 75), enter the URL of the assembly to be evaluated or click **Browse** to navigate the file system to select the file.

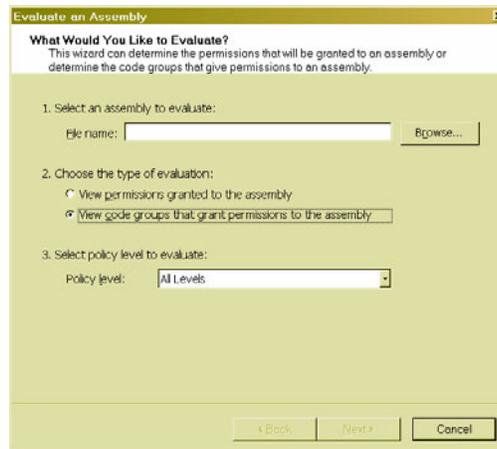


Figure 75. Evaluate an Assembly Dialog Box.

- Select the **View code groups that grant permissions to the assembly** radio button.
- Select the desired policy level, and click **Next**.

The Code Groups to which the selected assembly belongs will be displayed in the dialog box. If the specified assembly satisfies the Membership Condition of more than one Code Group marked Exclusive at any policy level, the task will fail. On the other hand, if the specified

assembly belongs to an Exclusive Code Group as well as other Code Groups, those other Code Groups are listed along with the Exclusive Code Group, even though they would not contribute permissions to the assembly's Allowed Permission Set (although they could still affect access if they were marked with the Level Final attribute). If the specified assembly satisfies the Membership Condition of a Code Group marked Level Final, then lower policy levels will not be evaluated.

View an assembly's Allowed Permission Set

An assembly's Allowed Permission Set may be different from the set of permissions actually granted to the assembly when it is executing (its Granted Permission Set). The Allowed Permission Set will take into account Exclusive or Level Final Code Groups, but will not include any Application Domain Policy, as this policy is determined at runtime. The Allowed Permission Set also does not include permission requests (Minimum, Optional, or Refused permissions) made by the assembly itself. Furthermore, the Allowed Permission Set is based on the evidence presented by the specified assembly, and the actual evidence presented by the assembly at runtime may be different if the assembly is obtained from a different source. To view an assembly's Allowed Permission Set, perform the following steps:

- Right-click the **Runtime Security Policy** node in the console tree and select **Evaluate Assembly...** from the context menu.

or

Select the **Runtime Security Policy** node and select **Action, Evaluate Assembly...** from the menu bar.

- In the Evaluate an Assembly dialog box (Figure 75), enter the URL of the assembly to be evaluated or click **Browse** to navigate the file system to select the file.
- Select the **View permissions granted to the assembly** radio button.
- Select the desired policy level, and click **Next**.

The assembly's Allowed Permission Set will be displayed in the dialog box, unless there were errors finding or evaluating the assembly. If the specified assembly satisfies the Membership Condition of more than one Code Group marked Exclusive at any policy level, the task will fail.

Create a tailored Code Group

To create a Code Group that will associate a particular set of permissions to a particular assembly, perform the following steps:

- Create a Named Permission Set that contains the desired permissions. See the Add or remove a Named Permission Set task above for details.

- Make a Code Group whose Membership Condition will discriminate between the targeted assembly and other assemblies. Set the Membership Condition to discriminate as little as possible commensurate with operational requirements and organizational policy. At the same time, insist on cryptography-based identities unless prohibited by policy constraints. Associate the new Named Permission Set with this Code Group. See the Add or remove a Code Group task above for details.

Use the Trust an Assembly Wizard

The Trust an Assembly Wizard can be used to modify the permissions associated with an assembly. This Wizard will create a Code Group with a Membership Condition tailored to some specific evidence associated with the assembly. New Code Groups created through the Trust an Assembly Wizard at the Machine level will have the Level Final attribute set. This will prevent User level policy settings from decreasing the permissions allowed by the Enterprise and Machine level policies.

The administrator can then select one of the built-in Named Permission Sets to associate with the Code Group. As in the Adjust Zone Security task above, the Named Permission Sets correspond to “levels of trust” (see Table 25).

Because the Allowed Permission Set is created by combining all the permissions for all Code Groups to which an assembly belongs, the Wizard will potentially add permissions to the Allowed Permission Set of the targeted assembly. To selectively restrict the permissions that would be granted to the assembly to the specified Named Permission Set, the Code Group created with the Wizard should be marked Exclusive. See the Set or clear the Exclusive or Level Final attribute of a Code Group task above for details.

- Select the **Runtime Security Policy** node in the console tree and click the **Increase Assembly Trust** task in the detail pane on the right.

or

Right-click the **Runtime Security Policy** node in the console tree and select **Trust Assembly...** from the context menu

or

Select the **Runtime Security Policy** node in the console tree and select **Action, Trust Assembly...** from the pull-down menu.

- In the Trust an Assembly Wizard (Figure 76), select the CAS policy level to modify and click **Next**. Selecting **Make changes to this computer** will modify the Machine level policy. Selecting **Make changes for the current user only** will modify the User level policy.

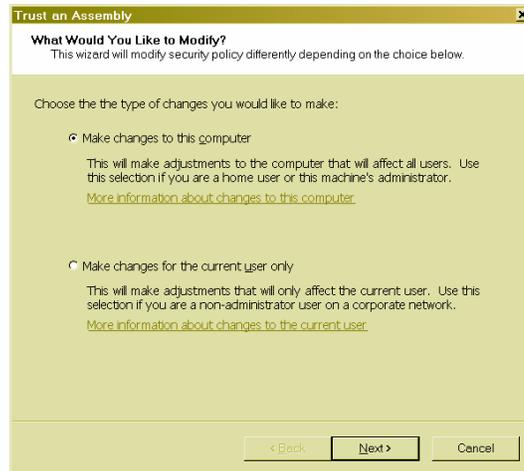


Figure 76. Trust an Assembly Wizard.

- Enter the path or URL of the targeted assembly, or click **Browse** to navigate to it. Select **Next** to continue.
- The Code Group created with this Wizard will use one of the cryptography-based identities associated with the assembly (see Figure 77). If the assembly is digitally signed with a software publisher's certificate, the Publisher Membership Condition will be available (select **All assemblies from the same publisher**). If the assembly is strong-named, the Strong Name Membership Condition will be available, with or without the version number (select **All assemblies with the same assembly public key** and check or clear the **Include version number** checkbox). In all cases, the SHA-1 hash of the assembly may be used as a Hash Membership Condition (select **This one assembly**). Select one radio button and click **Next** to set the Named Permission Set that will be associated with the new Code Group.

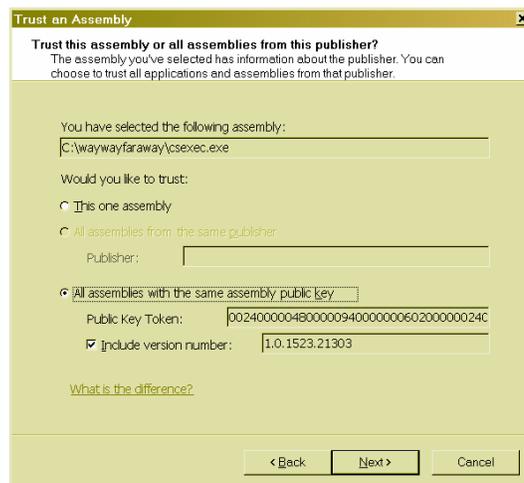


Figure 77. Membership Condition Selection in the Trust an Assembly Wizard.

- Set the Named Permission Set associated with the new Code Group by using the slide control to pick a “level of trust” (Figure 78):

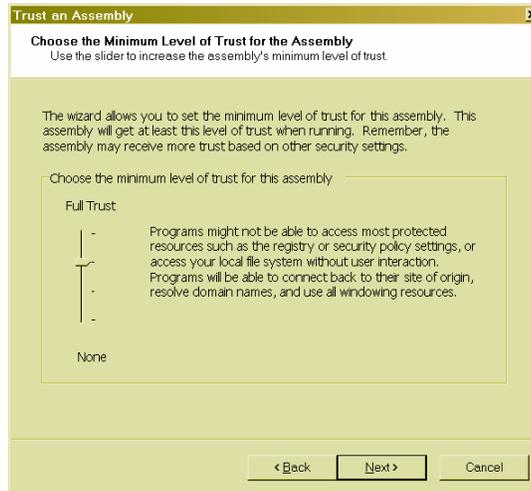


Figure 78. Named Permission Set Selection in the Trust an Assembly Wizard.

- Click **Next** to view a summary of the selected settings. Click **Finish** to apply the settings.

Applications

Several aspects of administering managed applications can be handled through the Applications node. These include settings that affect the way the application runs in conjunction with libraries that it needs to use or other applications with which it must communicate.

Applications Tasks

The following tasks can be performed under the Applications node:

- Add an application to be configured
- Configure application properties
- View assembly dependencies for an application
- View list of assemblies configured for an application
- Configure an assembly for an application
- Fix an application (roll back application Binding Policy)
- Configure Remoting Services for an application

Add an application to be configured

This task simply adds a specific managed application (identified by file location) as an entry under the Applications node. The configuration settings for an application are stored in an XML file in the same directory as the application executable file. Adding the application to the list of configurable applications will not create this XML file if it does not already exist. The configuration file will be created by `mscorcfg.msc` once a setting is changed. To add an application to the Applications node:

- Select the **Applications** node and click **Add an Application to Configure** in the details pane on the right.

or

Right-click the **Applications** node and select **Add...** from the context menu.

or

Select the Applications node and select Action, Add... from the menu bar.

- Select the application to configure from a list of managed applications that have been executed on the local computer, or select **Other...** to browse to the location of the executable to configure. Click **OK** to add the selected application as a child node under the **Applications** node.

Configure application properties

Garbage collection concurrency, application-wide publisher policy safe mode, and the application's probing path can be configured with this task. To configure these properties:

- Expand the **Applications** node and select the child node that represents the targeted application.
- Click **View the Application's Properties** in the details pane on the right.

or

Right-click on the application and select **Properties** from the context menu.

or

Select the application and select **Action, Properties** from the menu bar.

- Change the application properties using the Application Properties Dialog Box (Figure 79). The three properties configurable through this dialog box are discussed below. Click **OK** to apply the changes.

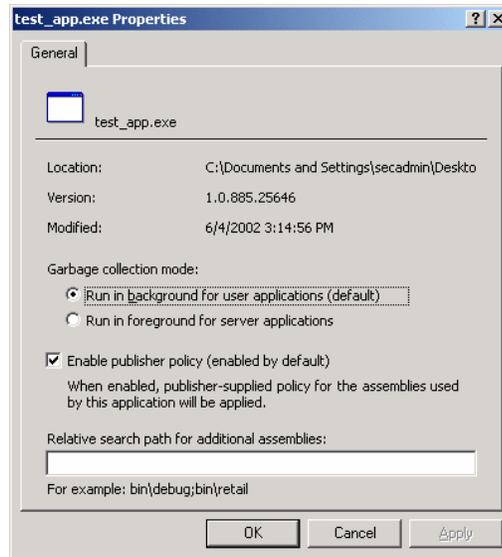


Figure 79. Application Properties Dialog Box.

Garbage Collection Concurrency

The CLR will periodically recover unused memory for reuse by the application. For applications whose user experience would be impaired by noticeable pauses to perform garbage collection, this process should be set to concurrent (“background”) mode. For processes that have limited user interfaces, are already written to do many things at the same time, or are often waiting for messages from other processes, garbage collection should be non-concurrent (“foreground”).

Publisher Policy Safe Mode

Publisher policies are supplied by software developers to deprecate old versions of libraries in favor of new versions that are claimed to be backward-compatible (see the discussion of Binding Policy in chapter 6 under Assembly Tasks). Since some applications depend on a specific version of a library and the claims of publishers may not be completely accurate, it may be necessary to ignore the version redirection rules supplied in the publisher policy. When publisher policy is disabled for all assemblies used by this application (“safe mode”), only the Binding Policy settings in the application and machine configuration files will be applied.

Probing Path

The probing path is a search path relative to the location (folder or URL) of this application that specifies the set of folders or resource paths that may contain private assemblies (i.e., not installed in the GAC) used by this application. When the CLR attempts to load an assembly referenced by this application it will check the folder or URL where the application is located, and then begin looking through the subfolders and resource subpaths specified by the probing path. The CLR will not search outside the folder or URL where the application is located.

View assembly dependencies for an application

To list the assemblies the application was explicitly developed to use:

- Expand the **Applications** node and select the child node that represents the targeted application.
- Click **View the Assembly Dependencies** in the details pane on the right.

or

Expand the node for the targeted application and select the **Assembly Dependencies** node. If the list of dependent assemblies is not already shown in the details pane on the right, select the **View Assembly Dependencies** task.

View list of assemblies configured for an application

When an application references one of its dependent assemblies, the CLR will attempt to find the correct version and location for the referenced assembly. The application's manifest contains the version of the assembly that it was developed to use, so this will be used by the CLR in the absence of any policy settings to the contrary. The CLR will also look for the assembly file using a set of rules that determine the default places to check.

When an alternate version (Binding Policy) or a specific location (CodeBase) is specified to constrain or redirect the CLR's search for the right assembly, the assembly is said to have been "configured" for this application. Technically, it is the application that has been configured, not the assembly. The policy settings will apply to the assembly only when it is invoked by the targeted application. In some cases an application will need to use an assembly that was not specified as a dependency when it was being developed, perhaps because the assembly is determined by user input or the output of another assembly. These assemblies may still be configured in anticipation of their future use by the application.

Configuring an assembly for an application is similar to configuring the assembly itself using the Configured Assemblies node of the console tree. See the Configure an assembly task above for more details. To view the list of assemblies that have been configured for an application,

- Expand the **Applications** node and select the child node that represents the targeted application.
- Select the **Managed Configured Assemblies** task in the details pane on the right and, if necessary, select View List of Configured Assemblies.

or

Expand the node for the targeted application, and select the **Configured Assemblies** node. If the list of assemblies configured for this application is not

already shown in the details pane on the right, select the **View List of Configured Assemblies** task.

Configure an assembly for an application

This task is used to set rules for an assembly that will apply when it is used by the selected application. The rules determine which version of an assembly will be used (Binding Policy) and the location from which the CLR will obtain a redirected assembly (CodeBase).

Before an assembly can be configured for an application, it must be added to the list of assemblies “configured” for the application. Once this is done, the Binding Policy and CodeBase settings may be configured through the Properties dialog box.

To add an assembly to the list of assemblies configured for a targeted application:

- Expand the **Applications** node and select the child node that represents the targeted application.
- Expand the node for the targeted application, right-click the **Configured Assemblies** node, and select Add... from the context menu.

or

Expand the node for the targeted application, select the **Configured Assemblies** node, and select **Action, Add...** from the menu bar.

- Select the assembly to configure using the Configure an Assembly dialog box (Figure 80) and click **Finish**. The assembly may be selected from a list of dependent assemblies for this application or from the GAC (by clicking **Choose Assembly...**), or the information can be entered manually.

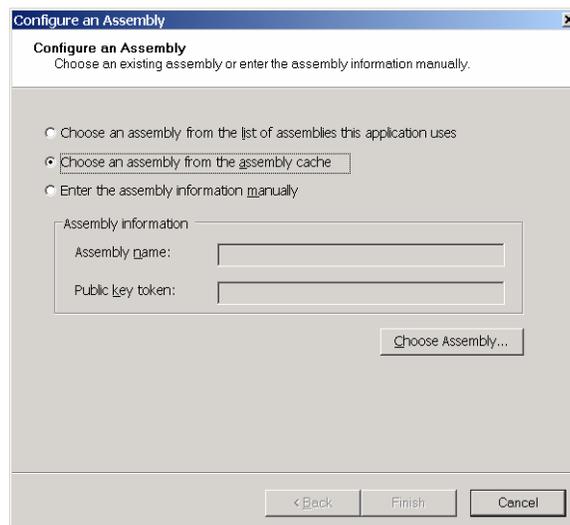


Figure 80. Configure an Assembly for an Application Dialog Box.

Once the assembly has been identified, the Properties dialog box is displayed, and the assembly may be immediately configured. See Using the Properties dialog box below for details.

To invoke the Properties dialog box for an assembly that is already on the list of assemblies configured for the targeted application:

- Display the list of assemblies configured for the targeted application. See the View list of assemblies configured for an application task for more details.
- Right-click on the desired assembly and select **Properties** from the context menu

or

Select the desired assembly and select **Action, Properties** from the menu bar.

Using the Properties Dialog Box

The Properties dialog box is used to set Binding Policy and CodeBases for the assembly when it is used by the targeted application. Configuring these properties is similar to configuring an assembly using the Configured Assemblies node above. See the Configure an assembly task above for details.

The configuration of an assembly for an application will be overridden by the machine assembly configuration for the given .NET Framework version of the CLR that the application uses. In addition, a publisher-supplied Binding Policy may also override these settings. For more information about the relationship between the application settings, the publisher-supplied Binding Policy, and the machine settings, see the discussion of Binding Policy in chapter 6 under Assembly Tasks.

To prevent a publisher-supplied Binding Policy from overriding the configuration of an assembly for an application, uncheck the **Enable publisher policy** check box (see Figure 81). The machine settings (see the Configure an Assembly task under the Configured Assemblies node above) will always override the application-specific configuration.

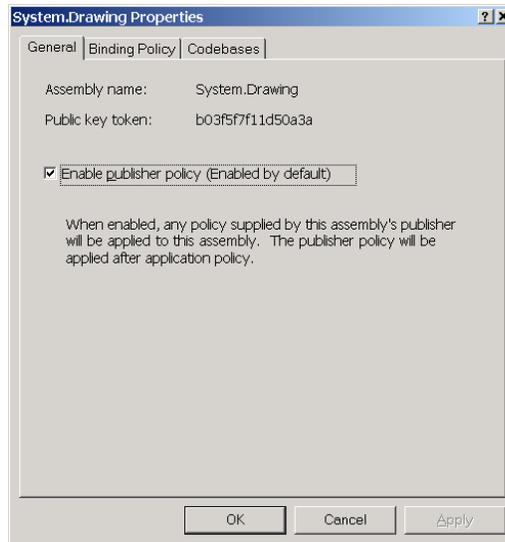


Figure 81. Enable Publisher Policy Check Box.

Fix an application (roll back application Binding Policy)

This task invokes the .NET Application Restore tool, which might not be available through `mscorcfg.msc` if the default installation was used. If unavailable through `mscorcfg.msc` (the links to Fix an Application don't work), run `configwizards.exe` in the same .NET Framework version directory as `mscorcfg.msc` and select Fix an Application from the ConfigWizards dialog box, as shown in Figure 82:



Figure 82. ConfigWizards Dialog Box.

An application must be executed at least once before it can be repaired. Applications may stop working properly if changes in binding policy cause it to use different libraries that don't provide the same functionality that the application is expecting. The restore tool will attempt to rollback binding policy changes to a restore point in an attempt to fix the problem. If the problem still persists, it may be because a new publisher policy is affecting the versions of dependent assemblies used. In this case, a safe mode restore point may be used to bypass publisher policy. To invoke the .NET Application Restore tool through `mscorcfg.msc`, perform the following steps:

- Expand the **Applications** node and select the child node that represents the targeted application.

- Click **Fix an Application** in the details pane on the right and select a managed application from the .NET Application Restore dialog box (Figure 83).

or

Select the targeted application and click **Fix this Application** in the details pane on the right.

or

Right-click the targeted application node and select **Fix Application...** from the context menu.

or

Select the targeted application node and select **Action, Fix Application...** from the menu bar.

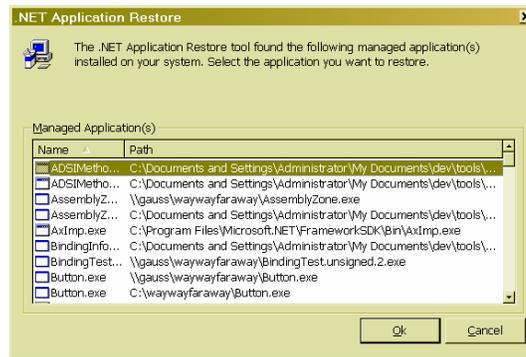


Figure 83. .NET Application Restore Dialog Box – Managed Applications.

The .NET Application Restore tool will present a list of binding policy restore points, plus a safe mode configuration that will simply disable any publisher policy for the current binding settings (Figure 84):

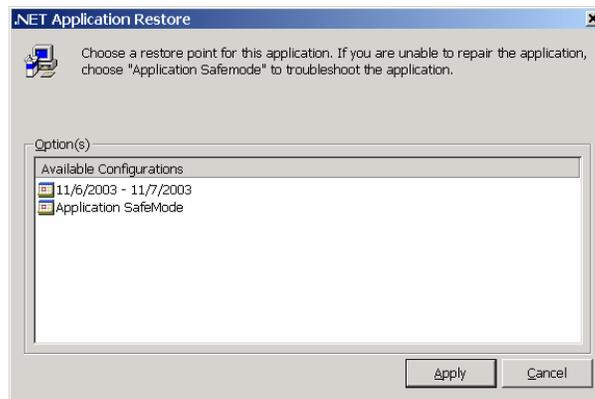


Figure 84. .NET Application Restore Dialog Box – Restore Points.

To attempt to fix the targeted application, choose a Binding Policy restore point for this application, identified by the date the previous Binding Policy was in effect, or choose **Application SafeMode** to disable any publisher-supplied policy. Click **Apply**, and test the changes.

If none of the restore points work, the problem may be with a machine Binding Policy for one of the dependent assemblies that is overriding the application and publisher policies. In this case, the application may have to be disabled until it is reengineered to function with different versions of the libraries. Machine policies based on identified security or stability flaws in libraries should not be discarded simply because applications require the flawed libraries to function.

Configure Remoting Services for an application

The .NET Framework Remoting system provides a means for applications to exchange data with and request services from other applications either on the same host or on a different host. To configure Remoting Services for an application,

- Add the application to the list of configured applications (see Configure an Assembly above).
- Select the application under the **Applications** node and click **Adjust Remoting Services** in the details pane on the right, or select the **Remoting Services** node under the application node and either click on **View Remoting Services Properties** in the details pane on the right or right-click the **Remoting Services** node and select **Properties**.

Configuring Remoting Services consists of

- Determining the type and properties of the communication channels that will be used. Channel settings may be configured on the **Channels** tab.
- Specifying the URLs of “known objects” (published names of software components) belonging to other applications with which the selected application will communicate. These URLs may be configured on the **Remote Applications** tab.
- Publishing internal software components as “known objects” that can be used by other applications. These components may be identified on the **Exposed Types** tab.

Some of these items can be configured through the Remoting Services Properties dialog box. All items can be configured by adding or modifying elements in an application’s configuration file or in the system-wide configuration file for each .NET Framework version, `machine.config`. For information on the XML structure of the Remoting Services configuration settings, see the discussion under the Remoting Services node above.

Summary

The .NET Framework Configuration Tool (`mscorcfg.msc`) is the central administrative tool for the .NET Framework. This tool automates the modification of some of the XML configuration files. Each version of the .NET Framework has its own version of `mscorcfg.msc` that is designed to configure that version only.

Recommendations in This Section

-  ***Recommendation: Create frequent backups of configuration files administered using `mscorcfg.msc`. This can be done by making a copy of the `CONFIG` folder for each installed version of the .NET Framework. For hosts with specialized policy-driven configurations, copies of these files should be stored away from the host to facilitate recovery and restoration of host operation.***

-  ***Recommendation: Rename any shortcut to `mscorcfg.msc` to reflect the version of the .NET Framework it is designed to configure. Example: “`mscorcfg v1.1.4322`”***

-  ***Recommendation: Use authentication and encryption for all remoting channels when `typeFilterLevel` attribute of the channel's formatter sink is set to “`full`”. Use authentication and encryption for all remoting channels in version 1.0 of the .NET Framework.***

Summary of Recommendations and Checklist

Summary of Recommendations

Table 26 lists all the recommendations made in this document in document order. The subject of the recommendation is in bold face.

Recommendation	Page
Chapter 2: Features of the .NET Framework Security Model	
File IO Permission: Only grant the File IO access permissions Read, Write, or Append to code that is trusted not to allow unauthorized access to file system resources. Grant File IO access to the most restrictive set of files and folders possible. Do not grant File IO access to file system roots or other broadly specified resources simply because they contain a few scattered files of interest. In many cases, the File Dialog or Isolated Storage File permissions are viable alternatives.	14
File Dialog Permission: Grant the File Dialog permission to code that needs user-discretionary access to files and folders. Use the File Dialog permission to allow the user rather than partially trusted code to browse the file system to the desired items. Where Append access is necessary or direct file system access cannot be allowed, the Isolated Storage File permission may be a viable alternative.	15
Isolated Storage Permission: Grant Administer Isolated Storage by User access only to highly trusted administrative tools. Grant Assembly Isolation by User/Roaming User access only to assemblies that need to use user-specific data applicable to many applications, and do not use application-specific data. Grant Domain Isolation by User/Roaming User access to all other assemblies. Note that this recommendation entails a separation of duties among assemblies: those that process data of common relevance to multiple applications should not also process application-specific data and vice versa.	16
User Interface Permission (Windowing): Code with limited trust should be granted at most Safe subwindows permission. Highly trusted code that accepts user authentication information or allows the user to authorize program actions through a graphical interface should be granted at most Safe top-level windows permission.	20

UNCLASSIFIED

Recommendation	Page
<p>User Interface Permission (Clipboard): The clipboard is a convenience for users who wish to change the presentation context of data or reuse data without retyping it into another application. It is a “broadcast” channel in that most software can programmatically read the contents of the clipboard and write data to it whether initiated by user input or not. Nevertheless, software should use other means to communicate and reserve the clipboard for discretionary use by the user. Read access (i.e., through the All Clipboard permission) should be reserved for highly trusted code.</p>	20
<p>Reflection Permission: Grant the Type Information permission only to highly trusted code that requires access to implementation details—typically this is restricted to software engineering tools or software interoperability services. Grant the Member Access permission only to highly trusted code.</p>	21
<p>Printing Permission: Grant All Printing permission only to highly trusted code.</p>	22
<p>DNS Permission: The DNS permission should typically be granted only to code that originates from within the local network (evidenced by a strong name with a public key associated with a local entity). Code from an external entity should be highly trusted to be granted this permission.</p>	23
<p>Socket Access Permission: The Socket Access permission should only be granted to highly trusted code or code that originates from the local network (evidenced by a strong name with a public key associated with a local entity) and provides networking services.</p>	23
<p>Web Access Permission: Grant the Web Access permission to a specified URL only to code that is denied access to information or resources that should not be shared with the remote site, or is trusted to protect resources that it can access. Unrestricted Web Access should only be granted to highly trusted code that performs networking services.</p>	24
<p>Message Queue Permission: The Message Queue permission should only be granted to code that originates from within the local network (evidenced by a strong name with a public key associated with a local entity) or from a highly trusted external entity. Administer access to any queue and Browse access to all queues should only be granted to highly trusted administrative tools.</p>	25
<p>Service Controller Permission: Grant the Service Controller permission for a Windows service only to assemblies whose trust is as high as the service itself and commensurate with the value of the availability of the service.</p>	25
<p>Database Permission: Grant any of the database permissions only to assemblies that are highly trusted.</p>	33
<p>Security Permission (Extend Infrastructure): Grant the Extend Infrastructure permission only to code that is trusted to have complete control over message processing.</p>	35
<p>Security Permission (Enable Remoting Configuration): The Enable Remoting Configuration permission should be granted only to software from a highly trusted source. The same considerations apply that would govern the granting of Unrestricted Web access or Unrestricted network socket access.</p>	35

UNCLASSIFIED

Recommendation	Page
Security Permission (Enable Serialization Formatter): Grant Enable Serialization Formatter permission only to highly trusted code that will be considered an extension to the CLR's trusted library base.	36
Security Permission (Enable Thread Control): Grant Enable Thread Control permission only to Fully Trusted code.	36
Security Permission (Allow Principal Control): Grant the Allow Principal Control permission only to code that is trusted at least as much as the most trusted user account on the system.	37
Security Permission (Enable Assembly Execution): Enable Assembly Execution should be granted based on the level of trust associated with its origin, as established by evidence stronger than URL Security Zone. If possible, separate the Execution permission from resource access permissions, so that the former is tied to origin and embodies a trust relationship, while the latter are tied to functional requirements of code and embody the principle of least privilege.	38
Security Permission (Skip Verification): Skip Verification should be granted only to highly trusted code based on a hash identity or strong name evidence that includes the assembly's name, version, and public key associated to a trusted party. If possible, separate the Skip Verification permission from resource access permissions, so that the former is tied to a specific assembly from a trusted point of origin and embodies a trust relationship, while the latter are tied to functional requirements of code and embody the principle of least privilege.	39
Security Permission (Allow Calls to Unmanaged Assemblies): Allow Calls to Unmanaged Assemblies should be granted only to code that is trusted to execute with the same privileges as the CLR itself.	40
Security Permission (Allow Policy Control): The Allow Policy Control permission should be granted only to highly trusted .NET Framework administrative tools.	40
Security Permission (Allow Domain Policy Control): If custom Runtime Host applications are in use that implement organizational policy using the AppDomain CAS policy level, then the Allow Domain Policy Control permission should be granted only to code that is highly trusted. In other cases (including the typical default installation), this permission should be granted only to code that is designed to dynamically launch other applications that may be less trusted than itself.	41
Security Permission (Allow Evidence Control): The Allow Evidence Control permission should be granted only to code developed by trusted parties with demonstrated secure coding practices. Code granted this permission effectively becomes an extension of the CLR's access control system.	41
Security Permission (Assert any Permission that Has Been Granted): The Assert any Permission that Has Been Granted permission should be granted only to software that is from a trusted developer with demonstrated secure coding practices. Typically, this permission is granted to highly trusted extensions to the CLR base libraries.	42

Recommendation	Page
Performance Counter Permission: Grant Performance Counter access to the most restrictive set of performance counter categories possible. Grant Instrument or Administer access only to trusted code that provides or administers a monitoring service.	44
Environment Permission: The Environment permission with Unrestricted access should be granted only to highly trusted code.	48
Event Log Permission: The Event Log permission with Audit or Unrestricted access should be granted only to administrative tools from trusted developers that monitor system and application events.	49
Registry Permission: Grant the Registry permission with the most restrictive access type and to the most restrictive set of registry keys possible.	49
Directory Services Permission: Grant the Directory Services permission with the most restrictive access type and to the most restrictive set of directory node paths possible. Grant Browse access to the Windows system directory services (Active Directory/Global Catalog, IIS Metabase) only to code of local origin (evidenced by a strong name with a public key associated with a local entity). Only highly trusted administrative tools should be granted Write access to the Windows system directory services.	50
Strong Name Membership Condition: Strong name verification should never be simulated in an operational environment.	57
First Match Code Groups: Editing CAS policy files to use First Match Code Groups may create invalid or corrupt XML, as these files are also modified by automated tools and parsed by the CLR. Thus, it is recommended that CAS policy be configured using Union Code Groups configured through <code>mscorcfg.msc</code> .	59
File Code Groups, Net Code Groups: Editing CAS policy files to create File Code Groups or Net Code Groups may create invalid or corrupt XML, as these files are also modified by automated tools and parsed by the CLR. Thus, it is recommended that these groups be avoided or the XML of the default groups be copied and imported using <code>mscorcfg.msc</code> .	61
Assembly Permission Requests: Although software developers may and should support an organizational security policy through assembly permission requests, security policy should not rely on these requests, but should be implemented with CAS policy settings.	69
Level Final Code Group Attribute: Code Groups with the Level Final attribute should have Membership Conditions that are as narrowly defined as possible.	70
Chapter 3, Deploying .NET Framework CAS Policy Using Group Policy	
Adminstrating Deployment: Back up any custom host CAS policy using a Windows Installer package before configuring different policies for deployment.	74
Adminstrating Deployment: Configure policy for deployment on a protected host.	74

Recommendation	Page
Software Distribution Point: When using a shared network folder as a software distribution point for CAS policy, set the folder permissions to restrict access to administrators or others authorized to maintain .NET Framework CAS policy deployment files.	76
Group Policy Deployment: When CAS policy is deployed via Group Policy software installation, disable Fast Logon Optimization.	82
Administrating Deployment: Archive Windows Installer packages for all CAS policy deployments, including the default CAS policy, for use as restoration points when rollback to a prior policy state is desired.	88
Chapter 4, URL Security Zones and the .NET Framework Zone Membership Condition	
Zone Membership Condition: Do not grant or restrict access to resources based on a Zone Membership Condition in support of an organizational policy unless user mappings are disabled.	96
Zone Membership Condition: Only use Zone Membership Conditions as part of a multi-factor code authorization check that relies on at least one additional type of evidence before granting access to resources.	96
Chapter 5, Cryptographic Localization in the .NET Framework	
Cryptographic Configuration (Short Class Names): Use short class names as abbreviated forms of the algorithm classes. Do not use short class names to express policy-driven roles such as algorithm defaults or use conditions.	108
Cryptographic Configuration (Friendly Names): Use friendly names to express the policy-driven roles played by particular algorithm classes (identified by their short class names) in the local execution environment. These roles can be defaults (i.e., “DefaultHashAlgorithm”) or use conditions (i.e., “FinancialDataEncryption”).	109
Chapter 6, Administrative Task and Tools	
Administering CAS Policy: Never disable CAS policy on a computer connected to an untrusted network such as the Internet.	121
Administering the Windows Environment: Disable trust of Test Root certificates in an operational environment.	127
Administering the Windows Environment: Enable checking for expired certificates.	127
Publisher Membership Condition: Only use the Publisher Membership Condition for software publishers with whom your organization has a well-established history of trust. Access to resources may be granted to code that presents expired certificates, so the use of the Publisher Membership Condition assumes that the publisher was trustworthy in the past as well as the present.	128
Administering the Windows Environment: Enable checking for revoked certificates.	128

Recommendation	Page
Administering the Windows Environment: Disable automatic trust for certificates whose revocation status cannot be determined.	129
Administering the Windows Environment: Enable checking for revoked time stamp provider's certificate.	130
Strong Name Membership Condition: Base trust on a strong name only where the public key is verifiably associated with a trustworthy party, and the public key owner can be trusted to limit access to the corresponding private key.	132
Administering CAS Policy: Strong name verification should never be simulated in an operational environment.	133
Appendix A, Administrative Tools Reference	
Administering CAS Policy: Make Code Group names unique across the entire Code Group tree for any given CAS policy level.	145
Administering CAS Policy: Back up both the source and the target CAS policies at the Enterprise and Machine levels before running <code>migpol.exe</code> .	164
Administering CAS Policy: Review any policy migrated using <code>migpol.exe</code> to ensure that it conforms to organizational security policy.	165
Administering CAS Policy: Although an assembly may be safe even though it fails <code>peverify.exe</code> , do not allow unverifiable code to execute in an operational environment by granting the Skip Verification permission unless the code is from a highly trusted source.	167
Appendix B, <code>mscorcfg.msc</code> – The .NET Framework Configuration Tool	
Administering CAS Policy: Create frequent backups of configuration files administered using <code>mscorcfg.msc</code> . This can be done by making a copy of the <code>CONFIG</code> folder for each installed version of the .NET Framework. For hosts with specialized policy-driven configurations, copies of these files should be stored away from the host to facilitate recovery and restoration of host operation.	177
Administering CAS Policy: Rename any shortcut to <code>mscorcfg.msc</code> to reflect the version of the .NET Framework it is designed to configure. Example: " <code>mscorcfg v1.1.4322</code> ".	179
Remoting Services: Use authentication and encryption for all remoting channels when <code>typeFilterLevel</code> attribute of the channel's formatter sink is set to " <code>full</code> ". Use authentication and encryption for all remoting channels in version 1.0 of the .NET Framework.	202

Table 26. Summary of Recommendations.

CAS Policy Checklist

The following checklist is an aid to an administrator or an organization in applying the recommendations in this document to an operational host or network. The configuration of the .NET Framework begins well before the administrator begins to specify access control

settings in CAS policy for each host. In order to apply this checklist properly, the role of the .NET Framework in the overall system and network security architecture should be understood – is the .NET Framework merely a optional add-on to an existing comprehensive security plan, is it a critical control, or is it something in between? The design phase of an information system is the appropriate time to determine the role of the .NET Framework in the system security architecture and document that role in the organization’s security plan. This will direct the application of the guide’s recommendations toward meeting the specific goals of the organization for the .NET Framework as a security control.

An important consideration to keep in mind when including the .NET Framework in a security plan is that the CAS policy should be used to protect rather than to restrict the user. CAS policy protects the user from code by preventing code from performing unauthorized functions. Other controls should be employed to protect the system from the user by preventing the user from taking unauthorized actions.

First Steps to Configuring CAS Policy

Know how you will support the policy

Change is inevitable; changes in systems and operating environments will necessitate updates to the CAS policy. Plan for change from the beginning.

- The personnel responsible for approving, creating, reviewing, deploying, and maintaining the .NET Framework policies are identified and their roles delegated.
- A schedule for periodic review of CAS policy is established and documented.
 - This review will consider new software sources, reevaluate the levels of trust associated with software sources, and reevaluate the privilege associated with each protected resource.
 - Policy must be reviewed whenever the system or its operating environment changes. Policy should also be reviewed at periodic intervals whether there are known system changes or not.
- A process is in place for the security administrator responsible for reviewing and updating CAS policy to be made aware of changes to the operating environment of a host.
- A deployment method for CAS policy is established.
- A system for CAS policy archival is established to enable security incident review, operations recovery, and rollback of policy to prior states.

An organization can better take advantage of the granularity of CAS policy by adopting a software development methodology that creates modular applications. The following practices support a granular access control policy:

- Each assembly performs only related functions. Small, functionally focused assemblies lend themselves to more tailored access control decisions.
- All assemblies are strong named and access to the private keys is protected.
- Different strong names or publisher certificates are used to discriminate between groups of assemblies that are likely to be associated with different trust levels. For example, office automation applications should use a different strong name than code that provides networking or other infrastructure services.

Know what the policy needs to do for you

The value of using the .NET Framework CAS policy on a host cannot be estimated unless the risks that the policy is supposed to mitigate are known. Knowing the risk involves knowing the types of managed code that could run on the host, knowing the operating environment of the host, and knowing the resources that the policy will be designed to protect.

Know the code

The .NET Framework protects the user from threats posed by managed code of diverse characteristics and origin. Knowing the code involves identifying the range of code that could potentially execute on a host, and knowing the level of risk or, conversely, the level of assurance associated with various types of managed code.

- The strong names and publisher certificates authorized for use in access control decisions are known and documented, with documented trust assessments.
 - Trust assessments appropriately reflect knowledge of the documented private key management practices of internal and external entities, as available.
 - Trust assessments for strong names appropriately reflect the integrity of the channel used to obtain the strong name credentials.
- Standards for demonstrating secure coding practice are identified and documented. Software from developers that meet these standards may be granted more trust.
- The level of trust associated with managed code is identified and documented. The trust determination should:
 - include all managed code reachable from the host
 - incorporate the trust levels associated with the originator of the code, using the trust assessments associated with strong names, publisher certificates, and secure coding practices identified above

UNCLASSIFIED

- incorporate the trust levels associated with the sources from which code could be obtained and the security of the possible channels that could be used to transfer the code.
- assume that software will not limit its own access to resources (code should be assumed to be resource-greedy and unstable at best, and malicious at worst)

Know the environment

The .NET Framework is not a stand-alone security solution. It is layered on the operating system security and other features of the host and network. Knowing the base security environment is a necessary prelude to effectively integrating CAS policy into a layered security system.

- The operating environment of each host is known, including operating system security settings, network connectivity, and the secure and non-secure channels available.
- The operating system has been hardened appropriate to the operational use of each host, including the following items:
 - The .NET Framework folders are protected by NTFS file permissions to allow write access only to administrators.
 - All strong name simulation (“skip verification”) entries have been removed from the registry.
 - Certificate verification has been configured to check for expired and revoked certificates. The test root certificate authority is not trusted.

Know the resources

Sound access control decisions for managed code are based on the functions that are allowed or denied and the system assets that are protected by each .NET Framework permission. Knowing the protected resource includes knowing the privilege level of those functions and the value of those system assets.

- If access to a .NET Framework-protected resource by code should only be granted in the context of an additional security setting provided by the operating system (for example, a file permission), another part of the system (for example, a communications channel encryptor), or the resource itself (for example, a database application with integrated access control), then the required security context for the use of that resource is documented.
- The privilege associated with access to each resource on a host is identified and documented. Access privilege is a way of expressing the value of a system resource.

To access a resource, code must have a level of trust commensurate with the privilege associated with that resource. The privilege determination should:

- include all .NET Framework-protected resources
- be based on the range of functions that can be performed using that resource in the context of the most privileged user account, and the effects of those functions on other parts of the system
- reflect resource scarcity – access to a resource with a limited bandwidth may require higher privilege if contention for that resource would result in an unacceptable degradation of a system resource or capability
- be appropriate to the type of host on which the resource is located.

CAS Policy Creation

General Guidelines

CAS policies for different .NET Framework versions exist side-by-side and are administered separately. Policy must be configured, reviewed, maintained, and deployed separately for each version of the .NET Framework. Security configuration for the .NET Framework is not complete until every version of the .NET Framework has been appropriately configured.

CAS policy can easily become overly complex. The need to appropriately restrict code from a wide variety of sources is at odds with the need for a simple, modular design that facilitates regular review and update. CAS policy creation is an iterative process that should include frequent checks for possibilities to simplify. Over time, incremental updates caused by changes in the operating environment can result in an unwieldy and difficult-to-understand policy. A discipline of keeping simplicity and maintainability in mind enables the administration of the .NET Framework as a security control over the lifecycle of a host.

One aspect of CAS policy simplicity is the minimization of custom policy software components such as custom Code Group, Membership Condition, or Permission types. In general, system and network security architecture should rely on open standards as much as possible. Similarly, a CAS policy should be constructed from standard policy elements as much as possible, to make review and analysis of that policy easier.

- Custom Code Groups, Membership Conditions, and Permissions are evaluated, approved, and documented across the organization. The use of unapproved custom CAS policy elements has been prohibited by organizational security policy.

Policy Refinement Process

The following steps should be repeated until the access control policy is appropriately set for all code accessible from the host. Since code that is not granted permission to use resources is implicitly denied access to those resources, all CAS policies, including the default policy, will embody access control decisions for all code. CAS policy creation is therefore a process

of successive refinement of the existing policy to more appropriately reflect the access control needs of a particular operational environment.

1. Identify a category of software (“code group”) accessible from this host that can be defined by an available Membership Condition. Identify and document the trust associated with this software. This should correspond to the least trusted software that belongs to the identified category. For a discussion of the scope of each Membership Condition, see Table 16. Since categories may be nested, this process should begin with broadly defined Membership Conditions and progressively identify more specific groups of code.

Since access control in the .NET Framework is performed per assembly, CAS policy can be as granular as the modularity of an application – different parts of an application can be granted different access to resources. In particular, shared code components can be granted the least access they need to function in their limited roles. Conversely, large assemblies that access a broad set of resources should be associated with a higher level of trust.

2. Identify and document the authorized functions for the current category of software, if any.
3. Identify and document the .NET Framework-protected resources that need to be granted to perform the authorized functions, and the minimum type of access to those resources.
4. Identify the functions that the software could actually perform given access to the resources identified in the previous step. The set of resources needed for software to perform its authorized functions may allow it to perform unauthorized functions as well, as the following example illustrates.

		Functions				
		A	B	C	D	E
Resources	1	✓		✓		✓
	2	✓	✓		✓	✓
	3		✓	✓	✓	✓
	4				✓	✓
	5	✓	✓	✓	✓	

Figure 85. Example Relationship Between Functions and Resources.

Example (see Figure 85): Code authorized to perform functions A and C needs access to resources 1, 2, 3, and 5 to perform both functions. However, this combination of resource access also allows it to perform the unauthorized action B. The trust level of this code should be commensurate with both the privilege associated with resources 1, 2, 3, and 5, and the set of functions {A, B, C} that it can actually perform with these resources.

5. Create a Named Permission Set that contains only the access to resources appropriate for the current software category. The appropriate set of access permissions is determined by comparing the trust level of the software with the aggregate privilege level of the set of resources needed to perform its authorized functions. This aggregate privilege is based on the functions identified in the previous step and may be greater than the privilege of each individual resource. The trust level of the software should be commensurate with the functions it can actually perform, not just with the functions it is authorized to perform.

If an access to a resource is too highly privileged or allows the performance of an unauthorized function that is too highly privileged in comparison to the software's level of trust, it should not be included in the Named Permission Set. Thus, the Named Permission Set may allow the software to perform only a subset of its authorized functions.

6. Create a Code Group that has the specified Membership Condition and Named Permission Set. If the current software category is a subset of a previously defined category, the Code Group may be created as a child code group.
7. Review the resulting CAS policy for opportunities to simplify. It may be possible to combine similar Named Permission Sets without violating the principle of least privilege. It may be possible to remove permissions from a child Code Group that have already been granted through a parent Code Group.

CAS Policy Review

Once the CAS policy has been created, it should be reviewed using the following steps.

Host Policy Review

The same principles discussed above with respect to CAS policy creation should be considered as part of the policy review. The review process should evaluate how well the policy embodies those principles. A policy review is not complete until the CAS policy of every version of the .NET Framework has been examined.

- The recommendations in this configuration guide have been followed. To aid policy review, exceptions appropriate to the operating environment should have a documented justification.
- The policy is not unnecessarily complex.
- The use of custom Code Groups, Membership Conditions, and Permissions has been minimized.
- The policy adequately protects the most privileged user account under which it will be applied.

- The policy appropriately restricts all code reachable from the host.
- The policy implements the principle of least privilege by not granting access to any resource that is not required for code to perform its authorized functions.
- The policy does not grant code access to any resource that would permit it to perform a function it is not trusted to perform.
- The policy accurately applies the documented trust levels associated with code and privileges associated with system resources. Unlike policy creation, which focuses on a single Membership Condition, the policy review also focuses on the union of multiple Membership Conditions: given a resource, the trust level associated with the set of all code that is granted access to that resource must be commensurate with its associated privilege.

Security Architecture Review

- The composite security policy provided by the .NET Framework and the operating system is appropriate for the operating environment of each host.
- The deployment of different CAS policies to different hosts in a network is coordinated and consistent with the operating environment of different network segments. Perimeter hosts should have very restrictive CAS policies. The security of a network is the security of its weakest link. A well thought-out deployment of CAS policy avoids weak links: hosts with poor or overly permissive CAS policies that create avenues of attack.
- The overall deployment of the .NET Framework is as simple as possible. The number of host-specific CAS policies is minimized.
- The CAS policy is in conformance with any organizational security policy regarding the configuration, distribution, or execution of software, including executable content (code bundled with data that executes as a side effect of accessing or viewing the data) and other forms of mobile code (all code obtained from remote systems and executed locally as needed).

Summary

Because the security configuration of the .NET Framework depends on the interplay between the operating system security settings, the operating environment of the host, and the nature of the software that is required or available for execution on the host, it does not lend itself naturally to a step-by-step procedure. Configuration is a cycle of successive refinement, beginning with a default policy. The outcome should not only be tailored policies for each host, but a set of new base policies that can make configuration of additional hosts or networks faster and more consistent with the overall security architecture.

This page has been intentionally left blank.

Appendix

D

Works Cited

- [ECMA-335, 2002] ECMA International. *Common Language Infrastructure (CLI)*, Standard ECMA-335, Second Edition. ECMA International, 2002.
<http://www.ecma-international.org/publications/files/ecma-st/ecma-335.pdf>
- [Barker (NIST SP 800-59), 2003] Barker, William C., *Guideline for Identifying an Information System as a National Security System*, NIST Special Publication 800-59, Gaithersburg, MD: National Institute of Standards and Technology, 2003.
<http://csrc.nist.gov/publications/nistpubs/800-59/sp800-59.pdf>
- [BS/ISO/IEC 17799, 2000] British Standards Institution, *Information Technology – Code of Practice for Information Security Management*, BS ISO/IEC 17799:2000.
<http://www.bspl.com/iso17799software>
- [FISMA, 2002] *Federal Information Security Management Act of 2002* (Title III of the E-Government Act of 2002, H.R. 2458), Public Law 107-347, Title III, 2002.
http://www.cio.gov/documents/e_gov_act_2002.pdf
- [Fraser (RFC 2196), 1997] Fraser, Barbara Y., Ed., *Site Security Handbook*, RFC 2196, Software Engineering Institute, Carnegie Mellon University, September, 1997.
<http://www.ietf.org/rfc/rfc2196.txt>
- [GAO/AIMD-12.19.6, 1999] Government Accounting Office, *Federal Information System Controls Audit Manual*, GAO/AIMD-12.19.6, 1999.
http://www.gao.gov/policy/12_19_6.pdf
- [GASSP, 1999] The International Information Security Foundation (I²SF)-Sponsored Committee to Develop and Promulgate Generally Accepted System Security Principles (GASSPC), *Generally Accepted System Security Principles*, Version 2.0, International Information Security Foundation, 1999.
<http://web.mit.edu/security/www/gassp1.html>
- [Haney, 2001] Haney, Julie M., *Guide to Securing Microsoft Windows 2000 Group Policy*, National Security Agency, 2001.
<http://www.nsa.gov>
- [LaMacchia, et al., 2002] LaMacchia, Brian A., Sebastian Lange, Matthew Lyons, Rudi Martin, and Kevin T. Price, *.NET Framework Security*, Boston: Addison-Wesley, 2002.

[Meier, et al., 2003] Meier, J. D., Alex Mackman, Michael Dunner, Srinath Vasireddy, Ray Escamilla, and Anandha Murukan, *Checklist: Security Review for Managed Code*.
http://msdn.microsoft.com/library/default.asp?url=/library/┘en-us/dnnetsec/html/CL_SecRevi.asp

[Microsoft, MSDN] The Microsoft Developer Network.
<http://msdn.microsoft.com>

[Microsoft, .NET Framework] Microsoft .NET Framework Developer Center.
<http://msdn.microsoft.com/netframework>

[Microsoft, 2002] Microsoft Corporation, *Secure Coding Guidelines for the .NET Framework*.
<http://msdn.microsoft.com/library/default.asp?url=/library/┘en-us/dnnetset/html/seccodeguide.asp>

[Sanderson and Rice, 2000] Sanderson, Mark J. and David C. Rice, *Guide to Securing Microsoft Windows 2000 Active Directory*, National Security Agency, 2000.
<http://www.nsa.gov>

[Swanson and Guttman (NIST SP 800-14), 1996] Swanson, Marianne and Barbara Guttman, *Generally Accepted Principles and Practices for Securing Information Technology Systems*, NIST Special Publication 800-14, Gaithersburg, MD: National Institute of Standards and Technology, 1996.
<http://csrc.nist.gov/publications/nistpubs/800-14/800-14.pdf>

[Swanson (NIST SP 800-18), 1998] Swanson, Marianne, *Guide for Developing Security Plans for Information Technology Systems*, NIST Special Publication 800-18, Gaithersburg, MD: National Institutes of Standards and Technology, 1998.
<http://csrc.nist.gov/publications/nistpubs/800-18/planguide.pdf>

[Weise and Martin, 2001] Weise, Joel and Charles R. Martin, *Developing a Security Policy*, Sun BluePrints OnLine, Santa Clara, Calif.: Sun Microsystems, December, 2001.
<http://www.sun.com/solutions/blueprints/1201/secpolicy.pdf>

[Watkins and Lange, 2002] Watkins, Demien and Sebastian Lange, *An Overview of Security in the .NET Framework*.
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/┘dnnetsec/html/netframesecover.asp>

[W3C, 2001] David C. Fallside, Ed., *XML Schema Part 0: Primer: W3C Recommendation, 2 May 2001*.
<http://www.w3.org/TR/xmlschema-0/>