Security and the Frameworks

Taken from the web by sldeskins to provide structured reading

[Securing Applications 12](#_Toc426471852)

[Security in the .NET Framework 14](#_Toc426471853)

[Security in the .NET Framework 15](#_Toc426471854)

[Security in the .NET Framework 16](#_Toc426471855)

[Security Changes in the .NET Framework 4 18](#_Toc426471856)

[Summary of Changes in Code Access Security 22](#_Toc426471857)

[Security Changes in the .NET Framework 25](#_Toc426471858)

[Security How-to Topics 25](#_Toc426471859)

[Security How-to Topics 26](#_Toc426471860)

[Security How-to Topics 27](#_Toc426471861)

[Key Security Concepts 27](#_Toc426471862)

[Permissions .NET Framework 1.1, 2.0, 3.0 29](#_Toc426471863)

[Security Permissions 30](#_Toc426471864)

[.NET Framework 3.5 30](#_Toc426471865)

[Security Permissions 31](#_Toc426471866)

[Code Access Permissions 33](#_Toc426471867)

[Identity Permissions 36](#_Toc426471868)

[Role-Based Security Permissions 37](#_Toc426471869)

[PrincipalPermission Class 37](#_Toc426471870)

[Type Safety and Security 41](#_Toc426471871)

[Security Policy .NET Framework 1.1, 2.0, 3.0, 3.5 41](#_Toc426471872)

[.NET Framework Security Policy .NET Framework 4 42](#_Toc426471873)

[Principal 42](#_Toc426471874)

[Authentication 42](#_Toc426471875)

[Authentication 43](#_Toc426471876)

[Authentication 44](#_Toc426471877)

[Authorization 46](#_Toc426471878)

[Security Concerns for Internal Virtual and Overloads Overridable Friend Keywords 47](#_Toc426471879)

[Key Security Concepts 48](#_Toc426471880)

[Security permissions 48](#_Toc426471881)

[**Code access permissions** 49](#_Toc426471882)

[**Role-based security permissions** 49](#_Toc426471883)

[Type safety and security 50](#_Toc426471884)

[Principal 50](#_Toc426471885)

[Authentication 50](#_Toc426471886)

[**Example** 51](#_Toc426471887)

[Authorization 52](#_Toc426471888)

[Security concerns for internal virtual keywords 53](#_Toc426471889)

[Code Access Security 54](#_Toc426471890)

[Code Access Security 56](#_Toc426471891)

[Introduction to Code Access Security 59](#_Toc426471892)

[Code Access Security Basics 61](#_Toc426471893)

[Code Access Security Basics 62](#_Toc426471894)

[Code Access Security Basics 62](#_Toc426471895)

[Writing Verifiably Type-Safe Code 66](#_Toc426471896)

[Writing Verifiably Type-Safe Code 67](#_Toc426471897)

[Security Syntax 67](#_Toc426471898)

[Declarative Security 68](#_Toc426471899)

[Extending Metadata Using Attributes 69](#_Toc426471900)

[Extending Metadata Using Attributes 69](#_Toc426471901)

[Extending Metadata Using Attributes 69](#_Toc426471902)

[Metadata and Self-Describing Components 70](#_Toc426471903)

[SecurityAction Enumeration 70](#_Toc426471904)

[Imperative Security 71](#_Toc426471905)

[Requesting Permissions 72](#_Toc426471906)

[Requesting Permissions 75](#_Toc426471907)

[Requesting Minimum Permissions 78](#_Toc426471908)

[How to: Request Permission to Access Unmanaged Code 78](#_Toc426471909)

[How to: Request Minimum Permissions by Using the RequestMinimum Flag 79](#_Toc426471910)

[Requesting Optional Permissions 80](#_Toc426471911)

[How to: Request Optional Permissions by Using the RequestOptional Flag 80](#_Toc426471912)

[How to: Request Optional Permissions by Using the RequestOptional Flag 82](#_Toc426471913)

[Refusing Permissions 86](#_Toc426471914)

[How to: Refuse Permissions by Using the RequestRefuse Flag 86](#_Toc426471915)

[Requesting Built-in Permission Sets 87](#_Toc426471916)

[How to: Request Permission for a Named Permission Set 87](#_Toc426471917)

[Requesting XML-Encoded Permissions 88](#_Toc426471918)

[Using Secure Class Libraries 90](#_Toc426471919)

[Using Managed Wrapper Classes 91](#_Toc426471920)

[Security-Transparent Code 91](#_Toc426471921)

[Security-Transparent Code, Level 1 94](#_Toc426471922)

[Security-Transparent Code, Level 2 98](#_Toc426471923)

[Code Access Security Policy Compatibility and Migration 103](#_Toc426471924)

[Using Libraries from Partially Trusted Code 106](#_Toc426471925)

[Using Libraries from Partially Trusted Code 107](#_Toc426471926)

[Deciding When To Enable Partially Trusted Callers 109](#_Toc426471927)

[Deciding When To Enable Partially Trusted Callers 109](#_Toc426471928)

[Sharing a Library with Partially Trusted Code 110](#_Toc426471929)

[Sharing a Library with Partially Trusted Code 111](#_Toc426471930)

[Requiring Full Trust for Types Within an AllowPartiallyTrustedCallersAttribute Assembly 113](#_Toc426471931)

[Requiring Full Trust for Types Within an AllowPartiallyTrustedCallersAttribute Assembly 114](#_Toc426471932)

[.NET Framework Assemblies Marked with AllowPartiallyTrustedCallersAttribute 115](#_Toc426471933)

[.NET Framework Assemblies Marked with AllowPartiallyTrustedCallersAttribute 117](#_Toc426471934)

[.NET Framework Assemblies Callable by Partially Trusted Code 118](#_Toc426471935)

[Writing Secure Class Libraries 119](#_Toc426471936)

[Security Demands 120](#_Toc426471937)

[Demands 121](#_Toc426471938)

[Demands 123](#_Toc426471939)

[Link Demands 125](#_Toc426471940)

[Link Demands 126](#_Toc426471941)

[Inheritance Demands 127](#_Toc426471942)

[Inheritance Demands 129](#_Toc426471943)

[Overriding Security Checks 130](#_Toc426471944)

[Assert 132](#_Toc426471945)

[Deny 136](#_Toc426471946)

[PermitOnly 139](#_Toc426471947)

[Declarative Security Used with Class and Member Scope 141](#_Toc426471948)

[Declarative Security Used with Class and Member Scope 142](#_Toc426471949)

[Declarative Security Used with Class and Member Scope 144](#_Toc426471950)

[Security Optimizations 146](#_Toc426471951)

[Writing Secure Managed Controls 149](#_Toc426471952)

[Creating Your Own Code Access Permissions 151](#_Toc426471953)

[Creating Your Own Code Access Permissions 152](#_Toc426471954)

[Designing a Permission 153](#_Toc426471955)

[Implementing a Custom Permission 155](#_Toc426471956)

[Adding Declarative Security Support 166](#_Toc426471957)

[Demanding a Custom Permission 168](#_Toc426471958)

[Updating Security Policy 169](#_Toc426471959)

[Creating Other Custom Permissions 171](#_Toc426471960)

[Role-Based Security 173](#_Toc426471961)

[Introduction to Role-Based Security 173](#_Toc426471962)

[Role-Based Security 174](#_Toc426471963)

[Principal and Identity Objects 176](#_Toc426471964)

[Creating WindowsIdentity and WindowsPrincipal Objects 177](#_Toc426471965)

[How to: Create a WindowsPrincipal Object 181](#_Toc426471966)

[Creating GenericPrincipal and GenericIdentity Objects 183](#_Toc426471967)

[How to: Create GenericPrincipal and GenericIdentity Objects 183](#_Toc426471968)

[Replacing a Principal Object 185](#_Toc426471969)

[Impersonating and Reverting 187](#_Toc426471970)

[PrincipalPermission Objects 189](#_Toc426471971)

[Combining PrincipalPermission Objects 189](#_Toc426471972)

[Role-Based Security Checks 191](#_Toc426471973)

[Performing Imperative Security Checks 191](#_Toc426471974)

[How to: Perform Imperative Security Checks 191](#_Toc426471975)

[Performing Declarative Security Checks 194](#_Toc426471976)

[Directly Accessing a Principal Object 194](#_Toc426471977)

[Interoperation with COM+ 1.0 Security 198](#_Toc426471978)

[Cryptographic Services 198](#_Toc426471979)

[Cryptographic Services 199](#_Toc426471980)

[Cryptography Overview 200](#_Toc426471981)

[Cryptographic Services 204](#_Toc426471982)

[.NET Framework Cryptography Model 212](#_Toc426471983)

[.NET Framework Cryptography Model 213](#_Toc426471984)

[Cryptographic Tasks 215](#_Toc426471985)

[Cryptographic Tasks 216](#_Toc426471986)

[Encrypting and Decrypting Data 216](#_Toc426471987)

[Encrypting and Decrypting Data 217](#_Toc426471988)

[Generating Keys for Encryption and Decryption 217](#_Toc426471989)

[Generating Keys for Encryption and Decryption 223](#_Toc426471990)

[Encrypting Data 224](#_Toc426471991)

[Encrypting Data 229](#_Toc426471992)

[Encrypting Data 232](#_Toc426471993)

[Decrypting Data 236](#_Toc426471994)

[Cryptographic Signatures 240](#_Toc426471995)

[Cryptographic Signatures 240](#_Toc426471996)

[Cryptographic Signatures 241](#_Toc426471997)

[Cryptographic Signatures in Silverlight 243](#_Toc426471998)

[Generating Signatures 243](#_Toc426471999)

[Verifying Signatures 244](#_Toc426472000)

[Ensuring Data Integrity with Hash Codes 246](#_Toc426472001)

[Generating a Hash 246](#_Toc426472002)

[Verifying a Hash 247](#_Toc426472003)

[Ensuring Data Integrity with Hash Codes 249](#_Toc426472004)

[Ensuring Data Integrity with Hash Codes 251](#_Toc426472005)

[Creating a Cryptographic Scheme 253](#_Toc426472006)

[Extending the KeyedHashAlgorithm Class 254](#_Toc426472007)

[XML Encryption and Digital Signatures 257](#_Toc426472008)

[How to: Encrypt XML Elements with Symmetric Keys 258](#_Toc426472009)

[How to: Decrypt XML Elements with Symmetric Keys 265](#_Toc426472010)

[How to: Encrypt XML Elements with Asymmetric Keys 270](#_Toc426472011)

[How to: Decrypt XML Elements with Asymmetric Keys 279](#_Toc426472012)

[How to: Encrypt XML Elements with X.509 Certificates 283](#_Toc426472013)

[How to: Decrypt XML Elements with X.509 Certificates 288](#_Toc426472014)

[How to: Sign XML Documents with Digital Signatures 291](#_Toc426472015)

[How to: Verify the Digital Signatures of XML Documents 296](#_Toc426472016)

[How to: Use Data Protection 300](#_Toc426472017)

[How to: Access Hardware Encryption Devices 305](#_Toc426472018)

[Walkthrough: Creating a Cryptographic Application 307](#_Toc426472019)

[Prerequisites 308](#_Toc426472020)

[Creating a Windows Forms Application 308](#_Toc426472021)

[Declaring Global Objects 309](#_Toc426472022)

[Creating an Asymmetric Key 309](#_Toc426472023)

[Encrypting a File 309](#_Toc426472024)

[Decrypting a File 312](#_Toc426472025)

[Exporting a Public Key 315](#_Toc426472026)

[Importing a Public Key 315](#_Toc426472027)

[Getting a Private Key 316](#_Toc426472028)

[Testing the Application 316](#_Toc426472029)

[To create keys, encrypt, and decrypt 316](#_Toc426472030)

[To encrypt using the public key 316](#_Toc426472031)

[To decrypt using the private key 317](#_Toc426472032)

[Security Policy Management 317](#_Toc426472033)

[Security Policy Management 318](#_Toc426472034)

[Security Policy Model 320](#_Toc426472035)

[Security Policy Levels 320](#_Toc426472036)

[Code Groups 322](#_Toc426472037)

[Named Permission Sets 323](#_Toc426472038)

[Evidence 324](#_Toc426472039)

[Application Domain Hosts 325](#_Toc426472040)

[Permission Grants 326](#_Toc426472041)

[Computing the Allowed Permission Set 327](#_Toc426472042)

[Code Group Attributes 328](#_Toc426472043)

[Determining the Granted Permissions 329](#_Toc426472044)

[Default Security Policy 330](#_Toc426472045)

[Administering Security Policy 332](#_Toc426472046)

[Internet Explorer Security and Managed Execution 333](#_Toc426472047)

[How to: Enable Internet Explorer Security Settings for Managed Execution 333](#_Toc426472048)

[Security Policy Best Practices 334](#_Toc426472049)

[Security Policy Administration Overview 334](#_Toc426472050)

[General Security Policy Administration 337](#_Toc426472051)

[Determining When to Modify Security Policy 338](#_Toc426472052)

[Administration Tools 339](#_Toc426472053)

[Administration with Code Group Attributes 340](#_Toc426472054)

[Custom Permissions 341](#_Toc426472055)

[How to: Add Custom Permissions to Security Policy 341](#_Toc426472056)

[Administration Tips 342](#_Toc426472057)

[Deploying Security Policy 343](#_Toc426472058)

[Enterprise Policy Administration 345](#_Toc426472059)

[Machine Policy Administration 346](#_Toc426472060)

[User Policy Administration 346](#_Toc426472061)

[Secure Coding Guidelines 347](#_Toc426472062)

[Secure Coding Overview 348](#_Toc426472063)

[Secure Coding Guidelines 349](#_Toc426472064)

[Permission Requests 352](#_Toc426472065)

[Securing State Data 353](#_Toc426472066)

[Boxed Value Types 353](#_Toc426472067)

[Securing Method Access 354](#_Toc426472068)

[Excluding Classes and Members from Use by Untrusted Code 356](#_Toc426472069)

[Virtual Internal Overrides or Overloads Overridable Friend 359](#_Toc426472070)

[Virtual Internal Overrides or Overloads Overridable Friend 359](#_Toc426472071)

[Securing Wrapper Code 360](#_Toc426472072)

[**Delegates** 360](#_Toc426472073)

[**Link demands and wrappers** 361](#_Toc426472074)

[**Assembly loading wrappers** 361](#_Toc426472075)

[**Demand vs. LinkDemand** 362](#_Toc426472076)

[Unmanaged Code 363](#_Toc426472077)

[Using SuppressUnmanagedCodeSecurityAttribute 364](#_Toc426472078)

[Naming Convention for Unmanaged Code Methods 365](#_Toc426472079)

[Security and Public Read-only Array Fields 365](#_Toc426472080)

[Securing Exception Handling 366](#_Toc426472081)

[Security and User Input 368](#_Toc426472082)

[Security and Remoting Considerations 369](#_Toc426472083)

[**Protected objects** 370](#_Toc426472084)

[**Application domain crossing issues** 370](#_Toc426472085)

[Security and Serialization 371](#_Toc426472086)

[Security and Race Conditions 371](#_Toc426472087)

[Security and On-the-Fly Code Generation 373](#_Toc426472088)

[Dangerous Permissions and Policy Administration 374](#_Toc426472089)

[Security and Setup Issues 375](#_Toc426472090)

[Secure Coding Guidelines for Unmanaged Code 375](#_Toc426472091)

[How to: Run Partially Trusted Code in a Sandbox 377](#_Toc426472092)

[How to: Run Partially Trusted Code in a Sandbox 382](#_Toc426472093)

[Security Tools 388](#_Toc426472094)

[ACL Technology Overview 390](#_Toc426472095)

[Technology Summary for ACLs 390](#_Toc426472096)

[ACL Technology Architecture 394](#_Toc426472097)

[ACL Technology Scenarios 396](#_Toc426472098)

[ACL Propagation Rules 397](#_Toc426472099)

[About System.Security.Cryptography.Pkcs 398](#_Toc426472100)

[Security Services Provided by CMS/PKCS #7 399](#_Toc426472101)

[Algorithm Support in CMS/PKCS #7 400](#_Toc426472102)

[Types of CMS/PKCS #7 Messages 400](#_Toc426472103)

[SignedCms Message 400](#_Toc426472104)

[EnvelopedCms Message 402](#_Toc426472105)

[Enveloped and Signed CMS/PKCS #7 Message 403](#_Toc426472106)

[Components of a CMS/PKCS #7 Message 404](#_Toc426472107)

[Using System.Security.Cryptography.Pkcs 405](#_Toc426472108)

[How to: Sign Messages by One Signer 406](#_Toc426472109)

[How to: Sign a Message by Multiple Signers 409](#_Toc426472110)

[How to: Countersign a Message 415](#_Toc426472111)

[How to: Envelope a Message for One Recipient 422](#_Toc426472112)

[How to: Envelope a Message for Multiple Recipients 426](#_Toc426472113)

[How to: Sign and Envelop a Message 432](#_Toc426472114)

[Supporting Tasks for Using System.Security.Cryptography.Pkcs 440](#_Toc426472115)

[How to: Export and Import a Public Key Certificate 440](#_Toc426472116)

[How to: Add an Authenticated Attribute to a Signed Message 441](#_Toc426472117)

[Windows Identity Foundation 446](#_Toc426472118)

[What's New in Windows Identity Foundation 4.5 446](#_Toc426472119)

[Windows Identity Foundation 4.5 Overview 449](#_Toc426472120)

[Claims-Based Identity Model 450](#_Toc426472121)

[Claims Based Authorization Using WIF 453](#_Toc426472122)

[WIF Claims Programming Model 455](#_Toc426472123)

[Getting Started With WIF 458](#_Toc426472124)

[Building My First Claims-Aware ASP.NET Web Application 458](#_Toc426472125)

[Building My First Claims-Aware WCF Service 460](#_Toc426472126)

[WIF Features 461](#_Toc426472127)

[Custom Token Handlers 461](#_Toc426472128)

[Identity and Access Tool for Visual Studio 2012 463](#_Toc426472129)

[WIF Session Management 463](#_Toc426472130)

[WIF and Web Farms 465](#_Toc426472131)

[WSFederation Authentication Module Overview 471](#_Toc426472132)

[WSTrustChannelFactory and WSTrustChannel 478](#_Toc426472133)

[WIF How-To's Index 481](#_Toc426472134)

[How To: Build Claims-Aware ASP.NET MVC Web Application Using WIF 481](#_Toc426472135)

[How To: Build Claims-Aware ASP.NET Web Forms Application Using WIF 485](#_Toc426472136)

[How To: Build Claims-Aware ASP.NET Application Using Forms-Based Authentication 490](#_Toc426472137)

[How To: Build Claims-Aware ASP.NET Application Using Windows Authentication 492](#_Toc426472138)

[How To: Debug Claims-Aware Applications And Services Using WIF Tracing 495](#_Toc426472139)

[How To: Enable WIF Tracing 501](#_Toc426472140)

[How To: Enable Token Replay Detection 503](#_Toc426472141)

[How To: Enable WIF for a WCF Web Service Application 505](#_Toc426472142)

[How To: Transform Incoming Claims 510](#_Toc426472143)

[WIF Guidelines 514](#_Toc426472144)

[Guidelines for Migrating an Application Built Using WIF 3.5 to WIF 4.5 514](#_Toc426472145)

[Namespace Mapping between WIF 3.5 and WIF 4.5 519](#_Toc426472146)

[WIF Code Sample Index 522](#_Toc426472147)

[WIF Extensions 522](#_Toc426472148)

[JSON Web Token Handler 522](#_Toc426472149)

[Downloading the JSON Web Token Handler Package 523](#_Toc426472150)

[JSON Web Token Handler API 524](#_Toc426472151)

[System.IdentityModel.Tokens Namespace 524](#_Toc426472152)

[Validating Issuer Name Registry 526](#_Toc426472153)

[Downloading the Validating Issuer Name Registry Package 527](#_Toc426472154)

[Validating Issuer Name Registry API 527](#_Toc426472155)

[WIF API Reference 527](#_Toc426472156)

[WIF Configuration Reference 529](#_Toc426472157)

[WIF Configuration Schema Conventions 529](#_Toc426472158)

[Classes 530](#_Toc426472159)

[PrincipalPermission Class 531](#_Toc426472160)

[IPrincipal Interface 531](#_Toc426472161)

[IIdentity Interface 531](#_Toc426472162)

# Securing Applications

**.NET Framework 1.1**

https://msdn.microsoft.com/en-us/library/fkytk30f(v=vs.71).aspx

The common language runtime and the .NET Framework provide many useful classes and services that enable developers to easily write security code. These classes and services also enable system administrators to customize the access that code has to protected resources. In addition, the runtime and the .NET Framework provide useful classes and services that facilitate the use of cryptography and role-based security.

[Key Security Concepts](https://msdn.microsoft.com/en-us/library/z164t8hs(v=vs.71).aspx)

Provides an overview of common language runtime security features. This section is of interest to developers and system administrators.

[Code Access Security](https://msdn.microsoft.com/en-us/library/930b76w0(v=vs.71).aspx)

Describes how to interact with code access security in your code. This section is of interest to developers.

[Role-Based Security](https://msdn.microsoft.com/en-us/library/52kd59t0(v=vs.71).aspx)

Describes how to interact with role-based security in your code. This section is of interest to developers.

[Cryptographic Services](https://msdn.microsoft.com/en-us/library/93bskf9z(v=vs.71).aspx)

Provides an overview of cryptographic services provided by the .NET Framework. This section is of interest to developers.

[Security Policy Management](https://msdn.microsoft.com/en-us/library/c1k0eed6(v=vs.71).aspx)

Describes how to manage code access security policy. This section is of interest to system administrators who manage .NET Framework applications.

[Security Policy Best Practices](https://msdn.microsoft.com/en-us/library/sa4se9bc(v=vs.71).aspx)

Describes some of the best practices for administrating code access security policy. This section is of interest to system administrators who manage .NET Framework applications.

[Secure Coding Guidelines](https://msdn.microsoft.com/en-us/library/d55zzx87(v=vs.71).aspx)

Describes some of the best practices for creating reliable .NET Framework applications.

[Security Tools](https://msdn.microsoft.com/en-us/library/7w3fd0wb(v=vs.71).aspx)

Describes tools that are useful to developers and administrators.

Related Sections

[Securing ASP.NET Web Applications](https://msdn.microsoft.com/en-us/library/330a99hc(v=vs.71).aspx)

Describes ASP.NET security and provides instructions for using it in your code.

[Configuring Security Policy](https://msdn.microsoft.com/en-us/library/7c9c2y1w(v=vs.71).aspx)

Describes how to configure security policy using the .NET Framework Configuration tool and the Code Access Security Policy tool, and how to import security components using XML files.

Security in the .NET Framework

**.NET Framework 2.0**

The common language runtime and the .NET Framework provide many useful classes and services that enable developers to easily write security code. These classes and services also enable system administrators to customize the access that code has to protected resources. In addition, the runtime and the .NET Framework provide useful classes and services that facilitate the use of cryptography and role-based security.

In This Section

[Security How-to Topics](https://msdn.microsoft.com/en-us/library/ms172378(v=vs.80).aspx)

Lists links to How-to topics contained in this section.

[Key Security Concepts](https://msdn.microsoft.com/en-us/library/z164t8hs(v=vs.80).aspx)

Provides an overview of common language runtime security features. This section is of interest to developers and system administrators.

[Code Access Security](https://msdn.microsoft.com/en-us/library/930b76w0(v=vs.80).aspx)

Describes how to interact with code access security in your code. This section is of interest to developers.

[Role-Based Security](https://msdn.microsoft.com/en-us/library/52kd59t0(v=vs.80).aspx)

Describes how to interact with role-based security in your code. This section is of interest to developers.

[Cryptographic Services](https://msdn.microsoft.com/en-us/library/93bskf9z(v=vs.80).aspx)

Provides an overview of cryptographic services provided by the .NET Framework. This section is of interest to developers.

[Security Policy Management](https://msdn.microsoft.com/en-us/library/c1k0eed6(v=vs.80).aspx)

Describes how to manage code access security policy. This section is of interest to system administrators who manage .NET Framework applications.

[Security Policy Best Practices](https://msdn.microsoft.com/en-us/library/sa4se9bc(v=vs.80).aspx)

Describes some of the best practices for administrating code access security policy. This section is of interest to system administrators who manage .NET Framework applications.

[Secure Coding Guidelines](https://msdn.microsoft.com/en-us/library/d55zzx87(v=vs.80).aspx)

Describes some of the best practices for creating reliable .NET Framework applications. This section is of interest to developers.

[Security Tools](https://msdn.microsoft.com/en-us/library/7w3fd0wb(v=vs.80).aspx)

Describes tools that are useful to developers and administrators.

[ACL Technology Overview](https://msdn.microsoft.com/en-us/library/ms229742(v=vs.80).aspx)

Describes the managed APIs that allow you to programmatically create or modify discretionary access control lists (DACLs) and system access control lists (SACLs) for a number of protected resources such as files, folders, and so on. This section is of interest to developers.

[About System.Security.Cryptography.Pkcs](https://msdn.microsoft.com/en-us/library/ms180945(v=vs.80).aspx)

Describes the namespace that contains the managed code implementation of the Cryptographic Message Syntax (CMS) and Public-Key Cryptography Standards #7 (PKCS #7) standards. This section is of interest to developers.

[Using System.Security.Cryptography.Pkcs](https://msdn.microsoft.com/en-us/library/ms180955(v=vs.80).aspx)

Explains how to use the [System.Security.Cryptography.Pkcs](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs(v=vs.80).aspx) namespace to program the Cryptographic Message Syntax (CMS) and Public-Key Cryptography Standards #7 (PKCS #7) standards into your application. This section is of interest to developers.

Related Sections

[ASP.NET Web Application Security](https://msdn.microsoft.com/en-us/library/330a99hc(v=vs.80).aspx)

Describes ASP.NET security and provides instructions for using it in your code.

[Configuring Security Policy](https://msdn.microsoft.com/en-us/library/7c9c2y1w(v=vs.80).aspx)

Describes how to configure security policy using the .NET Framework Configuration tool and the Code Access Security Policy tool, and how to import security components using XML files.

Security in the .NET Framework

**.NET Framework 3.0**

The common language runtime and the .NET Framework provide many useful classes and services that enable developers to easily write security code. These classes and services also enable system administrators to customize the access that code has to protected resources. In addition, the runtime and the .NET Framework provide useful classes and services that facilitate the use of cryptography and role-based security.

**In This Section**

[Security How-to Topics](https://msdn.microsoft.com/en-us/library/ms172378(v=vs.85).aspx)

Lists links to How-to topics contained in this section.

[Key Security Concepts](https://msdn.microsoft.com/en-us/library/z164t8hs(v=vs.85).aspx)

Provides an overview of common language runtime security features. This section is of interest to developers and system administrators.

[Code Access Security](https://msdn.microsoft.com/en-us/library/930b76w0(v=vs.85).aspx)

Describes how to interact with code access security in your code. This section is of interest to developers.

[Role-Based Security](https://msdn.microsoft.com/en-us/library/52kd59t0(v=vs.85).aspx)

Describes how to interact with role-based security in your code. This section is of interest to developers.

[Cryptographic Services](https://msdn.microsoft.com/en-us/library/93bskf9z(v=vs.85).aspx)

Provides an overview of cryptographic services provided by the .NET Framework. This section is of interest to developers.

[Security Policy Management](https://msdn.microsoft.com/en-us/library/c1k0eed6(v=vs.85).aspx)

Describes how to manage code access security policy. This section is of interest to system administrators who manage .NET Framework applications.

[Security Policy Best Practices](https://msdn.microsoft.com/en-us/library/sa4se9bc(v=vs.85).aspx)

Describes some of the best practices for administrating code access security policy. This section is of interest to system administrators who manage .NET Framework applications.

[Secure Coding Guidelines](https://msdn.microsoft.com/en-us/library/d55zzx87(v=vs.85).aspx)

Describes some of the best practices for creating reliable .NET Framework applications. This section is of interest to developers.

[Security Tools](https://msdn.microsoft.com/en-us/library/7w3fd0wb(v=vs.85).aspx)

Describes tools that are useful to developers and administrators.

[ACL Technology Overview](https://msdn.microsoft.com/en-us/library/ms229742(v=vs.85).aspx)

Describes the managed APIs that allow you to programmatically create or modify discretionary access control lists (DACLs) and system access control lists (SACLs) for a number of protected resources such as files, folders, and so on. This section is of interest to developers.

[About System.Security.Cryptography.Pkcs](https://msdn.microsoft.com/en-us/library/ms180945(v=vs.85).aspx)

Describes the namespace that contains the managed code implementation of the Cryptographic Message Syntax (CMS) and Public-Key Cryptography Standards #7 (PKCS #7) standards. This section is of interest to developers.

[Using System.Security.Cryptography.Pkcs](https://msdn.microsoft.com/en-us/library/ms180955(v=vs.85).aspx)

Explains how to use the [System.Security.Cryptography.Pkcs](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs(v=vs.85).aspx) namespace to program the Cryptographic Message Syntax (CMS) and Public-Key Cryptography Standards #7 (PKCS #7) standards into your application. This section is of interest to developers.

**Related Sections**

[ASP.NET Web Application Security](https://msdn.microsoft.com/en-us/library/330a99hc(v=vs.85).aspx)

Describes ASP.NET security and provides instructions for using it in your code.

[Configuring Security Policy](https://msdn.microsoft.com/en-us/library/7c9c2y1w(v=vs.85).aspx)

Describes how to configure security policy using the .NET Framework Configuration tool and the Code Access Security Policy tool, and how to import security components using XML files.

# Security in the .NET Framework

**.NET Framework 4**

The common language runtime and the .NET Framework provide many useful classes and services that enable developers to easily write secure code and enable system administrators to customize the permissions granted to code so that it can access protected resources. In addition, the runtime and the .NET Framework provide useful classes and services that facilitate the use of cryptography and role-based security.

|  |
| --- |
| **Important noteImportant** |
| Effective with the .NET Framework version 4, there are major changes to the code access security system. Security policy is no longer applied to applications. All applications that can be run from the desktop are now executed as full-trust applications. This includes both applications on the computer and applications that can be run from a network share. Partially trusted applications must be run in a sandbox, which determines their grant set. The permission system continues to be used, but it is transcended by security transparency rules. For information about these changes, see [Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx). |

[**In This Section**](javascript:void(0))

[Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx)

Describes important changes to the .NET Framework security system.

[Security How-to Topics](https://msdn.microsoft.com/en-us/library/ms172378(v=vs.100).aspx)

Lists links to How-to topics contained in this section.

[Key Security Concepts](https://msdn.microsoft.com/en-us/library/z164t8hs(v=vs.100).aspx)

Provides an overview of common language runtime security features. This section is of interest to developers and system administrators.

[Code Access Security](https://msdn.microsoft.com/en-us/library/c5tk9z76(v=vs.100).aspx)

Describes how to interact with code access security in your code. This section is important to developers and can be of interest to system administrators.

[Role-Based Security](https://msdn.microsoft.com/en-us/library/shz8h065(v=vs.100).aspx)

Describes how to interact with role-based security in your code. This section is of interest to developers.

[Cryptographic Services](https://msdn.microsoft.com/en-us/library/92f9ye3s(v=vs.100).aspx)

Provides an overview of cryptographic services provided by the .NET Framework. This section is of interest to developers.

[Security Policy Management](https://msdn.microsoft.com/en-us/library/c1k0eed6(v=vs.100).aspx)

(Applies only to code that uses legacy security rules.) Describes how to manage code access security policy. This section is of interest to system administrators who manage .NET Framework applications.

[Security Policy Best Practices](https://msdn.microsoft.com/en-us/library/sa4se9bc(v=vs.100).aspx)

(Applies only to code that uses legacy security rules.) Describes some of the best practices for administrating code access security policy. This section is of interest to system administrators who manage .NET Framework applications.

[Secure Coding Guidelines](https://msdn.microsoft.com/en-us/library/8a3x2b7f(v=vs.100).aspx)

Describes some of the best practices for creating reliable .NET Framework applications. This section is of interest to developers.

[ACL Technology Overview](https://msdn.microsoft.com/en-us/library/ms229742(v=vs.100).aspx)

Describes the managed classes that enable you to programmatically create or modify discretionary access control lists (DACLs) and system access control lists (SACLs) for protected resources such as files and folders. This section is of interest to developers.

[**Related Sections**](javascript:void(0))

[Security Tools (.NET Framework)](https://msdn.microsoft.com/en-us/library/dd233106(v=vs.100).aspx)

Describes command-line tools that help you perform security-related tasks, such as configuring security policy, managing certificates, and digitally signing files.

[ASP.NET Web Application Security](https://msdn.microsoft.com/en-us/library/330a99hc(v=vs.100).aspx)

Describes ASP.NET security and provides instructions for using it in your code.

[Configuring Security Policy](https://msdn.microsoft.com/en-us/library/7c9c2y1w(v=vs.100).aspx)

(Aapplies only to code that uses legacy security rules.) Describes how to configure security policy using the .NET Framework Configuration tool and the Code Access Security Policy tool, and how to import security components using XML files.

# Security Changes in the .NET Framework 4

**.NET Framework 4**

There have been two major changes to security in the .NET Framework version 4. Machine-wide security policy has been eliminated, although the permissions system remains in place, and [security transparency](https://msdn.microsoft.com/en-us/library/ee191569(v=vs.100).aspx) has become the default enforcement mechanism. (For more information, see [Security-Transparent Code, Level 2](https://msdn.microsoft.com/en-us/library/dd233102(v=vs.100).aspx).) In addition, some permission operations that presented the potential for security vulnerabilities have been made obsolete.

|  |
| --- |
| **Important noteImportant** |
| Code access security (CAS) has not been eliminated, Security policy has been eliminated from CAS, but evidence and permissions are still in effect. A few permissions have been eliminated, and transparency has simplified the enforcement of security. For a brief overview of the changes, see [Summary of Changes in Code Access Security](https://msdn.microsoft.com/en-us/library/ff527276(v=vs.100).aspx). |

You should be aware of the following key points:

* Transparency separates code that runs as part of the application from code that runs as part of the infrastructure. It was introduced in .NET Framework version 2.0, and has been enhanced to become the code access security enforcement mechanism. Unlike security policy, [level 2 transparency](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx#transparency2) rules are enforced at run time, not at assembly load time. These rules are always in effect, even for assemblies that run as fully trusted by default. However, level 2 transparency does not affect fully trusted code that is not annotated, such as desktop applications. Assemblies (including desktop assemblies) that are marked with the [SecurityTransparentAttribute](https://msdn.microsoft.com/en-us/library/system.security.securitytransparentattribute(v=vs.100).aspx) and that call methods marked with the [SecurityCriticalAttribute](https://msdn.microsoft.com/en-us/library/system.security.securitycriticalattribute(v=vs.100).aspx) receive a [MethodAccessException](https://msdn.microsoft.com/en-us/library/system.methodaccessexception(v=vs.100).aspx). You can change this behavior by applying the [SecurityRulesAttribute](https://msdn.microsoft.com/en-us/library/system.security.securityrulesattribute(v=vs.100).aspx) and setting the [SecurityRulesAttribute.RuleSet](https://msdn.microsoft.com/en-us/library/dd288474(v=vs.100).aspx) property to [Level1](https://msdn.microsoft.com/en-us/library/system.security.securityruleset(v=vs.100).aspx); however, you should do this only for backwards compatibility. You must explicitly mark a desktop application as security-transparent to apply transparency restrictions to it.
* Code that calls security policy APIs receives a [NotSupportedException](https://msdn.microsoft.com/en-us/library/system.notsupportedexception(v=vs.100).aspx) in addition to compiler warnings at run time. Policy may be re-enabled by using the [<NetFx40\_LegacySecurityPolicy> configuration element](https://msdn.microsoft.com/en-us/library/dd409253(v=vs.100).aspx). When policy is enabled, security transparency is still in effect. Security policy is applied at assembly load time and has no effect on transparency, which is enforced by the runtime.
* The obsolete request permissions ([RequestMinimum](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx), [RequestOptional](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx), and [RequestRefuse](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx)) receive compiler warnings and do not work in the .NET Framework 4, but they do not cause an exception to be thrown. [Deny](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx) requests cause a [NotSupportedException](https://msdn.microsoft.com/en-us/library/system.notsupportedexception(v=vs.100).aspx) to be thrown at run time.
* The [LinkDemand](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx) security action is not obsolete, but it should not be used for verifying permissions. Instead, use the[SecurityCriticalAttribute](https://msdn.microsoft.com/en-us/library/system.security.securitycriticalattribute(v=vs.100).aspx) for types and methods that require full trust, or use the [Demand](https://msdn.microsoft.com/en-us/library/system.security.codeaccesspermission.demand(v=vs.100).aspx) method for types and methods that require individual permissions.
* If your application is built with Visual Studio 2010, you can run it without these changes by specifying a target .NET Framework version that is earlier than the .NET Framework 4 in the Visual Studio project settings. However, you will not be able to use new .NET Framework 4 types and members. You can also specify an earlier version of the .NET Framework by using the [<supportedRuntime> element](https://msdn.microsoft.com/en-us/library/w4atty68(v=vs.100).aspx) in the startup settings schema in your [application configuration file](https://msdn.microsoft.com/en-us/library/1xtk877y(v=vs.100).aspx).

The following sections discuss these and other changes in the .NET Framework 4:

* [Security Policy Simplification](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx#simplification)
* [Security Transparency Level 2](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx#transparency2)
* [Obsolete Permission Requests](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx#obsolete)
* [Conditional APTCA](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx#aptca)
* [Evidence Objects](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx#evidence_object)
* [Evidence Collections](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx#evidence_collection)

[**Security Policy Simplification**](javascript:void(0))

Starting with the .NET Framework 4, the common language runtime (CLR) is moving away from providing security policy for computers. Historically, the .NET Framework has provided code access security (CAS) policy as a mechanism to tightly control and configure the capabilities of managed code. Although CAS policy is powerful, it can be complicated and restrictive. Furthermore, CAS policy does not apply to native applications, so its security guarantees are limited. System administrators should look to operating system-level solutions such as [Windows Software Restriction Policies](http://go.microsoft.com/fwlink/?LinkId=178101) (SRP) or [AppLocker](http://go.microsoft.com/fwlink/?LinkId=178102) on Windows 7 and Windows Server 2008 R2 as a replacement for CAS policy. SRP and AppLocker policies provide simple trust mechanisms that apply to both managed and native code. As a security policy solution, SRP and AppLocker are simpler and provide better security guarantees than CAS.

In the .NET Framework 4, machine-wide security policy is turned off by default. Applications that are not hosted (that is, applications that are executed through Windows Explorer or from a command prompt) now run as full trust. This includes all applications that reside on shares on the local network. Hosted or sandboxed applications continue to run with trust policies that are decided by their hosts (for example, by Internet Explorer, ClickOnce, or ASP.NET). Applications or controls that run in sandboxes are considered partially trusted.

To simplify the security policy, the transparency model has been applied to the .NET Framework. Applications and controls that run in a host or sandbox with the limited permission set granted by the sandbox are considered transparent. Transparency means that you do not have to be concerned about checking CAS policy when you are running partially trusted applications. Transparent applications just run using their grant set. As a programmer, your only concern should be that your applications target the grant set for their sandbox and that they do not call code that requires full trust (security-critical code).

|  |
| --- |
| **Important noteImportant** |
| As a result of these security policy changes, you may encounter compilation warnings and runtime exceptions if you call the obsolete CAS policy types and members either explicitly or implicitly (through other types and members). For a list of obsolete types and members and their replacements, see [Code Access Security Policy Compatibility and Migration](https://msdn.microsoft.com/en-us/library/ee191568(v=vs.100).aspx).  You can avoid the warnings and errors by using the [<NetFx40\_LegacySecurityPolicy> configuration element](https://msdn.microsoft.com/en-us/library/dd409253(v=vs.100).aspx) in the runtime settings schema to opt into the legacy CAS policy behavior. However, specifying the use of legacy security policy does not include any custom CAS policy for that version unless it is migrated to .NET Framework 4.  You can also enable legacy CAS policy by setting the target .NET Framework version for your Visual Studio project to an earlier version than the .NET Framework 4. This enables legacy CAS policy and includes any custom CAS policies you specified for that version. However, you will not be able to use new .NET Framework 4 types and members. You can also specify an earlier version of the .NET Framework by using the [<supportedRuntime> element](https://msdn.microsoft.com/en-us/library/w4atty68(v=vs.100).aspx) in the startup settings schema. |

[Back to top](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx#top)

[**Security Transparency Level 2**](javascript:void(0))

Security transparency was introduced in the .NET Framework version 2.0, but it was very limited and used primarily to improve code validation efficiency. In the .NET Framework 4, transparency is an enforcement mechanism that separates code that runs as part of the application from code that runs as part of the infrastructure. Transparency draws a barrier between code that can do privileged things (critical code), such as calling native code, and code that cannot (transparent code). Transparent code can execute commands within the bounds of the permission set it is operating within, but cannot execute, call, derive from, or contain critical code.

The primary goal of transparency enforcement is to provide a simple, effective mechanism for isolating different groups of code based on privilege. In the sandboxing model, these privilege groups are either fully trusted (that is, not restricted) or partially trusted (that is, restricted to the permission set granted to the sandbox).

Desktop applications run as fully trusted; therefore, they are not affected by the transparency model. For more information about security transparency changes, see [Security-Transparent Code, Level 2](https://msdn.microsoft.com/en-us/library/dd233102(v=vs.100).aspx).

[Back to top](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx#top)

[**Obsolete Permission Requests**](javascript:void(0))

Runtime support has been removed for enforcing the [Deny](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx), [RequestMinimum](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx), [RequestOptional](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx), and [RequestRefuse](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx) permission requests. In general, these requests were not well understood and presented the potential for security vulnerabilities when they were not used properly:

* A [Deny](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx) action could be easily overridden by an [Assert](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx) action. The code in an assembly was able to execute an [Assert](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx) action for a permission if the permission was in the grant set for the assembly. The [Assert](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx) prevented the [Deny](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx) from being seen on the stack, making it ineffective.
* [RequestMinimum](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx) could not be used effectively outside the application scope. If [RequestMinimum](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx) appeared on an executable (.exe) file and the grant set was not met, end users of the file received an unhandled [FileLoadException](https://msdn.microsoft.com/en-us/library/system.io.fileloadexception(v=vs.100).aspx) exception that contained no information about how to correct the problem. You could not use a single minimum request set for libraries (.dll files), because different types and members in the assembly generally have different permission requirements.
* [RequestOptional](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx) was confusing and often used incorrectly with unexpected results. Developers could easily omit permissions from the list without realizing that doing so implicitly refused the omitted permissions.
* [RequestRefuse](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx) did not provide an effective least-privilege model, because it required you to explicitly identify permissions you did not want instead of identifying the permissions you needed. In addition, if new permissions became available, they would not be included in the list. Furthermore, refusal did not make sense for all permissions. For example, you could refuse a value for the[UserQuota](https://msdn.microsoft.com/en-us/library/system.security.permissions.isolatedstoragepermission.userquota(v=vs.100).aspx) property in the [IsolatedStoragePermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.isolatedstoragepermission(v=vs.100).aspx).

Finally, specifying only the permissions you did not want created the potential for security vulnerabilities if you failed to identify all potentially harmful permissions.

* [RequestOptional](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx) and [RequestRefuse](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx) enabled developers to break homogenous domains by creating multiple permission sets within the domain.

The .NET Framework 4 removes runtime enforcement of these enumeration values. Assemblies containing the attributes that use these[SecurityAction](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx) values will continue to load; however, the CLR will not refuse to load the referenced assemblies or modify their grant set based upon the permission sets.

[Back to top](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx#top)

[**Conditional APTCA**](javascript:void(0))

The conditional use of the [AllowPartiallyTrustedCallersAttribute](https://msdn.microsoft.com/en-us/library/system.security.allowpartiallytrustedcallersattribute(v=vs.100).aspx) (APTCA) attribute enables hosts to identify which assemblies they want to expose to partial-trust callers that are loaded within the context of the host. The candidate assemblies must already be designed for partial trust; that is, they must either be APTCA (security-safe-critical in the transparency model) or fully transparent. A new constructor for the[AllowPartiallyTrustedCallersAttribute](https://msdn.microsoft.com/en-us/library/system.security.allowpartiallytrustedcallersattribute(v=vs.100).aspx) enables the host to specify the level of visibility for an APTCA assembly by using the[PartialTrustVisibilityLevel](https://msdn.microsoft.com/en-us/library/system.security.partialtrustvisibilitylevel(v=vs.100).aspx) enumeration in the constructor call.

[Back to top](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx#top)

[**Evidence Objects**](javascript:void(0))

Before the .NET Framework 4, almost any object could be used as an evidence object if the hosting code wanted to apply it as evidence. For example, some .NET Framework code recognized [System.Uri](https://msdn.microsoft.com/en-us/library/system.uri(v=vs.100).aspx) objects as evidence. The runtime considered evidence objects as[System.Object](https://msdn.microsoft.com/en-us/library/system.object(v=vs.100).aspx) references and did not apply any type safety to them.

This presented a problem because the .NET Framework imposed implicit restrictions on which types could be used as evidence objects. Specifically, any object used as evidence had to be serializable and could not be null. If these requirements were not met, the CLR threw an exception whenever an operation that required one of these assumptions was performed.

To enable constraints on the types of objects that can be used as evidence and to provide the ability to add new features and requirements to all evidence objects, the .NET Framework 4 introduces a new base class, [System.Security.Policy.EvidenceBase](https://msdn.microsoft.com/en-us/library/system.security.policy.evidencebase(v=vs.100).aspx), which all evidence objects must derive from. The [EvidenceBase](https://msdn.microsoft.com/en-us/library/system.security.policy.evidencebase(v=vs.100).aspx) class ensures, upon instantiation, that the evidence object is serializable. In addition, new evidence requirements can be created in the future by adding new default implementations to the base class.

Backward Compatibility

All the types used by the CLR as evidence objects have been updated in the .NET Framework 4 to derive from [EvidenceBase](https://msdn.microsoft.com/en-us/library/system.security.policy.evidencebase(v=vs.100).aspx). However, custom evidence types used by third-party applications are not known and cannot be updated. Therefore, those evidence types cannot be used with the new members that expect evidence derived from [EvidenceBase](https://msdn.microsoft.com/en-us/library/system.security.policy.evidencebase(v=vs.100).aspx).

[Back to top](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx#top)

[**Evidence Collections**](javascript:void(0))

Before the .NET Framework 4, the CLR generated the full set of evidence objects that applied to an assembly when the assembly was loaded. These objects were stored in a list, which consumers would then iterate over looking for a specific object. Therefore, all evidence was made available, whether or not it was used. For most evidence objects, this behavior was not an issue; however, for evidence objects such as [System.Security.Policy.Publisher](https://msdn.microsoft.com/en-us/library/system.security.policy.publisher(v=vs.100).aspx) (which requires Authenticode verification), this behavior was inefficient.

To improve this behavior, the interaction with the evidence collection has been redesigned in .NET Framework 4. An evidence collection now behaves like a dictionary instead of a list. Instead of iterating over the evidence collection to see if a required evidence object exists, consumers can now request a specific type of evidence, and the collection returns the evidence if it is found. For example, the callStrongName name = evidence.GetHostEvidence<StrongName>(); returns a [StrongName](https://msdn.microsoft.com/en-us/library/system.security.policy.strongname(v=vs.100).aspx) object if one exists; otherwise, it returns null.

This dictionary model delays the generation of evidence objects until they are requested. In the [Publisher](https://msdn.microsoft.com/en-us/library/system.security.policy.publisher(v=vs.100).aspx) evidence example, the performance overhead of verifying the Authenticode signature of an assembly is delayed until that information is needed. In the most common case of full-trust applications where [Publisher](https://msdn.microsoft.com/en-us/library/system.security.policy.publisher(v=vs.100).aspx) evidence is not needed, the verification process is avoided altogether.

[Back to top](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx#top)

[**See Also**](javascript:void(0))

Concepts

[Security-Transparent Code](https://msdn.microsoft.com/en-us/library/ee191569(v=vs.100).aspx)

[Security-Transparent Code, Level 2](https://msdn.microsoft.com/en-us/library/dd233102(v=vs.100).aspx)

[Code Access Security Policy Compatibility and Migration](https://msdn.microsoft.com/en-us/library/ee191568(v=vs.100).aspx)

## Summary of Changes in Code Access Security

**.NET Framework 4**

In the .NET Framework version 4, code access security (CAS) has undergone major changes, with the purpose of simplifying the security system. In earlier versions of the .NET Framework, the rights of a managed application were determined by security policy rules, which were set computer-wide to establish runtime settings. Starting with the .NET Framework 4:

* Security policy is no longer in effect. Permissions are still in use; only the policy system has been eliminated.
* Access rights for applications are determined by two factors: their permissions (the grant set established by their application domain) and their [transparency](https://msdn.microsoft.com/en-us/library/ff527276(v=vs.100).aspx#security_transparency). All partial-trust applications are classified as transparent. Transparent applications do not have to be concerned with security. Transparency was first used for Microsoft Silverlight and has now been extended to all hosted environments.
* Desktop and local intranet applications are granted full trust.

|  |
| --- |
| **Important noteImportant** |
| The major change to CAS is the elimination of security policy. CAS itself has not been eliminated; only the use of policy (and some permission requests) has been removed. |

This topic provides a short overview of CAS changes in the .NET Framework 4. For more information, see [Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx).

[**Sandboxing and the Permission Model**](javascript:void(0))

The following list describes the trust model for desktop and hosted applications in the .NET Framework 4. For more information, see[Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx).

* **Desktop applications.** As in previous versions of the .NET Framework, managed applications that reside on the desktop (unless they have been downloaded from the Web) are granted full trust. Applications that reside on shares on the local intranet are also granted full trust. You can no longer use policy to restrict permissions for an application based on its folder on the local hard drive.
* **Hosted applications.** Applications that run in a sandbox (for example, Silverlight-based applications) are granted a limited set of permissions that determine which computer resources they can access (for example, which files they are allowed to use). Sandboxes provide the ability to identify some assemblies within the sandbox as being partially trusted and some as being fully trusted. The partial-trust assemblies are granted a specific set of permissions, as determined by the application domain ([System.AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.100).aspx)) that created the sandbox. Some of the full-trust code in the full-trust libraries can be called by partially trusted code. That trusted code can make calls to protected resources on the computer. However, publicly accessible full-trust types and members that can call into protected resources must have undergone a security audit. Those members are classified as being safe-critical, as discussed in the next section. They can be called by partial-trust (transparent) code and, in turn, they can call into critical code.

[**Security Transparency**](javascript:void(0))

Security transparency separates security-sensitive code from non-security-sensitive code. It was introduced in the .NET Framework version 2.0 to make security audits easier by annotating code that had to perform security-sensitive actions as security-critical. This meant that any code that was not security-critical (that is, transparent code) did not require a thorough review. However, in these earlier versions of the .NET Framework, transparency was used only by Microsoft code.

In the .NET Framework 4, this model has been extended and the rules have been tightened to turn security transparency into an enforcement model. In this enhanced model, code that is security-sensitive and callable by partial-trust applications is more easily identifiable. This further reduces the surface area that has to be audited.

The following table shows the transparency categories and the associated attributes for annotating code.

|  |  |  |
| --- | --- | --- |
| **Security category** | **Attribute** | **Description** |
| Transparent | [SecurityTransparentAttribute](https://msdn.microsoft.com/en-us/library/system.security.securitytransparentattribute(v=vs.100).aspx) | Code that does not do anything that is inherently security-sensitive. |
| Critical | [SecurityCriticalAttribute](https://msdn.microsoft.com/en-us/library/system.security.securitycriticalattribute(v=vs.100).aspx) | Code that can do anything, but that cannot be called from partial-trust applications. |
| Safe-critical | [SecuritySafeCriticalAttribute](https://msdn.microsoft.com/en-us/library/system.security.securitysafecriticalattribute(v=vs.100).aspx) | Code that can do anything and that can be called from partial-trust applications. This is the safe brokering layer; its purpose is to perform proper security checks and validation before calling critical code. |

Transparent code cannot do the following, regardless of the permissions granted to it:

* Contain unverifiable code.
* Use [platform invoke](https://msdn.microsoft.com/en-us/library/26thfadc(v=vs.100).aspx).
* Perform [Assert](https://msdn.microsoft.com/en-us/library/system.security.codeaccesspermission.assert(v=vs.100).aspx) operations.
* Call critical code.
* Derive from critical code.
* Call code that is protected by a [LinkDemand](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx) (that is, code that is considered critical).

If your code tries to violate these rules, exceptions are thrown (even if your code has full trust). For more information, see [Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx).

Note that security sensitivity is defined in the common language runtime (CLR) as actions that are prohibited for transparent code. The transparency model does not protect against scenario-specific security violations such as storing passwords in fields.

[**How the Security Model Works**](javascript:void(0))

* Each [AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.100).aspx) has an associated permission set, which is defined by the host in a hosted scenario. (The permission set is full trust for code that is not hosted.)
* Partial-trust code is always transparent; therefore, it cannot perform the actions prohibited for transparent code (see [transparency](https://msdn.microsoft.com/en-us/library/ff527276(v=vs.100).aspx#security_transparency)).
* By default, full-trust code is critical unless it has been marked as being transparent. If a desktop application is marked as transparent, it cannot call critical code, even though it has full trust.
* Libraries may be exposed to partial-trust code both by the host and by the .NET Framework. These libraries contain a mix of transparent, critical, and safe-critical code.
* The safe-critical code must demand appropriate permissions before using critical functionality. For example, the [File.Open](https://msdn.microsoft.com/en-us/library/system.io.file.open(v=vs.100).aspx) method demands [FileIOPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.fileiopermission(v=vs.100).aspx) before it opens a file.
* The safe-critical code must also perform any other checks and validation before and after calls to critical functionality. For example, exceptions and messages may have to be filtered before being passed to partially trusted code.
* Critical code has to assert the permissions it needs when it is called by partial-trust code, because critical code might be doing something that the partial-trust code is not allowed to do.

[**See Also**](javascript:void(0))

**Concepts**

[Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx)

[Security-Transparent Code](https://msdn.microsoft.com/en-us/library/ee191569(v=vs.100).aspx)

# Security Changes in the .NET Framework

**.NET Framework 4.6 and 4.5**

The most important change to security in the .NET Framework 4.5 is in strong naming. See [Enhanced Strong Naming](https://msdn.microsoft.com/en-us/library/hh415055(v=vs.110).aspx) for a description of those changes.

The .NET Framework 4.5 provides a two-tier security model for managed applications. Windows Store apps run in a Windows security container that limits access to resources. Within that container, managed applications run fully trusted. From a code access security (CAS) perspective, there is nothing a developer can do to elevate privileges. For information about the privileges granted by Windows, see [App capability declarations (Windows Store apps)](http://go.microsoft.com/fwlink/?LinkId=230436) in the Windows Dev Center. For information about creating a Windows Store app, see [Create your first Windows Store app using C# or Visual Basic](http://go.microsoft.com/fwlink/?LinkId=230461).

|  |
| --- |
| **Important note Important** |
| Important changes to code access security (CAS) were made in the .NET Framework 4. For a description of those changes, see [Security Changes in the .NET Framework 4](http://go.microsoft.com/fwlink/?LinkId=230442). |

[**See Also**](javascript:void(0))

**Concepts**

[Security-Transparent Code](https://msdn.microsoft.com/en-us/library/ee191569(v=vs.110).aspx)

[Security-Transparent Code, Level 2](https://msdn.microsoft.com/en-us/library/dd233102(v=vs.110).aspx)

[Code Access Security Policy Compatibility and Migration](https://msdn.microsoft.com/en-us/library/ee191568(v=vs.110).aspx)

## Security How-to Topics

**.NET Framework 2.0**

The following list includes links to the How-to topics found in the conceptual documentation for security.

* [How to: Request Permission to Access Unmanaged Code](https://msdn.microsoft.com/en-US/library/d0313z05(v=vs.80).aspx)
* [How to: Request Minimum Permissions by Using the RequestMinimum Flag](https://msdn.microsoft.com/en-US/library/ms229913(v=vs.80).aspx)
* [How to: Request Optional Permissions by Using the RequestOptional Flag](https://msdn.microsoft.com/en-US/library/ea5yat38(v=vs.80).aspx)
* [How to: Refuse Permissions by Using the RequestRefuse Flag](https://msdn.microsoft.com/en-US/library/4b7hy971(v=vs.80).aspx)
* [How to: Request Permission for a Named Permission Set](https://msdn.microsoft.com/en-US/library/0d005ted(v=vs.80).aspx)
* [How to: Create a WindowsPrincipal Object](https://msdn.microsoft.com/en-US/library/t6547wf1(v=vs.80).aspx)
* [How to: Create GenericPrincipal and GenericIdentity Objects](https://msdn.microsoft.com/en-US/library/y9dd5fx0(v=vs.80).aspx)
* [How to: Perform Imperative Security Checks](https://msdn.microsoft.com/en-US/library/dc8ztsad(v=vs.80).aspx)
* [How to: Store Asymmetric Keys in a Key Container](https://msdn.microsoft.com/en-US/library/tswxhw92(v=vs.80).aspx)
* [How to: Encrypt XML Elements with Symmetric Keys](https://msdn.microsoft.com/en-US/library/sb7w85t6(v=vs.80).aspx)
* [How to: Decrypt XML Elements with Symmetric Keys](https://msdn.microsoft.com/en-US/library/ms229740(v=vs.80).aspx)
* [How to: Encrypt XML Elements with Asymmetric Keys](https://msdn.microsoft.com/en-US/library/ms229746(v=vs.80).aspx)
* [How to: Decrypt XML Elements with Asymmetric Keys](https://msdn.microsoft.com/en-US/library/ms229919(v=vs.80).aspx)
* [How to: Encrypt XML Elements with X.509 Certificates](https://msdn.microsoft.com/en-US/library/ms229744(v=vs.80).aspx)
* [How to: Decrypt XML Elements with X.509 Certificates](https://msdn.microsoft.com/en-US/library/ms229943(v=vs.80).aspx)
* [How to: Sign XML Documents with Digital Signatures](https://msdn.microsoft.com/en-US/library/ms229745(v=vs.80).aspx)
* [How to: Verify the Digital Signatures of XML Documents](https://msdn.microsoft.com/en-US/library/ms229950(v=vs.80).aspx)
* [How to: Use Data Protection](https://msdn.microsoft.com/en-US/library/ms229741(v=vs.80).aspx)
* [How to: Enable Internet Explorer Security Settings for Managed Execution](https://msdn.microsoft.com/en-US/library/101853ac(v=vs.80).aspx)
* [How to: Add Custom Permissions to Security Policy](https://msdn.microsoft.com/en-US/library/xt8xsee5(v=vs.80).aspx)
* [How to: Sign Messages by One Signer](https://msdn.microsoft.com/en-US/library/ms180956(v=vs.80).aspx)
* [How to: Sign a Message by Multiple Signers](https://msdn.microsoft.com/en-US/library/ms180957(v=vs.80).aspx)
* [How to: Countersign a Message](https://msdn.microsoft.com/en-US/library/ms180958(v=vs.80).aspx)
* [How to: Envelope a Message for One Recipient](https://msdn.microsoft.com/en-US/library/ms180959(v=vs.80).aspx)
* [How to: Envelope a Message for Multiple Recipients](https://msdn.microsoft.com/en-US/library/ms180960(v=vs.80).aspx)
* [How to: Sign and Envelop a Message](https://msdn.microsoft.com/en-US/library/ms180961(v=vs.80).aspx)
* [How to: Export and Import a Public Key Certificate](https://msdn.microsoft.com/en-US/library/ms180963(v=vs.80).aspx)
* [How to: Add an Authenticated Attribute to a Signed Message](https://msdn.microsoft.com/en-US/library/ms180964(v=vs.80).aspx)
* [How to: Access Hardware Encryption Devices](https://msdn.microsoft.com/en-US/library/ms229931(v=vs.80).aspx)

## Security How-to Topics

**.NET Framework 3.0**

The following list includes links to the How-to topics found in the conceptual documentation for security.

* [How to: Request Permission to Access Unmanaged Code](https://msdn.microsoft.com/en-us/library/d0313z05(v=vs.85).aspx)
* [How to: Request Minimum Permissions by Using the RequestMinimum Flag](https://msdn.microsoft.com/en-us/library/ms229913(v=vs.85).aspx)
* [How to: Request Optional Permissions by Using the RequestOptional Flag](https://msdn.microsoft.com/en-us/library/ea5yat38(v=vs.85).aspx)
* [How to: Refuse Permissions by Using the RequestRefuse Flag](https://msdn.microsoft.com/en-us/library/4b7hy971(v=vs.85).aspx)
* [How to: Request Permission for a Named Permission Set](https://msdn.microsoft.com/en-us/library/0d005ted(v=vs.85).aspx)
* [How to: Create a WindowsPrincipal Object](https://msdn.microsoft.com/en-us/library/t6547wf1(v=vs.85).aspx)
* [How to: Create GenericPrincipal and GenericIdentity Objects](https://msdn.microsoft.com/en-us/library/y9dd5fx0(v=vs.85).aspx)
* [How to: Perform Imperative Security Checks](https://msdn.microsoft.com/en-us/library/dc8ztsad(v=vs.85).aspx)
* [How to: Store Asymmetric Keys in a Key Container](https://msdn.microsoft.com/en-us/library/tswxhw92(v=vs.85).aspx)
* [How to: Encrypt XML Elements with Symmetric Keys](https://msdn.microsoft.com/en-us/library/sb7w85t6(v=vs.85).aspx)
* [How to: Decrypt XML Elements with Symmetric Keys](https://msdn.microsoft.com/en-us/library/ms229740(v=vs.85).aspx)
* [How to: Encrypt XML Elements with Asymmetric Keys](https://msdn.microsoft.com/en-us/library/ms229746(v=vs.85).aspx)
* [How to: Decrypt XML Elements with Asymmetric Keys](https://msdn.microsoft.com/en-us/library/ms229919(v=vs.85).aspx)
* [How to: Encrypt XML Elements with X.509 Certificates](https://msdn.microsoft.com/en-us/library/ms229744(v=vs.85).aspx)
* [How to: Decrypt XML Elements with X.509 Certificates](https://msdn.microsoft.com/en-us/library/ms229943(v=vs.85).aspx)
* [How to: Sign XML Documents with Digital Signatures](https://msdn.microsoft.com/en-us/library/ms229745(v=vs.85).aspx)
* [How to: Verify the Digital Signatures of XML Documents](https://msdn.microsoft.com/en-us/library/ms229950(v=vs.85).aspx)
* [How to: Use Data Protection](https://msdn.microsoft.com/en-us/library/ms229741(v=vs.85).aspx)
* [How to: Enable Internet Explorer Security Settings for Managed Execution](https://msdn.microsoft.com/en-us/library/101853ac(v=vs.85).aspx)
* [How to: Add Custom Permissions to Security Policy](https://msdn.microsoft.com/en-us/library/xt8xsee5(v=vs.85).aspx)
* [How to: Sign Messages by One Signer](https://msdn.microsoft.com/en-us/library/ms180956(v=vs.85).aspx)
* [How to: Sign a Message by Multiple Signers](https://msdn.microsoft.com/en-us/library/ms180957(v=vs.85).aspx)
* [How to: Countersign a Message](https://msdn.microsoft.com/en-us/library/ms180958(v=vs.85).aspx)
* [How to: Envelope a Message for One Recipient](https://msdn.microsoft.com/en-us/library/ms180959(v=vs.85).aspx)
* [How to: Envelope a Message for Multiple Recipients](https://msdn.microsoft.com/en-us/library/ms180960(v=vs.85).aspx)
* [How to: Sign and Envelop a Message](https://msdn.microsoft.com/en-us/library/ms180961(v=vs.85).aspx)
* [How to: Export and Import a Public Key Certificate](https://msdn.microsoft.com/en-us/library/ms180963(v=vs.85).aspx)
* [How to: Add an Authenticated Attribute to a Signed Message](https://msdn.microsoft.com/en-us/library/ms180964(v=vs.85).aspx)
* [How to: Access Hardware Encryption Devices](https://msdn.microsoft.com/en-us/library/ms229931(v=vs.85).aspx)

## Security How-to Topics

**.NET Framework 4**

The following list includes links to the How-to and Walkthrough topics found in the conceptual documentation for security.

**Code Access Security**

* [How to: Request Permission for a Named Permission Set](https://msdn.microsoft.com/en-us/library/0d005ted(v=vs.100).aspx)
* [How to: Make APTCA Assemblies Inaccessible to Partially Trusted Code](https://msdn.microsoft.com/en-us/library/bb397854(v=vs.100).aspx)

**Role-based Security**

* [How to: Create a WindowsPrincipal Object](https://msdn.microsoft.com/en-us/library/t6547wf1(v=vs.100).aspx)
* [How to: Create GenericPrincipal and GenericIdentity Objects](https://msdn.microsoft.com/en-us/library/y9dd5fx0(v=vs.100).aspx)
* [How to: Perform Imperative Security Checks](https://msdn.microsoft.com/en-us/library/dc8ztsad(v=vs.100).aspx)

**Cryptographic Services**

* [How to: Store Asymmetric Keys in a Key Container](https://msdn.microsoft.com/en-us/library/tswxhw92(v=vs.100).aspx)
* [How to: Encrypt XML Elements with Symmetric Keys](https://msdn.microsoft.com/en-us/library/sb7w85t6(v=vs.100).aspx)
* [How to: Decrypt XML Elements with Symmetric Keys](https://msdn.microsoft.com/en-us/library/ms229740(v=vs.100).aspx)
* [How to: Encrypt XML Elements with Asymmetric Keys](https://msdn.microsoft.com/en-us/library/ms229746(v=vs.100).aspx)
* [How to: Decrypt XML Elements with Asymmetric Keys](https://msdn.microsoft.com/en-us/library/ms229919(v=vs.100).aspx)
* [How to: Encrypt XML Elements with X.509 Certificates](https://msdn.microsoft.com/en-us/library/ms229744(v=vs.100).aspx)
* [How to: Decrypt XML Elements with X.509 Certificates](https://msdn.microsoft.com/en-us/library/ms229943(v=vs.100).aspx)
* [How to: Sign XML Documents with Digital Signatures](https://msdn.microsoft.com/en-us/library/ms229745(v=vs.100).aspx)
* [How to: Verify the Digital Signatures of XML Documents](https://msdn.microsoft.com/en-us/library/ms229950(v=vs.100).aspx)
* [How to: Use Data Protection](https://msdn.microsoft.com/en-us/library/ms229741(v=vs.100).aspx)
* [Walkthrough: Creating a Cryptographic Application](https://msdn.microsoft.com/en-us/library/bb397867(v=vs.100).aspx)
* [How to: Access Hardware Encryption Devices](https://msdn.microsoft.com/en-us/library/ms229931(v=vs.100).aspx)

**Security Policy Management**

* [How to: Enable Internet Explorer Security Settings for Managed Execution](https://msdn.microsoft.com/en-us/library/101853ac(v=vs.100).aspx)
* [How to: Add Custom Permissions to Security Policy](https://msdn.microsoft.com/en-us/library/xt8xsee5(v=vs.100).aspx)

**Secure Coding Guidelines**

* [How to: Run Partially Trusted Code in a Sandbox](https://msdn.microsoft.com/en-us/library/bb763046(v=vs.100).aspx)

## Key Security Concepts

**.NET Framework 1.1, 2.0, 3.0, 3.5**

The Microsoft .NET Framework offers code access security and role-based security to help address security concerns about mobile code and to provide support that enables components to determine what users are authorized to do. These security mechanisms use a simple, consistent model so that developers familiar with code access security can easily use role-based security, and vice versa. Both code access security and role-based security are implemented using a common infrastructure supplied by the common language runtime.

|  |
| --- |
| **NoteNote** |
| Starting with the .NET Framework version 4, security transparency is the default enforcement mechanism. Security transparency separates code that runs as part of the application from code that runs as part of the infrastructure. For more information, see [Security-Transparent Code](https://msdn.microsoft.com/en-us/library/ee191569(v=vs.100).aspx). |

Because they use the same model and infrastructure, code access security and role-based security share several underlying concepts, which are described in this section. Make sure that you are familiar with these concepts before reading the documentation for .NET Framework code access security and role-based security.

[Permissions](https://msdn.microsoft.com/en-us/library/5ba4k1c5(v=vs.71).aspx) 1.1, [Security Permissions](https://msdn.microsoft.com/en-us/library/5ba4k1c5(v=vs.80).aspx) 2.0, 3.0, 3.5, 4.0

Describes permission objects and how they are used by the runtime.

[Type Safety and Security](https://msdn.microsoft.com/en-us/library/hbzz1a9a(v=vs.71).aspx) 1.1, 2.0, 3.0, 3.5, 4.0

Describes memory type safety and the security benefits it provides.

[Security Policy](https://msdn.microsoft.com/en-us/library/tha13y5z(v=vs.71).aspx) 1.1, 2.0, 3.0, 3.5

Provides an overview of security policy and how it is used by the runtime.

[Principal](https://msdn.microsoft.com/en-us/library/axt6w9h8(v=vs.71).aspx) 1.1, 2.0, 3.0, 3.5, 4.0

Describes three kinds of principals supported by .NET Framework role-based security.

[Authentication](https://msdn.microsoft.com/en-us/library/syf5yeat(v=vs.71).aspx) 1.1, 2.0, 3.0, 3.5, 4.0

Provides an overview of the authentication process used in .NET Framework role-based security.

[Authorization](https://msdn.microsoft.com/en-us/library/h2ds7dy5(v=vs.71).aspx) 1.1, 2.0, 3.0, 3.5, 4.0

Provides an overview of the authorization process used in .NET Framework role-based security.

[Security Concerns for Internal Virtual and Overloads Overridable Friend Keywords](https://msdn.microsoft.com/en-us/library/heyd8kky(v=vs.71).aspx) 1.1, 2.0, 3.0, 3.5, 4.0

Explains security concerns when using these keywords.

**Related Sections**

[Securing ASP.NET Web Applications](https://msdn.microsoft.com/en-us/library/330a99hc(v=vs.71).aspx)

Describes ASP.NET security in detail and provides instructions for using it in your code.

[Code Access Security](https://msdn.microsoft.com/en-us/library/930b76w0(v=vs.71).aspx)

Describes .NET Framework code access security in detail and provides instructions for using it in your code.

[Role-Based Security](https://msdn.microsoft.com/en-us/library/52kd59t0(v=vs.71).aspx)

Describes .NET Framework role-based security in detail and provides instructions for using it in your code.

[Security Policy Management](https://msdn.microsoft.com/en-us/library/c1k0eed6(v=vs.71).aspx)

Describes the .NET Framework security policy model.

Describes .NET Framework code access security in detail and provides instructions for using it in your code.

[Role-Based Security](https://msdn.microsoft.com/en-us/library/52kd59t0(v=vs.71).aspx)

Describes .NET Framework role-based security in detail and provides instructions for using it in your code.

[ASP.NET Web Application Security](https://msdn.microsoft.com/en-us/library/330a99hc(v=vs.100).aspx)

Describes ASP.NET security in detail and provides instructions for using it in your code.

[Code Access Security](https://msdn.microsoft.com/en-us/library/c5tk9z76(v=vs.100).aspx)

Describes .NET Framework code access security in detail and provides instructions for using it in your code.

[Role-Based Security](https://msdn.microsoft.com/en-us/library/shz8h065(v=vs.100).aspx)

Describes .NET Framework role-based security in detail and provides instructions for using it in your code.

[Security-Transparent Code, Level 2](https://msdn.microsoft.com/en-us/library/dd233102(v=vs.100).aspx)

Describes how security transparency is implemented in the .NET Framework 4.

### Permissions **.NET Framework 1.1, 2.0, 3.0**

The common language runtime allows code to perform only those operations that the code has permission to perform. The runtime uses objects called permissions to implement its mechanism for enforcing restrictions on managed code. The primary uses of permissions are as follows:

* Code can request the permissions it either needs or could use. The .NET Framework security system determines whether such requests are honored. Requests are honored only if the code's evidence merits granting those permissions. Code never receives more permission than the current security settings allow based upon a request. However, code will be granted less permission based upon a request.
* The runtime can grant permissions to code based on characteristics of the code's identity, on the permissions that are requested, and on how much the code is trusted (as determined by security policy set by an administrator). For more information about how the runtime decides which permissions to grant, see [Security Policy](https://msdn.microsoft.com/en-us/library/tha13y5z(v=vs.71).aspx) and [Permission Grants](https://msdn.microsoft.com/en-us/library/abt16x18(v=vs.71).aspx).
* Code can demand that its callers have specific permissions. If you place a demand for a certain permission on your code, all code that uses your code must have that permission to run.

There are three kinds of permissions, each with a specific purpose:

* Code access permissions, which represent access to a protected resource or the ability to perform a protected operation.
* Identity permissions, which indicate that code has credentials that support a particular kind of identity.
* Role-based security permissions, which provide a mechanism for discovering whether a user (or the agent acting on the user's behalf) has a particular identity or is a member of a specified role. [PrincipalPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.principalpermission(v=vs.71).aspx) is the only role-based security permission.

The runtime provides built-in permission classes in several namespaces and also supplies support for designing and implementing custom permission classes.

### Security Permissions

### **.NET Framework 3.5**

Updated: September 2008

The common language runtime allows code to perform only those operations that the code has permission to perform. The runtime uses objects called permissions to implement its mechanism for enforcing restrictions on managed code.

There are three kinds of permissions, and each has a specific purpose:

* [Code Access Permissions](https://msdn.microsoft.com/en-us/library/h846e9b3(v=vs.90).aspx), which represent access to a protected resource or the ability to perform a protected operation.
* [Identity Permissions](https://msdn.microsoft.com/en-us/library/d3wktt6a(v=vs.90).aspx) (a category of code access permissions), which indicate that code has credentials that support a particular kind of identity.
* [Role-Based Security Permissions](https://msdn.microsoft.com/en-us/library/7sxk9k2h(v=vs.90).aspx), which provide a mechanism for discovering whether a user (or the agent acting on the user's behalf) has a particular identity or is a member of a specified role. [PrincipalPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.principalpermission(v=vs.90).aspx) is the only role-based security permission.

Security permissions can be in the form of a permission class ([Imperative Security](https://msdn.microsoft.com/en-us/library/0xkh23z7(v=vs.90).aspx)) or an attribute that represents a permission class ([Declarative Security](https://msdn.microsoft.com/en-us/library/kaacwy28(v=vs.90).aspx)). The base class for security permissions is [CodeAccessPermission](https://msdn.microsoft.com/en-us/library/system.security.codeaccesspermission(v=vs.90).aspx); the base class for security permission attributes is[CodeAccessSecurityAttribute](https://msdn.microsoft.com/en-us/library/system.security.permissions.codeaccesssecurityattribute(v=vs.90).aspx).

An application, in the form of an assembly, is granted a set of permissions. The grants are made by using predefined permission sets. The .NET Framework provides default [NamedPermissionSets](https://msdn.microsoft.com/en-us/library/53dz82ah(v=vs.90).aspx). The grant set determines the permissions the code has available to it. The runtime grants permissions to code based on characteristics of the code's identity, on the permissions that are requested, and on how much the code is trusted (as determined by security policy set by an administrator). For more information about how the runtime decides which permissions to grant, see [Security Policy](https://msdn.microsoft.com/en-us/library/tha13y5z(v=vs.90).aspx) and [Permission Grants](https://msdn.microsoft.com/en-us/library/abt16x18(v=vs.90).aspx).

The primary uses of permissions are as follows:

* Library code can demand that its callers have specific permissions. If you place a [Demand](https://msdn.microsoft.com/en-us/library/1a68f9a5(v=vs.90).aspx) for a permission in your code, all code that uses your code is expected to have that permission to run. Demands can be used to determine whether callers have access to specific resources or to discover the identity of a caller.
* Code can verify that it has the permissions it either needs or could use. Use the [RequestMinimum](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.90).aspx) flag to confirm that code has the permissions it needs to run. For more information, see [How to: Request Minimum Permissions by Using the RequestMinimum Flag](https://msdn.microsoft.com/en-us/library/ms229913(v=vs.90).aspx).
* Code can use permissions to deny access to resources it wants to protect. You can use the ability to deny permissions to protect against inadvertent access by your own code; for example, you can limit file access to a specific location when you accept a file path from user input. However, we do not recommend using permission requests to prohibit access for the purpose of protecting against intentional misuse. Called assemblies, which have the refused permissions in their grant set, can override denied permissions by using the [Assert](https://msdn.microsoft.com/en-us/library/c82hh6x8(v=vs.90).aspx) method. The only way to securely protect resources from untrusted code in referenced assemblies is to execute that code with a grant set that does not include those permissions.
* The runtime provides built-in permission classes in several namespaces and also supplies support for designing and implementing custom permission classes.

[**See Also**](javascript:void(0))

Concepts

[Code Access Permissions](https://msdn.microsoft.com/en-us/library/h846e9b3(v=vs.90).aspx)

[Identity Permissions](https://msdn.microsoft.com/en-us/library/d3wktt6a(v=vs.90).aspx)

[Role-Based Security Permissions](https://msdn.microsoft.com/en-us/library/7sxk9k2h(v=vs.90).aspx)

Other Resources

[Key Security Concepts](https://msdn.microsoft.com/en-us/library/z164t8hs(v=vs.90).aspx)

[**Change History**](javascript:void(0))

|  |  |  |
| --- | --- | --- |
| **Date** | **History** | **Reason** |
| September 2008 | Added new information. | Customer feedback. |

### Security Permissions

**.NET Framework 4, 4.5**

The common language runtime allows code to perform only those operations that the code has permission to perform. The runtime uses objects called permissions to enforce restrictions on managed code. The runtime provides built-in permission classes in several namespaces and also supports designing and implementing custom permission classes.

There are two kinds of permissions, and each has a specific purpose:

* [Code Access Permissions](https://msdn.microsoft.com/en-us/library/h846e9b3(v=vs.100).aspx) represent access to a protected resource or the ability to perform a protected operation.
* [Role-Based Security Permissions](https://msdn.microsoft.com/en-us/library/7sxk9k2h(v=vs.100).aspx) provide a mechanism for discovering whether a user (or the agent acting on the user's behalf) has a particular identity or is a member of a specified role. [PrincipalPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.principalpermission(v=vs.100).aspx) is the only role-based security permission.

Security permissions can be in the form of a permission class ([imperative security](https://msdn.microsoft.com/en-us/library/0xkh23z7(v=vs.100).aspx)) or an attribute that represents a permission class ([declarative security](https://msdn.microsoft.com/en-us/library/kaacwy28(v=vs.100).aspx)). The base class for security permissions is [System.Security.CodeAccessPermission](https://msdn.microsoft.com/en-us/library/system.security.codeaccesspermission(v=vs.100).aspx); the base class for security permission attributes is [System.Security.Permissions.CodeAccessSecurityAttribute](https://msdn.microsoft.com/en-us/library/system.security.permissions.codeaccesssecurityattribute(v=vs.100).aspx).

An application, in the form of an assembly, is granted a set of permissions at the time it is loaded into an application domain. The grants are typically made by using predefined permission sets that are determined by the [SecurityManager.GetStandardSandbox](https://msdn.microsoft.com/en-us/library/dd414122(v=vs.100).aspx) method. The grant set determines the permissions the code has available to it. The runtime grants permissions based on the code's origin location (for example, the local machine, local intranet, or the Internet). Code can also be granted special permissions if it is loaded into a sandbox. For more information about running code in a sandbox, see [How to: Run Partially Trusted Code in a Sandbox](https://msdn.microsoft.com/en-us/library/bb763046(v=vs.100).aspx).

The primary uses of permissions are as follows:

* Library code can demand that its callers have specific permissions. If you place a [Demand](https://msdn.microsoft.com/en-us/library/1a68f9a5(v=vs.100).aspx) for a permission in your code, all code that uses your code is expected to have that permission to run. Demands can be used to determine whether callers have access to specific resources or to discover the identity of a caller.
* Code can use permissions to deny access to resources it wants to protect. You can use [SecurityAction.PermitOnly](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx) to specify a limited permission set, implicitly denying all other permissions. However, we do not recommend using [PermitOnly](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx) to prohibit access for the purpose of protecting against intentional misuse. Called assemblies, which have the implicitly refused permissions in their grant set, can override denied permissions by performing an [SecurityAction.Assert](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx) for any permission they want to use. For example, if you permitted only [UIPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.uipermission(v=vs.100).aspx) and called an assembly that inherently has [FileIOPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.fileiopermission(v=vs.100).aspx), the assembly can simply do an [Assert](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx) for[FileIOPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.fileiopermission(v=vs.100).aspx) and perform file operations. The only way to securely protect resources from untrusted code in referenced assemblies is to execute that code with a grant set that does not include those permissions.

[**See Also**](javascript:void(0))

Concepts

[Code Access Permissions](https://msdn.microsoft.com/en-us/library/h846e9b3(v=vs.100).aspx)

[Role-Based Security Permissions](https://msdn.microsoft.com/en-us/library/7sxk9k2h(v=vs.100).aspx)

Other Resources

[Key Security Concepts](https://msdn.microsoft.com/en-us/library/z164t8hs(v=vs.100).aspx)

#### Code Access Permissions

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4.0, 4.5**

Code access permissions are permission objects that are used to help protect resources and operations from unauthorized use. They are a fundamental part of the common language runtime's mechanism for enforcing security restrictions on managed code.

Each code access permission represents one of the following rights:

* The right to access a protected resource, such as files or environment variables.
* The right to perform a protected operation, such as accessing unmanaged code.

All code access permissions can be requested or demanded by code, and the runtime decides which permissions, if any, to grant the code.

Each code access permission derives from the [CodeAccessPermission](https://msdn.microsoft.com/en-us/library/system.security.codeaccesspermission(v=vs.71).aspx) class, which means that all code access permissions have methods in common, such as **Demand**, **Assert**, **Deny**, **PermitOnly**, **IsSubsetOf**, **Intersect**, and **Union**.

The .NET Framework provides the following code access permissions.

|  |  |
| --- | --- |
| **Permission class name** | **Right represented** |
| [AspNetHostingPermission](https://msdn.microsoft.com/en-us/library/system.web.aspnethostingpermission(v=vs.71).aspx) | Access resources in ASP.NET-hosted environments. |
| [DirectoryServicesPermission](https://msdn.microsoft.com/en-us/library/system.directoryservices.directoryservicespermission(v=vs.71).aspx) | Access to the [System.DirectoryServices](https://msdn.microsoft.com/en-us/library/system.directoryservices(v=vs.71).aspx) classes. |
| [DnsPermission](https://msdn.microsoft.com/en-us/library/system.net.dnspermission(v=vs.71).aspx) | Access to Domain Name System (DNS). |
| [EnvironmentPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.environmentpermission(v=vs.71).aspx) | Read or write environment variables. |
| [EventLogPermission](https://msdn.microsoft.com/en-us/library/system.diagnostics.eventlogpermission(v=vs.71).aspx) | Read or write access to event log services. |
| [FileDialogPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.filedialogpermission(v=vs.71).aspx) | Access files that have been selected by the user in an **Open** dialog box. |
| [FileIOPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.fileiopermission(v=vs.71).aspx) | Read, append, or write files or directories. |
| [IsolatedStorageFilePermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.isolatedstoragefilepermission(v=vs.71).aspx) | Access isolated storage, which is storage that is associated with a specific user and with some aspect of the code's identity, such as its Web site, publisher, or signature. |
| [MessageQueuePermission](https://msdn.microsoft.com/en-us/library/system.messaging.messagequeuepermission(v=vs.71).aspx) | Access message queues through the managed Microsoft Message Queuing (MSMQ) interfaces. |
| [OdbcPermission](https://msdn.microsoft.com/en-us/library/system.data.odbc.odbcpermission(v=vs.71).aspx) | Access an ODBC data source. |
| [OleDbPermission](https://msdn.microsoft.com/en-us/library/system.data.oledb.oledbpermission(v=vs.71).aspx) | Access databases using OLE DB. |
| [OraclePermission](https://msdn.microsoft.com/en-us/library/system.data.oracleclient.oraclepermission(v=vs.71).aspx) | Access an Oracle database. |
| [PerformanceCounterPermission](https://msdn.microsoft.com/en-us/library/system.diagnostics.performancecounterpermission(v=vs.71).aspx) | Access performance counters. |
| [PrintingPermission](https://msdn.microsoft.com/en-us/library/system.drawing.printing.printingpermission(v=vs.71).aspx) | Access printers. |
| [ReflectionPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.reflectionpermission(v=vs.71).aspx) | Discover information about a type at run time. |
| [RegistryPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.registrypermission(v=vs.71).aspx) | Read, write, create, or delete registry keys and values. |
| [SecurityPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermission(v=vs.71).aspx) | Execute, assert permissions, call into unmanaged code, skip verification, and other rights. |
| [ServiceControllerPermission](https://msdn.microsoft.com/en-us/library/system.serviceprocess.servicecontrollerpermission(v=vs.71).aspx) | Access running or stopped services. |
| [SocketPermission](https://msdn.microsoft.com/en-us/library/system.net.socketpermission(v=vs.71).aspx) | Make or accept connections on a transport address. |
| [SqlClientPermission](https://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlclientpermission(v=vs.71).aspx) | Access SQL databases. |
| [UIPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.uipermission(v=vs.71).aspx) | Access user interface functionality. |
| [WebPermission](https://msdn.microsoft.com/en-us/library/system.net.webpermission(v=vs.71).aspx) | Make or accept connections on a Web address. |

Additionally, the .NET Framework provides the following abstract classes that you can use to create your own custom permissions.

|  |  |
| --- | --- |
| **Permission class name** | **Right represented** |
| [DBDataPermission](https://msdn.microsoft.com/en-us/library/system.data.common.dbdatapermission(v=vs.71).aspx) | Access a database. |
| [IsolatedStoragePermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.isolatedstoragepermission(v=vs.71).aspx) | Access isolated storage. |
| [ResourcePermissionBase](https://msdn.microsoft.com/en-us/library/system.security.permissions.resourcepermissionbase(v=vs.71).aspx) | Access system resources. |

#### Identity Permissions

**.NET Framework 1.1, 2.0, 3.0, 3.5**

Identity permissions represent characteristics that identify an assembly. The common language runtime grants identity permissions to an assembly based on the information it obtains about the assembly. This information, called evidence, is provided by the loader or a [trusted host](https://msdn.microsoft.com/en-us/library/6700e49f(v=vs.71).aspx) and can include items such as the digital signature of the assembly or the Web site where it originates. Each identity permission represents a particular kind of evidence that an assembly must have in order to run. For example, one permission represents the [strong name](https://msdn.microsoft.com/en-us/library/wd40t7ad(v=vs.71).aspx)an assembly must have, another represents the Web site where the code must have originated, and so on.

Because the identity permissions have a set of functionality in common with code access permissions, they are derived from the same base class as the code access permissions, [CodeAccessPermission](https://msdn.microsoft.com/en-us/library/system.security.codeaccesspermission(v=vs.71).aspx).

Identity permissions help protect code from unauthorized access. The runtime grants identity permissions when the assembly is loaded based on the evidence that is provided. Although identity permissions can be requested, they cannot be granted unless the code has the proper identity evidence. Identity permissions can also be demanded.

The .NET Framework provides the following identity permissions.

|  |  |
| --- | --- |
| **Class name** | **Identity represented** |
| [PublisherIdentityPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.publisheridentitypermission(v=vs.71).aspx) | The software publisher's digital signature. |
| [SiteIdentityPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.siteidentitypermission(v=vs.71).aspx) | The Web site where the code originated. |
| [StrongNameIdentityPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.strongnameidentitypermission(v=vs.71).aspx) | The [strong name](https://msdn.microsoft.com/en-us/library/wd40t7ad(v=vs.71).aspx) of the assembly. |
| [URLIdentityPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.urlidentitypermission(v=vs.71).aspx) | The URL where the code originated (including the protocol prefix — http, https, ftp, and so on). |
| [ZoneIdentityPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.zoneidentitypermission(v=vs.71).aspx) | The zone where the code originated. For more information, see [System.Security.SecurityZone](https://msdn.microsoft.com/en-us/library/system.security.securityzone(v=vs.71).aspx). |

#### Role-Based Security Permissions

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4.0, 4.5**

[PrincipalPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.principalpermission(v=vs.71).aspx) is a role-based security permission that can be used to determine whether a user has a specified identity or is a member of a specified role. **PrincipalPermission** is the only role-based security permission supplied by the .NET Framework class library.

##### PrincipalPermission Class

**.NET Framework 1.1,2.0, 3.0, 3.5, 4, 4.5, 4.6**

Allows checks against the active principal (see [IPrincipal](https://msdn.microsoft.com/en-us/library/system.security.principal.iprincipal(v=vs.71).aspx)) using the language constructs defined for both declarative and imperative security actions. This class cannot be inherited.

[**Syntax**](javascript:void(0))

C#

[SerializableAttribute]

[ComVisibleAttribute(true)]

public sealed class PrincipalPermission : IPermission,

ISecurityEncodable, IUnrestrictedPermission

The PrincipalPermission type exposes the following members.

[Constructors](javascript:void(0))

|  |  |  |
| --- | --- | --- |
|  | **Name** | **Description** |
| Public method | [PrincipalPermission(PermissionState)](https://msdn.microsoft.com/en-us/library/230thh38(v=vs.110).aspx) | Initializes a new instance of the PrincipalPermission class with the specified[PermissionState](https://msdn.microsoft.com/en-us/library/system.security.permissions.permissionstate(v=vs.110).aspx). |
| Public method | [PrincipalPermission(String, String)](https://msdn.microsoft.com/en-us/library/zxy9x672(v=vs.110).aspx) | Initializes a new instance of the PrincipalPermission class for the specified *name*and *role*. |
| Public method | [PrincipalPermission(String, String, Boolean)](https://msdn.microsoft.com/en-us/library/3ht8409h(v=vs.110).aspx) | Initializes a new instance of the PrincipalPermission class for the specified *name*,*role*, and authentication status. |

[**Methods**](javascript:void(0))

|  |  |  |
| --- | --- | --- |
|  | **Name** | **Description** |
| Public method | [Copy](https://msdn.microsoft.com/en-us/library/0zye58ay(v=vs.110).aspx) | Creates and returns an identical copy of the current permission. |
| Public method | [Demand](https://msdn.microsoft.com/en-us/library/t5cyk9b0(v=vs.110).aspx) | Determines at run time whether the current principal matches the principal specified by the current permission. |
| Public method | [Equals](https://msdn.microsoft.com/en-us/library/t5tw73ff(v=vs.110).aspx) | Determines whether the specified PrincipalPermission object is equal to the current PrincipalPermission.(Overrides [Object.Equals(Object)](https://msdn.microsoft.com/en-us/library/bsc2ak47(v=vs.110).aspx).) |
| Public method | [FromXml](https://msdn.microsoft.com/en-us/library/b3sd1794(v=vs.110).aspx) | Reconstructs a permission with a specified state from an XML encoding. |
| Public method | [GetHashCode](https://msdn.microsoft.com/en-us/library/6es82tfx(v=vs.110).aspx) | Gets a hash code for the PrincipalPermission object that is suitable for use in hashing algorithms and data structures such as a hash table. (Overrides [Object.GetHashCode()](https://msdn.microsoft.com/en-us/library/zdee4b3y(v=vs.110).aspx).) |
| Public method | [GetType](https://msdn.microsoft.com/en-us/library/dfwy45w9(v=vs.110).aspx) | Gets the [Type](https://msdn.microsoft.com/en-us/library/system.type(v=vs.110).aspx) of the current instance. (Inherited from [Object](https://msdn.microsoft.com/en-us/library/system.object(v=vs.110).aspx).) |
| Public method | [Intersect](https://msdn.microsoft.com/en-us/library/hs8b07b2(v=vs.110).aspx) | Creates and returns a permission that is the intersection of the current permission and the specified permission. |
| Public method | [IsSubsetOf](https://msdn.microsoft.com/en-us/library/k3t0d94x(v=vs.110).aspx) | Determines whether the current permission is a subset of the specified permission. |
| Public method | [IsUnrestricted](https://msdn.microsoft.com/en-us/library/wfx08hsc(v=vs.110).aspx) | Returns a value indicating whether the current permission is unrestricted. |
| Public method | [ToString](https://msdn.microsoft.com/en-us/library/f0kzd01y(v=vs.110).aspx) | Creates and returns a string representing the current permission. (Overrides [Object.ToString()](https://msdn.microsoft.com/en-us/library/7bxwbwt2(v=vs.110).aspx).) |
| Public method | [ToXml](https://msdn.microsoft.com/en-us/library/bedasy0k(v=vs.110).aspx) | Creates an XML encoding of the permission and its current state. |
| Public method | [Union](https://msdn.microsoft.com/en-us/library/t8swcd04(v=vs.110).aspx) | Creates a permission that is the union of the current permission and the specified permission. |

**Thread Safety**

Any public static (**Shared** in Visual Basic) members of this type are thread safe. Any instance members are not guaranteed to be thread safe.

**Remarks**

By passing identity information (user name and role) to the constructor, **PrincipalPermission** can be used to demand that the identity of the active principal matches this information.

To match the active [IPrincipal](https://msdn.microsoft.com/en-us/library/system.security.principal.iprincipal(v=vs.71).aspx) and associated [IIdentity](https://msdn.microsoft.com/en-us/library/system.security.principal.iidentity(v=vs.71).aspx), both the specified identity and role must match. If a null reference (**Nothing** in Visual Basic) identity string is used, it is interpreted as a request to match any identity. Use of a null reference (**Nothing**) role string will match any role. By implication, passing a null reference (**Nothing**) parameter for *name* or *role* to **PrincipalPermission** will match the identity and roles in any **IPrincipal**. It is also possible to construct a **PrincipalPermission** that only determines whether the **IIdentity** represents an authenticated or unauthenticated entity. In this case, *name* and *role* are ignored.

Unlike most other permissions, **PrincipalPermission** does not extend [CodeAccessPermission](https://msdn.microsoft.com/en-us/library/system.security.codeaccesspermission(v=vs.71).aspx). It does, however, implement the [IPermission](https://msdn.microsoft.com/en-us/library/system.security.ipermission(v=vs.71).aspx)interface. This is because **PrincipalPermission** is not a code access permission; that is, it is not granted based on the identity of the executing assembly. Instead, it allows code to perform actions ([Demand](https://msdn.microsoft.com/en-us/library/t5cyk9b0(v=vs.71).aspx), [Union](https://msdn.microsoft.com/en-us/library/t8swcd04(v=vs.71).aspx), [Intersect](https://msdn.microsoft.com/en-us/library/hs8b07b2(v=vs.71).aspx), and so on) against the current user identity in a manner consistent with the way those actions are performed for code access and code identity permissions.

**Example**

[Visual Basic, C#, C++] The following example creates two **PrincipalPermission** objects representing two different administrative users, forms the union of the two, and makes a demand. [Demand](https://msdn.microsoft.com/en-us/library/t5cyk9b0(v=vs.71).aspx) will succeed only if the active implementation of [IPrincipal](https://msdn.microsoft.com/en-us/library/system.security.principal.iprincipal(v=vs.71).aspx) represents either user Bob in the role of Manager or user Louise in the role of Supervisor.

### [Type Safety and Security](https://msdn.microsoft.com/en-us/library/hbzz1a9a(v=vs.71).aspx)

Type Safety and Security

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4.0, 4.5**

Type-safe code accesses only the memory locations it is authorized to access. (For this discussion, type safety specifically refers to memory type safety and should not be confused with type safety in a broader respect.) For example, type-safe code cannot read values from another object's private fields. It accesses types only in well-defined, allowable ways.

During just-in-time (JIT) compilation, an optional verification process examines the metadata and Microsoft intermediate language (MSIL) of a method to be JIT-compiled into native machine code to verify that they are type safe. This process is skipped if the code has permission to bypass verification. For more information about verification, see [Compiling MSIL to Native Code](https://msdn.microsoft.com/en-us/library/ht8ecch6(v=vs.71).aspx).

Although verification of type safety is not mandatory to run managed code, type safety plays a crucial role in assembly isolation and security enforcement. When code is type safe, the common language runtime can completely isolate assemblies from each other. This isolation helps ensure that assemblies cannot adversely affect each other and it increases application reliability. Type-safe components can execute safely in the same process even if they are trusted at different levels. When code is not type safe, unwanted side effects can occur. For example, the runtime cannot prevent unsafe code from calling into native (unmanaged) code and performing malicious operations. When code is type safe, the runtime's security enforcement mechanism ensures that it does not access native code unless it has permission to do so. All code that is not type safe must have been granted [SecurityPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermission(v=vs.71).aspx) with the passed enum member [SkipVerification](https://msdn.microsoft.com/en-us/library/tht37fky(v=vs.71).aspx) to run.

See Also

[Key Security Concepts](https://msdn.microsoft.com/en-us/library/z164t8hs(v=vs.71).aspx) | [Writing Verifiably Type-Safe Code](https://msdn.microsoft.com/en-us/library/01k04eaf(v=vs.71).aspx)

### Security Policy ****.NET Framework 1.1, 2.0, 3.0, 3.5****

Security policy is the configurable set of rules that the common language runtime follows when it decides what it will allow code to do. Administrators set security policy and the runtime enforces it. The runtime helps ensure that code can access only the resources and call only the code allowed by security policy.

Whenever an attempt is made to load an assembly, the runtime uses security policy to determine which permissions to grant to the assembly. After examining information, called evidence, that describes the identity of the assembly, the runtime uses security policy to decide how much the code is trusted and, therefore, what permissions to grant to that assembly. Evidence includes, but is not limited to, the code's publisher, its site, and its zone. Security policy also determines which permissions to grant to application domains.

See Also

[Key Security Concepts](https://msdn.microsoft.com/en-us/library/z164t8hs(v=vs.71).aspx) | [Security Policy Management](https://msdn.microsoft.com/en-us/library/c1k0eed6(v=vs.71).aspx)

### .NET Framework Security Policy ****.NET Framework 4****

Security policy is the configurable set of rules that the common language runtime follows when it decides what it will allow code to do. Administrators set security policy and the runtime enforces it. The runtime helps ensure that code can access only the resources and call only the code allowed by security policy.

|  |
| --- |
| **NoteNote** |
| In the .NET Framework version 4, the common language runtime (CLR) is moving away from providing security policy for computers. For more information, see [Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx). |

Whenever an attempt is made to load an assembly, the runtime uses security policy to determine which permissions to grant to the assembly. After examining information, called evidence, that describes the identity of the assembly, the runtime uses security policy to decide how much the code is trusted and, therefore, what permissions to grant to that assembly. Evidence includes, but is not limited to, the code's publisher, its site, and its zone. Security policy also determines which permissions to grant to application domains.

### Principal

**.NET Framework 1.1, 2.0, 3.0, 4.0, 4.5**

A principal represents the identity and role of a user and acts on the user's behalf. Role-based security in the .NET Framework supports three kinds of principals:

* Generic principals represent users and roles that exist independent of Windows NT and Windows 2000 users and roles.
* Windows principals represent Windows users and their roles (or their Windows NT and Windows 2000 groups). A Windows principal can impersonate another user, which means the principal can access a resource on a user's behalf while presenting the identity that belongs to that user.
* Custom principals can be defined by an application in any way that is needed for that particular application. They can extend the basic notion of the principal's identity and roles.

### Authentication

**.NET Framework 1.1, 2.0, 3.0**

Authentication is the process of discovering and verifying the identity of a principal by examining the user's credentials and validating those credentials against some authority. The information obtained during authentication is directly usable by your code. That is, once the identity of the principal is discovered, you can use .NET Framework role-based security to determine whether to allow that principal to access your code.

A variety of authentication mechanisms are used today, many of which can be used with .NET Framework role-based security. Some of the most commonly used mechanisms are basic, digest, Passport, operating system (such as NTLM or Kerberos), or application-defined mechanisms.

### Authentication

**.NET Framework 3.5**

Updated: May 2009

Authentication is the process of discovering and verifying the identity of a principal by examining the user's credentials and validating those credentials against some authority. The information obtained during authentication is directly usable by your code. You can also use .NET Framework role-based security to authenticate the current user and to determine whether to allow that principal to access your code. See the overloads of the [WindowsPrincipal.IsInRole](https://msdn.microsoft.com/en-us/library/system.security.principal.windowsprincipal.isinrole(v=vs.90).aspx) method for examples of how to authenticate the principal for specific roles. For example, you can use the [WindowsPrincipal.IsInRole(String)](https://msdn.microsoft.com/en-us/library/fs485fwh(v=vs.90).aspx) method to determine whether the current user is a member of the Administrators group.

A variety of authentication mechanisms are used today, many of which can be used with .NET Framework role-based security. Some of the most commonly used mechanisms are basic, digest, Passport, operating system (such as NTLM or Kerberos), or application-defined mechanisms.

[**Example**](javascript:void(0))

The following example requires that the active principal be an administrator. The *name* parameter is null, which allows any user who is an administrator to pass the demand.

|  |
| --- |
| **NoteNote:** |
| In Windows Vista, User Account Control (UAC) determines the privileges of a user. If you are a member of the Built-in Administrators group, you are assigned two run-time access tokens: a standard user access token and an administrator access token. By default, you are in the standard user role. To execute the code that requires you to be an administrator, you must first elevate your privileges from standard user to administrator. You can do this when you start an application by right-clicking the application icon and indicating that you want to run as an administrator. |

[**See Also**](javascript:void(0))

Other Resources

[Key Security Concepts](https://msdn.microsoft.com/en-us/library/z164t8hs(v=vs.90).aspx)

[Role-Based Security](https://msdn.microsoft.com/en-us/library/52kd59t0(v=vs.90).aspx)

[**Change History**](javascript:void(0))

|  |  |  |
| --- | --- | --- |
| **Date** | **History** | **Reason** |
| May 2009 | Added link to [WindowsPrincipal.IsInRole](https://msdn.microsoft.com/en-us/library/system.security.principal.windowsprincipal.isinrole(v=vs.90).aspx) for examples. | Customer feedback. |
| September 2008 | Added examples. | Customer feedback. |

### Authentication

**.NET Framework 4, 4.5**

Authentication is the process of discovering and verifying the identity of a principal by examining the user's credentials and validating those credentials against some authority. The information obtained during authentication is directly usable by your code. You can also use .NET Framework role-based security to authenticate the current user and to determine whether to allow that principal to access your code. See the overloads of the [WindowsPrincipal.IsInRole](https://msdn.microsoft.com/en-us/library/system.security.principal.windowsprincipal.isinrole(v=vs.100).aspx) method for examples of how to authenticate the principal for specific roles. For example, you can use the [WindowsPrincipal.IsInRole(String)](https://msdn.microsoft.com/en-us/library/fs485fwh(v=vs.100).aspx) overload to determine if the current user is a member of the Administrators group.

A variety of authentication mechanisms are used today, many of which can be used with .NET Framework role-based security. Some of the most commonly used mechanisms are basic, digest, Passport, operating system (such as NTLM or Kerberos), or application-defined mechanisms.

[**Example**](javascript:void(0))

The following example requires that the active principal be an administrator. The *name* parameter is **null**, which allows any user who is an administrator to pass the demand.

|  |
| --- |
| **NoteNote** |
| In Windows Vista, User Account Control (UAC) determines the privileges of a user. If you are a member of the Built-in Administrators group, you are assigned two run-time access tokens: a standard user access token and an administrator access token. By default, you are in the standard user role. To execute the code that requires you to be an administrator, you must first elevate your privileges from standard user to administrator. You can do this when you start an application by right-clicking the application icon and indicating that you want to run as an administrator. |

C#

[**C++**](https://msdn.microsoft.com/en-us/library/syf5yeat(v=vs.100).aspx?cs-save-lang=1&cs-lang=cpp#code-snippet-1)

[**VB**](https://msdn.microsoft.com/en-us/library/syf5yeat(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

using System;

using System.Threading;

using System.Security.Permissions;

using System.Security.Principal;

class SecurityPrincipalDemo

{

public static void Main()

{

AppDomain.CurrentDomain.SetPrincipalPolicy(PrincipalPolicy.WindowsPrincipal);

PrincipalPermission principalPerm = new PrincipalPermission(null, "Administrators");

principalPerm.Demand();

Console.WriteLine("Demand succeeded.");

}

}

The following example demonstrates how to determine the identity of the principal and the roles available to the principal. An application of this example might be to confirm that the current user is in a role you allow for using your application.

C#

[**C++**](https://msdn.microsoft.com/en-us/library/syf5yeat(v=vs.100).aspx?cs-save-lang=1&cs-lang=cpp#code-snippet-2)

[**VB**](https://msdn.microsoft.com/en-us/library/syf5yeat(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

using System;

using System.Threading;

using System.Security.Permissions;

using System.Security.Principal;

class SecurityPrincipalDemo

{

public static void DemonstrateWindowsBuiltInRoleEnum()

{

AppDomain myDomain = Thread.GetDomain();

myDomain.SetPrincipalPolicy(PrincipalPolicy.WindowsPrincipal);

WindowsPrincipal myPrincipal = (WindowsPrincipal)Thread.CurrentPrincipal;

Console.WriteLine("{0} belongs to: ", myPrincipal.Identity.Name.ToString());

Array wbirFields = Enum.GetValues(typeof(WindowsBuiltInRole));

foreach (object roleName in wbirFields)

{

try

{

// Cast the role name to a RID represented by the WindowsBuildInRole value.

Console.WriteLine("{0}? {1}.", roleName,

myPrincipal.IsInRole((WindowsBuiltInRole)roleName));

Console.WriteLine("The RID for this role is: " + ((int)roleName).ToString());

}

catch (Exception)

{

Console.WriteLine("{0}: Could not obtain role for this RID.",

roleName);

}

}

// Get the role using the string value of the role.

Console.WriteLine("{0}? {1}.", "Administrators",

myPrincipal.IsInRole("BUILTIN\\" + "Administrators"));

Console.WriteLine("{0}? {1}.", "Users",

myPrincipal.IsInRole("BUILTIN\\" + "Users"));

// Get the role using the WindowsBuiltInRole enumeration value.

Console.WriteLine("{0}? {1}.", WindowsBuiltInRole.Administrator,

myPrincipal.IsInRole(WindowsBuiltInRole.Administrator));

// Get the role using the WellKnownSidType.

SecurityIdentifier sid = new SecurityIdentifier(WellKnownSidType.BuiltinAdministratorsSid, null);

Console.WriteLine("WellKnownSidType BuiltinAdministratorsSid {0}? {1}.", sid.Value, myPrincipal.IsInRole(sid));

}

public static void Main()

{

DemonstrateWindowsBuiltInRoleEnum();

}

}

[**See Also**](javascript:void(0))

Other Resources

[Key Security Concepts](https://msdn.microsoft.com/en-us/library/z164t8hs(v=vs.100).aspx)

[Role-Based Security](https://msdn.microsoft.com/en-us/library/shz8h065(v=vs.100).aspx)

### Authorization

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4, 4.5**

Authorization is the process of determining whether a principal is allowed to perform a requested action. Authorization occurs after authentication and uses information about the principal's identity and roles to determine what resources the principal can access. You can use .NET Framework role-based security to implement authorization.

See Also

[Key Security Concepts](https://msdn.microsoft.com/en-us/library/z164t8hs(v=vs.71).aspx) | [Role-Based Security](https://msdn.microsoft.com/en-us/library/52kd59t0(v=vs.71).aspx)

### Security Concerns for Internal Virtual and Overloads Overridable Friend Keywords

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4., 4.5**

You should never base the security of your application on a member that is marked with the internalvirtual modifier in C# (the [Overloads](https://msdn.microsoft.com/en-us/library/ybxdk47f(v=vs.71).aspx)[Overridable](https://msdn.microsoft.com/en-us/library/zcfd2sa9(v=vs.71).aspx)[Friend](https://msdn.microsoft.com/en-us/library/aa445159(v=vs.71).aspx)modifier in Visual Basic). Although members marked with these modifiers can only be overridden by other members within the current assembly, this rule is enforced only by the C# and Visual Basic languages. The runtime does not enforce this rule. It is therefore possible to override members marked as **internal virtual**in C# and **Overloads Overridable Friend** in Visual Basic using Microsoft Intermediate Language, or any other language that does not enforce this rule.

See Also

[Key Security Concepts](https://msdn.microsoft.com/en-us/library/z164t8hs(v=vs.71).aspx) | [Role-Based Security](https://msdn.microsoft.com/en-us/library/52kd59t0(v=vs.71).aspx)

## Key Security Concepts

**.NET Framework 4.6 and 4.5**

The Microsoft .NET Framework offers security transparency, code access security and role-based security to help address security concerns about mobile code and to provide support that enables components to determine what users are authorized to do. These security mechanisms use a simple, consistent model so that developers familiar with code access security can easily use role-based security, and vice versa. Both code access security and role-based security are implemented using a common infrastructure supplied by the common language runtime.

|  |
| --- |
| **Note Note** |
| Starting with the .NET Framework 4, security transparency is the default enforcement mechanism. Security transparency separates code that runs as part of the application from code that runs as part of the infrastructure. For more information, see [Security-Transparent Code](https://msdn.microsoft.com/en-us/library/ee191569(v=vs.110).aspx). |

Because they use the same model and infrastructure, code access security and role-based security share several underlying concepts, which are described in this section. Make sure that you are familiar with these concepts before reading the documentation for .NET Framework code access security and role-based security.

### [Security permissions](javascript:void(0))

The common language runtime allows code to perform only those operations that the code has permission to perform. The runtime uses objects called permissions to enforce restrictions on managed code. The runtime provides built-in permission classes in several namespaces and also supports designing and implementing custom permission classes.

There are two kinds of permissions, and each has a specific purpose:

* Code access permissions represent access to a protected resource or the ability to perform a protected operation.
* Role-based security permissions provide a mechanism for discovering whether a user (or the agent acting on the user's behalf) has a particular identity or is a member of a specified role. [PrincipalPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.principalpermission(v=vs.110).aspx) is the only role-based security permission.

Security permissions can be in the form of a permission class (imperative security) or an attribute that represents a permission class (declarative security). The base class for security permissions is [System.Security.CodeAccessPermission](https://msdn.microsoft.com/en-us/library/system.security.codeaccesspermission(v=vs.110).aspx); the base class for security permission attributes is [System.Security.Permissions.CodeAccessSecurityAttribute](https://msdn.microsoft.com/en-us/library/system.security.permissions.codeaccesssecurityattribute(v=vs.110).aspx).

An application, in the form of an assembly, is granted a set of permissions at the time it is loaded into an application domain. The grants are typically made by using predefined permission sets that are determined by the [SecurityManager.GetStandardSandbox](https://msdn.microsoft.com/en-us/library/dd414122(v=vs.110).aspx) method. The grant set determines the permissions the code has available to it. The runtime grants permissions based on the code's origin location (for example, the local machine, local intranet, or the Internet). Code can also be granted special permissions if it is loaded into a sandbox. For more information about running code in a sandbox, see [How to: Run Partially Trusted Code in a Sandbox](https://msdn.microsoft.com/en-us/library/bb763046(v=vs.110).aspx).

The primary uses of permissions are as follows:

* Library code can demand that its callers have specific permissions. If you place a [Demand](https://msdn.microsoft.com/en-us/library/1a68f9a5(v=vs.110).aspx) for a permission in your code, all code that uses your code is expected to have that permission to run. Demands can be used to determine whether callers have access to specific resources or to discover the identity of a caller.
* Code can use permissions to deny access to resources it wants to protect. You can use [SecurityAction.PermitOnly](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.110).aspx) to specify a limited permission set, implicitly denying all other permissions. However, we do not recommend using [PermitOnly](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.110).aspx) to prohibit access for the purpose of protecting against intentional misuse. Called assemblies, which have the implicitly refused permissions in their grant set, can override denied permissions by performing an [SecurityAction.Assert](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.110).aspx) for any permission they want to use. For example, if you permitted only [UIPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.uipermission(v=vs.110).aspx) and called an assembly that inherently has [FileIOPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.fileiopermission(v=vs.110).aspx), the assembly can simply do an [Assert](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.110).aspx) for[FileIOPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.fileiopermission(v=vs.110).aspx) and perform file operations. The only way to securely protect resources from untrusted code in referenced assemblies is to execute that code with a grant set that does not include those permissions.

#### [**Code access permissions**](javascript:void(0))

Code access permissions are permission objects that are used to help protect resources and operations from unauthorized use. They are a fundamental part of the common language runtime's mechanism for enforcing security restrictions on managed code.

Each code access permission represents one of the following rights:

* The right to access a protected resource, such as files or environment variables.
* The right to perform a protected operation, such as accessing unmanaged code.

All code access permissions can be requested or demanded by code, and the runtime decides which permissions, if any, to grant the code.

Each code access permission derives from the [CodeAccessPermission](https://msdn.microsoft.com/en-us/library/system.security.codeaccesspermission(v=vs.110).aspx) class, which means that all code access permissions have methods in common, such as **Demand**, **Assert**, **Deny**, **PermitOnly**, **IsSubsetOf**, **Intersect**, and **Union**.

|  |
| --- |
| **Important note Important** |
| In the .NET Framework 4, runtime support has been removed for enforcing the [Deny](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.110).aspx), [RequestMinimum](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.110).aspx), [RequestOptional](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.110).aspx), and[RequestRefuse](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.110).aspx) permission requests. These requests should not be used in code that is based on .NET Framework 4 or later. For more information about this and other changes, see [Security Changes in the .NET Framework](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.110).aspx). |

#### [**Role-based security permissions**](javascript:void(0))

[PrincipalPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.principalpermission(v=vs.110).aspx) is a role-based security permission that can be used to determine whether a user has a specified identity or is a member of a specified role. **PrincipalPermission** is the only role-based security permission supplied by the .NET Framework class library.

### [Type safety and security](javascript:void(0))

Type-safe code accesses only the memory locations it is authorized to access. (For this discussion, type safety specifically refers to memory type safety and should not be confused with type safety in a broader respect.) For example, type-safe code cannot read values from another object's private fields. It accesses types only in well-defined, allowable ways.

During just-in-time (JIT) compilation, an optional verification process examines the metadata and Microsoft intermediate language (MSIL) of a method to be JIT-compiled into native machine code to verify that they are type safe. This process is skipped if the code has permission to bypass verification. For more information about verification, see [Managed Execution Process](https://msdn.microsoft.com/en-us/library/k5532s8a(v=vs.110).aspx).

Although verification of type safety is not mandatory to run managed code, type safety plays a crucial role in assembly isolation and security enforcement. When code is type safe, the common language runtime can completely isolate assemblies from each other. This isolation helps ensure that assemblies cannot adversely affect each other and it increases application reliability. Type-safe components can execute safely in the same process even if they are trusted at different levels. When code is not type safe, unwanted side effects can occur. For example, the runtime cannot prevent managed code from calling into native (unmanaged) code and performing malicious operations. When code is type safe, the runtime's security enforcement mechanism ensures that it does not access native code unless it has permission to do so. All code that is not type safe must have been granted [SecurityPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermission(v=vs.110).aspx) with the passed enum member [SkipVerification](https://msdn.microsoft.com/en-us/library/tht37fky(v=vs.110).aspx) to run.

For more information, see [Writing Verifiably Type-Safe Code](https://msdn.microsoft.com/en-us/library/01k04eaf(v=vs.110).aspx).

### [Principal](javascript:void(0))

A principal represents the identity and role of a user and acts on the user's behalf. Role-based security in the .NET Framework supports three kinds of principals:

* Generic principals represent users and roles that exist independent of Windows users and roles.
* Windows principals represent Windows users and their roles (or their Windows groups). A Windows principal can impersonate another user, which means that the principal can access a resource on a user's behalf while presenting the identity that belongs to that user.
* Custom principals can be defined by an application in any way that is needed for that particular application. They can extend the basic notion of the principal's identity and roles.

For more information, see [Principal and Identity Objects](https://msdn.microsoft.com/en-us/library/ftx85f8x(v=vs.110).aspx).

### [Authentication](javascript:void(0))

Authentication is the process of discovering and verifying the identity of a principal by examining the user's credentials and validating those credentials against some authority. The information obtained during authentication is directly usable by your code. You can also use .NET Framework role-based security to authenticate the current user and to determine whether to allow that principal to access your code. See the overloads of the [WindowsPrincipal.IsInRole](https://msdn.microsoft.com/en-us/library/system.security.principal.windowsprincipal.isinrole(v=vs.110).aspx) method for examples of how to authenticate the principal for specific roles. For example, you can use the [WindowsPrincipal.IsInRole(String)](https://msdn.microsoft.com/en-us/library/fs485fwh(v=vs.110).aspx) overload to determine if the current user is a member of the Administrators group.

A variety of authentication mechanisms are used today, many of which can be used with .NET Framework role-based security. Some of the most commonly used mechanisms are basic, digest, Passport, operating system (such as NTLM or Kerberos), or application-defined mechanisms.

#### [**Example**](javascript:void(0))

The following example requires that the active principal be an administrator. The *name* parameter is **null**, which allows any user who is an administrator to pass the demand.

|  |
| --- |
| **Note Note** |
| In Windows Vista, User Account Control (UAC) determines the privileges of a user. If you are a member of the Built-in Administrators group, you are assigned two run-time access tokens: a standard user access token and an administrator access token. By default, you are in the standard user role. To execute the code that requires you to be an administrator, you must first elevate your privileges from standard user to administrator. You can do this when you start an application by right-clicking the application icon and indicating that you want to run as an administrator. |

C#

[**C++**](https://msdn.microsoft.com/en-us/library/z164t8hs(v=vs.110).aspx?cs-save-lang=1&cs-lang=cpp#code-snippet-1)

[**VB**](https://msdn.microsoft.com/en-us/library/z164t8hs(v=vs.110).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

using System;

using System.Threading;

using System.Security.Permissions;

using System.Security.Principal;

class SecurityPrincipalDemo

{

public static void Main()

{

AppDomain.CurrentDomain.SetPrincipalPolicy(PrincipalPolicy.WindowsPrincipal);

PrincipalPermission principalPerm = new PrincipalPermission(null, "Administrators");

principalPerm.Demand();

Console.WriteLine("Demand succeeded.");

}

}

The following example demonstrates how to determine the identity of the principal and the roles available to the principal. An application of this example might be to confirm that the current user is in a role you allow for using your application.

C#

[**C++**](https://msdn.microsoft.com/en-us/library/z164t8hs(v=vs.110).aspx?cs-save-lang=1&cs-lang=cpp#code-snippet-2)

[**VB**](https://msdn.microsoft.com/en-us/library/z164t8hs(v=vs.110).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

using System;

using System.Threading;

using System.Security.Permissions;

using System.Security.Principal;

class SecurityPrincipalDemo

{

public static void DemonstrateWindowsBuiltInRoleEnum()

{

AppDomain myDomain = Thread.GetDomain();

myDomain.SetPrincipalPolicy(PrincipalPolicy.WindowsPrincipal);

WindowsPrincipal myPrincipal = (WindowsPrincipal)Thread.CurrentPrincipal;

Console.WriteLine("{0} belongs to: ", myPrincipal.Identity.Name.ToString());

Array wbirFields = Enum.GetValues(typeof(WindowsBuiltInRole));

foreach (object roleName in wbirFields)

{

try

{

// Cast the role name to a RID represented by the WindowsBuildInRole value.

Console.WriteLine("{0}? {1}.", roleName,

myPrincipal.IsInRole((WindowsBuiltInRole)roleName));

Console.WriteLine("The RID for this role is: " + ((int)roleName).ToString());

}

catch (Exception)

{

Console.WriteLine("{0}: Could not obtain role for this RID.",

roleName);

}

}

// Get the role using the string value of the role.

Console.WriteLine("{0}? {1}.", "Administrators",

myPrincipal.IsInRole("BUILTIN\\" + "Administrators"));

Console.WriteLine("{0}? {1}.", "Users",

myPrincipal.IsInRole("BUILTIN\\" + "Users"));

// Get the role using the WindowsBuiltInRole enumeration value.

Console.WriteLine("{0}? {1}.", WindowsBuiltInRole.Administrator,

myPrincipal.IsInRole(WindowsBuiltInRole.Administrator));

// Get the role using the WellKnownSidType.

SecurityIdentifier sid = new SecurityIdentifier(WellKnownSidType.BuiltinAdministratorsSid, null);

Console.WriteLine("WellKnownSidType BuiltinAdministratorsSid {0}? {1}.", sid.Value, myPrincipal.IsInRole(sid));

}

public static void Main()

{

DemonstrateWindowsBuiltInRoleEnum();

}

}

### [Authorization](javascript:void(0))

Authorization is the process of determining whether a principal is allowed to perform a requested action. Authorization occurs after authentication and uses information about the principal's identity and roles to determine what resources the principal can access. You can use .NET Framework role-based security to implement authorization.

### [Security concerns for internal virtual keywords](javascript:void(0))

You should never base the security of your application on a member that is marked with the [internal](https://msdn.microsoft.com/en-us/library/7c5ka91b(v=vs.110).aspx) [virtual](https://msdn.microsoft.com/en-us/library/9fkccyh4(v=vs.110).aspx) modifier in C# (the[Overloads](https://msdn.microsoft.com/en-us/library/ybxdk47f(v=vs.110).aspx) [Overridable](https://msdn.microsoft.com/en-us/library/zcfd2sa9(v=vs.110).aspx) [Friend](https://msdn.microsoft.com/en-us/library/08w05ey2(v=vs.110).aspx) modifier in Visual Basic). Although members marked with these modifiers can only be overridden by other members within the current assembly, this rule is enforced only by the C# and Visual Basic languages. The runtime does not enforce this rule. It is therefore possible to override members marked as **internal virtual**in C# and **Overloads Overridable Friend** in Visual Basic using Microsoft Intermediate Language, or any other language that does not enforce this rule.

## Code Access Security

Code Access Security

**.NET Framework 1.1, 2.0, 3.0, 3.**

Today's highly connected computer systems are frequently exposed to code originating from various, possibly unknown sources. Code can be attached to e-mail, contained in documents, or downloaded over the Internet. Unfortunately, many computer users have experienced firsthand the effects of malicious mobile code, including viruses and worms, which can damage or destroy data and cost time and money.

Most common security mechanisms give rights to users based on their logon credentials (usually a password) and restrict resources (often directories and files) that the user is allowed to access. However, this approach fails to address several issues: users obtain code from many sources, some of which might be unreliable; code can contain bugs or vulnerabilities that enable it to be exploited by malicious code; and code sometimes does things that the user does not know it will do. As a result, computer systems can be damaged and private data can be leaked when cautious and trustworthy users run malicious or error-filled software. Most operating system security mechanisms require that every piece of code must be completely trusted in order to run, except perhaps for scripts on a Web page. Therefore, there is still a need for a widely applicable security mechanism that allows code originating from one computer system to execute with protection on another system, even when there is no trust relationship between the systems.

To help protect computer systems from malicious mobile code, to allow code from unknown origins to run with protection, and to help prevent trusted code from intentionally or accidentally compromising security, the .NET Framework provides a security mechanism called code access security. Code access security allows code to be trusted to varying degrees depending on where the code originates and on other aspects of the code's identity. Code access security also enforces the varying levels of trust on code, which minimizes the amount of code that must be fully trusted in order to run. Using code access security can reduce the likelihood that your code can be misused by malicious or error-filled code. It can reduce your liability because you can specify the set of operations your code should be allowed to perform as well as the operations your code should never be allowed to perform. Code access security can also help minimize the damage that can result from security vulnerabilities in your code.

All managed code that targets the common language runtime receives the benefits of code access security, even if that code does not make a single code access security call. However, all applications should make code access requests, as outlined in [Code Access Security Basics](https://msdn.microsoft.com/en-us/library/33tceax8(v=vs.71).aspx).

[Introduction to Code Access Security](https://msdn.microsoft.com/en-us/library/c5tk9z76(v=vs.71).aspx)

Provides an overview of code access security.

[Code Access Security Basics](https://msdn.microsoft.com/en-us/library/33tceax8(v=vs.71).aspx)

Describes code access security and its most common uses.

[Using Libraries from Partially Trusted Code](https://msdn.microsoft.com/en-us/library/8skskf63(v=vs.71).aspx)

Describes how to enable libraries for use with unmanaged code and how to use libraries from unmanaged code.

[Writing Secure Class Libraries](https://msdn.microsoft.com/en-us/library/e942ksxt(v=vs.71).aspx)

Describes security considerations for class libraries.

[Writing Secure Managed Controls](https://msdn.microsoft.com/en-us/library/fked7c79(v=vs.71).aspx)

Describes security considerations for developing and deploying managed controls.

[Creating Your Own Code Access Permissions](https://msdn.microsoft.com/en-us/library/yctbsyf4(v=vs.71).aspx)

Describes how to how create custom permissions.

**Related Sections**

[Securing Applications](https://msdn.microsoft.com/en-us/library/fkytk30f(v=vs.71).aspx)

Provides an overview of the entire .NET Framework security system.

[Key Security Concepts](https://msdn.microsoft.com/en-us/library/z164t8hs(v=vs.71).aspx)

Provides an overview of many of the key terms and concepts used in the .NET Framework security system.

[Role-Based Security](https://msdn.microsoft.com/en-us/library/52kd59t0(v=vs.71).aspx)

Describes how to incorporate security based on roles.

[Cryptographic Services](https://msdn.microsoft.com/en-us/library/93bskf9z(v=vs.71).aspx)

Describes how to incorporate cryptography into your applications.

[Security Policy Management](https://msdn.microsoft.com/en-us/library/c1k0eed6(v=vs.71).aspx)

Describes how to manage security policy.

[Security Tools](https://msdn.microsoft.com/en-us/library/7w3fd0wb(v=vs.71).aspx)

Describes the tools used to implement and administer the .NET Framework security system.

[Security in the .NET Framework](https://msdn.microsoft.com/en-us/library/fkytk30f(v=vs.80).aspx)

Provides an overview of the entire .NET Framework security system.

## Code Access Security

**.NET Framework 4, 4.5, 4.6**

Today's highly connected computer systems are frequently exposed to code originating from various, possibly unknown sources. Code can be attached to e-mail, contained in documents, or downloaded over the Internet. Unfortunately, many computer users have experienced firsthand the effects of malicious mobile code, including viruses and worms, which can damage or destroy data and cost time and money.

Most common security mechanisms give rights to users based on their logon credentials (usually a password) and restrict resources (often directories and files) that the user is allowed to access. However, this approach fails to address several issues: users obtain code from many sources, some of which might be unreliable; code can contain bugs or vulnerabilities that enable it to be exploited by malicious code; and code sometimes does things that the user does not know it will do. As a result, computer systems can be damaged and private data can be leaked when cautious and trustworthy users run malicious or error-filled software. Most operating system security mechanisms require that every piece of code must be completely trusted in order to run, except perhaps for scripts on a Web page. Therefore, there is still a need for a widely applicable security mechanism that allows code originating from one computer system to execute with protection on another system, even when there is no trust relationship between the systems.

The .NET Framework provides a security mechanism called code access security to help protect computer systems from malicious mobile code, to allow code from unknown origins to run with protection, and to help prevent trusted code from intentionally or accidentally compromising security. Code access security enables code to be trusted to varying degrees depending on where the code originates and on other aspects of the code's identity. Code access security also enforces the varying levels of trust on code, which minimizes the amount of code that must be fully trusted in order to run. Using code access security can reduce the likelihood that your code will be misused by malicious or error-filled code. It can reduce your liability, because you can specify the set of operations your code should be allowed to perform. Code access security can also help minimize the damage that can result from security vulnerabilities in your code.

|  |
| --- |
| **NoteNote** |
| Major changes have been made to code access security in the .NET Framework version 4. The most notable change has been [security transparency](https://msdn.microsoft.com/en-us/library/ee191569(v=vs.100).aspx), but there are also other significant changes that affect code access security. For information about these changes, see[Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx). |

Code access security primarily affects library code and partially trusted applications. Library developers must protect their code from unauthorized access from partially trusted applications. Partially trusted applications are applications that are loaded from external sources such as the Internet. Applications that are installed on your desktop or on the local intranet run in full trust. Full-trust applications are not affected by code access security unless they are marked as [security-transparent](https://msdn.microsoft.com/en-us/library/ee191569(v=vs.100).aspx), because they are fully trusted. The only limitation for full-trust applications is that applications that are marked with the [SecurityTransparentAttribute](https://msdn.microsoft.com/en-us/library/system.security.securitytransparentattribute(v=vs.100).aspx) attribute cannot call code that is marked with the[SecurityCriticalAttribute](https://msdn.microsoft.com/en-us/library/system.security.securitycriticalattribute(v=vs.100).aspx) attribute. Partially trusted applications must be run in a sandbox (for example, in Internet Explorer) so that code access security can be applied. If you download an application from the Internet and try to run it from your desktop, you will get a[NotSupportedException](https://msdn.microsoft.com/en-us/library/system.notsupportedexception(v=vs.100).aspx) with the message: "An attempt was made to load an assembly from a network location which would have caused the assembly to be sandboxed in previous versions of the .NET Framework. This release of the .NET Framework does not enable CAS policy by default, so this load may be dangerous." If you are sure that the application can be trusted, you can enable it to be run as full trust by using the [<loadFromRemoteSources> element](https://msdn.microsoft.com/en-us/library/dd409252(v=vs.100).aspx). For information about running an application in a sandbox, see [How to: Run Partially Trusted Code in a Sandbox](https://msdn.microsoft.com/en-us/library/bb763046(v=vs.100).aspx).

All managed code that targets the common language runtime receives the benefits of code access security, even if that code does not make a single code access security call. For more information, see [Code Access Security Basics](https://msdn.microsoft.com/en-us/library/33tceax8(v=vs.100).aspx).

[Key Functions of Code Access Security](javascript:void(0))

Code access security helps limit the access that code has to protected resources and operations. In the .NET Framework, code access security performs the following functions:

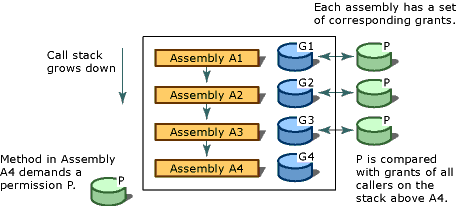
* Defines permissions and permission sets that represent the right to access various system resources.
* Enables code to demand that its callers have specific permissions.
* Enables code to demand that its callers possess a digital signature, thus allowing only callers from a particular organization or site to call the protected code.
* Enforces restrictions on code at run time by comparing the granted permissions of every caller on the call stack to the permissions that callers must have.

[Walking the Call Stack](javascript:void(0))

To determine whether code is authorized to access a resource or perform an operation, the runtime's security system walks the call stack, comparing the granted permissions of each caller to the permission being demanded. If any caller in the call stack does not have the demanded permission, a security exception is thrown and access is refused. The stack walk is designed to help prevent luring attacks, in which less-trusted code calls highly trusted code and uses it to perform unauthorized actions. Demanding permissions of all callers at run time affects performance, but it is essential to help protect code from luring attacks by less-trusted code. To optimize performance, you can have your code perform fewer stack walks; however, you must be sure that you do not expose a security weakness whenever you do this.

The following illustration shows the stack walk that results when a method in Assembly A4 demands that its callers have permission P.

Security stack walk



[**Related Topics**](javascript:void(0))

|  |  |
| --- | --- |
| **Title** | **Description** |
| [Code Access Security Basics](https://msdn.microsoft.com/en-us/library/33tceax8(v=vs.100).aspx) | Describes code access security and its most common uses. |
| [Security-Transparent Code, Level 2](https://msdn.microsoft.com/en-us/library/dd233102(v=vs.100).aspx) | Describes the security transparency model in the .NET Framework 4. |
| [Using Libraries from Partially Trusted Code](https://msdn.microsoft.com/en-us/library/8skskf63(v=vs.100).aspx) | Describes how to enable libraries for use with unmanaged code and how to use libraries from unmanaged code. |
| [Writing Secure Class Libraries](https://msdn.microsoft.com/en-us/library/e942ksxt(v=vs.100).aspx) | Describes security considerations for class libraries. |
| [Creating Your Own Code Access Permissions](https://msdn.microsoft.com/en-us/library/yctbsyf4(v=vs.100).aspx) | Describes how to create custom permissions. |
| [Key Security Concepts](https://msdn.microsoft.com/en-us/library/z164t8hs(v=vs.100).aspx) | Provides an overview of many of the key terms and concepts used in the .NET Framework security system. |
| [Role-Based Security](https://msdn.microsoft.com/en-us/library/shz8h065(v=vs.100).aspx) | Describes how to incorporate security based on roles. |
| [Cryptographic Services](https://msdn.microsoft.com/en-us/library/92f9ye3s(v=vs.100).aspx) | Describes how to incorporate cryptography into your applications. |
| [Security Tools (.NET Framework)](https://msdn.microsoft.com/en-us/library/dd233106(v=vs.100).aspx) | Describes the tools used to implement and administer the .NET Framework security system. |

### **Introduction to Code Access** Security

Introduction to Code Access Security

**.NET Framework 1.1, 2.0, 3.0, 3.5**

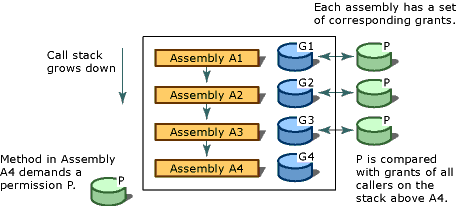
Code access security is a mechanism that helps limit the access that code has to protected resources and operations. In the .NET Framework, code access security performs the following functions:

* Defines permissions and permission sets that represent the right to access various system resources.
* Enables administrators to configure security policy by associating sets of permissions with groups of code (code groups).
* Enables code to request the permissions it requires in order to run, as well as the permissions that would be useful to have, and specifies which permissions the code must never have.
* Grants permissions to each assembly that is loaded, based on the permissions requested by the code and on the operations permitted by security policy.
* Enables code to demand that its callers have specific permissions.
* Enables code to demand that its callers possess a digital signature, thus allowing only callers from a particular organization or site to call the protected code.
* Enforces restrictions on code at run time by comparing the granted permissions of every caller on the call stack to the permissions that callers must have.

To determine whether code is authorized to access a resource or perform an operation, the runtime's security system walks the call stack, comparing the granted permissions of each caller to the permission being demanded. If any caller in the call stack does not have the demanded permission, a security exception is thrown and access is refused. The stack walk is designed to help prevent luring attacks, in which less-trusted code calls highly trusted code and uses it to perform unauthorized actions. Demanding permissions of all callers at run time affects performance, but it is essential to help protect code from luring attacks by less-trusted code. To optimize performance, you can have your code perform fewer stack walks; however, you must be sure that you do not expose a security weakness whenever you do this.

The following figure illustrates the stack walk that results when a method in Assembly A4 demands that its callers have permission P.

Security stack walk



In one typical use of code access security, an application downloads a control from a local intranet host Web site directly to the client so that the user can enter data. The control is built using an installed class library. The following are some of the ways code access security might be used in this scenario:

* Before load time, an administrator can configure security policy to specify that code be given special authority (more permission than local internet code would usually receive) if it has a particular digital signature. By default, the predefined **LocalIntranet** named permission set is associated with all code that is downloaded from the local intranet.
* At load time, the runtime grants the control no more permissions than those associated with the **LocalIntranet** named permission set, unless the control has a trusted signature. In that case, it is granted the permissions associated with the **LocalIntranet** permission set and potentially some additional permissions because of its trusted signature.
* At run time, whenever a caller (in this case the hosted control) accesses a library that exposes protected resources or a library that calls unmanaged code, the library makes a security demand, which causes the permissions of its callers to be checked for the appropriate permission grants. These security checks help prevent the control from performing unauthorized actions on the client's computers.

### Code Access Security Basics

**.NET Framework 1.1, 2.0, 3.0, 3.5**

Every application that targets the common language runtime must interact with the runtime's security system. When an application executes, it is automatically evaluated and given a set of permissions by the runtime. Depending on the permissions that the application receives, it either runs properly or generates a security exception. The local security settings on a particular computer ultimately decide which permissions code receives. Because these settings can change from computer to computer, you can never be sure that your code will receive sufficient permissions to run. This is in contrast to the world of unmanaged development, in which you do not have to worry about your code's permission to run.

Every developer must be familiar with the following code access security concepts in order to write effective applications targeting the common language runtime:

* Writing type-safe code: To enable code to benefit from code access security, you must use a compiler that generates verifiably type-safe code. For more information, see [Writing Verifiably Type-Safe Code](https://msdn.microsoft.com/en-us/library/01k04eaf(v=vs.71).aspx).
* Imperative and declarative syntax: Interaction with the runtime security system is performed using imperative and declarative security calls. Declarative calls are performed using attributes; imperative calls are performed using new instances of classes within your code. Some calls can be performed only imperatively, while others can be performed only declaratively. Some calls can be performed in either manner. For more information, see [Security Syntax](https://msdn.microsoft.com/en-us/library/a95batfc(v=vs.71).aspx).
* Requesting permissions for your code: Requests are applied to the assembly scope, where your code informs the runtime about permissions that it either needs to run or specifically does not want. Security requests are evaluated by the runtime when your code is loaded into memory. Requests cannot influence the runtime to give your code more permissions than the runtime would have given your code had the request not been made. However, requests are what your code uses to inform the runtime about the permissions it requires in order to run. For more information, see [Requesting Permissions](https://msdn.microsoft.com/en-us/library/yd267cce(v=vs.71).aspx).
* Using secure class libraries: Your class libraries use code access security to specify the permissions they require in order to be accessed. You should be aware of the permissions required to access any library that your code uses and make appropriate requests in your code. For more information, see [Using Secure Class Libraries](https://msdn.microsoft.com/en-us/library/32y7hb4c(v=vs.71).aspx).

See Also

[Writing Verifiably Type-Safe Code](https://msdn.microsoft.com/en-us/library/01k04eaf(v=vs.71).aspx) | [Security Syntax](https://msdn.microsoft.com/en-us/library/a95batfc(v=vs.71).aspx) | [Requesting Permissions](https://msdn.microsoft.com/en-us/library/yd267cce(v=vs.71).aspx) | [Using Secure Class Libraries](https://msdn.microsoft.com/en-us/library/32y7hb4c(v=vs.71).aspx) | [Code Access Security](https://msdn.microsoft.com/en-us/library/930b76w0(v=vs.71).aspx)

### Code Access Security Basics

**.NET Framework 4**

Every application that targets the common language runtime (that is, every managed application) must interact with the runtime's security system. When a managed application is loaded, its host automatically grants it a set of permissions. These permissions are determined by the host's local security settings or by the sandbox the application is in. Depending on these permissions, the application either runs properly or generates a security exception.

The default host for desktop applications allows code to run in full trust. Therefore, if your application targets the desktop, it has an unrestricted permission set. Other hosts or sandboxes provide a limited permission set for applications. Because the permission set can change from host to host, you must design your application to use only the permissions that your target host allows

You must be familiar with the following code access security concepts in order to write effective applications that target the common language runtime:

* Writing type-safe code: To enable code to benefit from code access security, you must use a compiler that generates verifiably type-safe code. For more information, see [Writing Verifiably Type-Safe Code](https://msdn.microsoft.com/en-us/library/01k04eaf(v=vs.100).aspx).
* Imperative and declarative syntax: Interaction with the runtime security system is performed using imperative and declarative security calls. Declarative calls are performed using attributes; imperative calls are performed using new instances of classes within your code. Some calls can be performed only imperatively, others can be performed only declaratively, and some calls can be performed in either manner. For more information, see [Security Syntax](https://msdn.microsoft.com/en-us/library/a95batfc(v=vs.100).aspx).
* Using secure class libraries: Class libraries use code access security to specify the permissions they require in order to be accessed. You should be aware of the permissions required to access any library that your code uses.
* Transparent code: In the .NET Framework version 4 and later, in addition to identifying specific permissions, you must also determine whether your code should run as security-transparent. Security-transparent code cannot call types or members that are identified as security-critical. This rule applies to full-trust applications as well as partially trusted applications. For more information, see [Using Secure Class Libraries](https://msdn.microsoft.com/en-us/library/32y7hb4c(v=vs.100).aspx) and [Security-Transparent Code](https://msdn.microsoft.com/en-us/library/ee191569(v=vs.100).aspx).

### Code Access Security Basics

**.NET Framework 4.6 and 4.5**

Every application that targets the common language runtime (that is, every managed application) must interact with the runtime's security system. When a managed application is loaded, its host automatically grants it a set of permissions. These permissions are determined by the host's local security settings or by the sandbox the application is in. Depending on these permissions, the application either runs properly or generates a security exception.

The default host for desktop applications allows code to run in full trust. Therefore, if your application targets the desktop, it has an unrestricted permission set. Other hosts or sandboxes provide a limited permission set for applications. Because the permission set can change from host to host, you must design your application to use only the permissions that your target host allows.

You must be familiar with the following code access security concepts in order to write effective applications that target the common language runtime:

* **Type-safe code**: Type-safe code is code that accesses types only in well-defined, allowable ways. For example, given a valid object reference, type-safe code can access memory at fixed offsets that correspond to actual field members. If the code accesses memory at arbitrary offsets outside the range of memory that belongs to that object's publicly exposed fields, it is not type-safe. To enable code to benefit from code access security, you must use a compiler that generates verifiably type-safe code. For more information, see the[Writing Verifiably Type-Safe Code](https://msdn.microsoft.com/en-us/library/33tceax8(v=vs.110).aspx#typesafe_code) section later in this topic.
* **Imperative and declarative syntax**: Code that targets the common language runtime can interact with the security system by requesting permissions, demanding that callers have specified permissions, and overriding certain security settings (given enough privileges). You use two forms of syntax to programmatically interact with the .NET Framework security system: declarative syntax and imperative syntax. Declarative calls are performed using attributes; imperative calls are performed using new instances of classes within your code. Some calls can be performed only imperatively, others can be performed only declaratively, and some calls can be performed in either manner.
* **Secure class libraries**: A secure class library uses security demands to ensure that the library's callers have permission to access the resources that the library exposes. For example, a secure class library might have a method for creating files that would demand that its callers have permissions to create files. The .NET Framework consists of secure class libraries. You should be aware of the permissions required to access any library that your code uses. For more information, see the [Using Secure Class Libraries](https://msdn.microsoft.com/en-us/library/33tceax8(v=vs.110).aspx#secure_library) section later in this topic.
* **Transparent code**: Starting with the .NET Framework 4, in addition to identifying specific permissions, you must also determine whether your code should run as security-transparent. Security-transparent code cannot call types or members that are identified as security-critical. This rule applies to full-trust applications as well as partially trusted applications. For more information, see [Security-Transparent Code](https://msdn.microsoft.com/en-us/library/ee191569(v=vs.110).aspx).

[**Writing Verifiably Type-Safe Code**](javascript:void(0))

Just-in-time (JIT) compilation performs a verification process that examines code and tries to determine whether the code is type-safe. Code that is proven during verification to be type-safe is called *verifiably type-safe code*. Code can be type-safe, yet might not be verifiably type-safe because of the limitations of the verification process or of the compiler. Not all languages are type-safe, and some language compilers, such as Microsoft Visual C++, cannot generate verifiably type-safe managed code. To determine whether the language compiler you use generates verifiably type-safe code, consult the compiler's documentation. If you use a language compiler that generates verifiably type-safe code only when you avoid certain language constructs, you might want to use the [PEVerify tool](https://msdn.microsoft.com/en-us/library/62bwd2yd(v=vs.110).aspx) to determine whether your code is verifiably type-safe.

Code that is not verifiably type-safe can attempt to execute if security policy allows the code to bypass verification. However, because type safety is an essential part of the runtime's mechanism for isolating assemblies, security cannot be reliably enforced if code violates the rules of type safety. By default, code that is not type-safe is allowed to run only if it originates from the local computer. Therefore, mobile code should be type-safe.

[**Using Secure Class Libraries**](javascript:void(0))

If your code requests and is granted the permissions required by the class library, it will be allowed to access the library and the resources that the library exposes will be protected from unauthorized access. If your code does not have the appropriate permissions, it will not be allowed to access the class library, and malicious code will not be able to use your code to indirectly access protected resources. Other code that calls your code must also have permission to access the library. If it does not, your code will be restricted from running as well.

Code access security does not eliminate the possibility of human error in writing code. However, if your application uses secure class libraries to access protected resources, the security risk for application code is decreased, because class libraries are closely scrutinized for potential security problems.

[**Declarative Security**](javascript:void(0))

Declarative security syntax uses [attributes](https://msdn.microsoft.com/en-us/library/5x6cd29c(v=vs.110).aspx) to place security information into the [metadata](https://msdn.microsoft.com/en-us/library/xcd8txaw(v=vs.110).aspx) of your code. Attributes can be placed at the assembly, class, or member level, to indicate the type of request, demand, or override you want to use. Requests are used in applications that target the common language runtime to inform the runtime security system about the permissions that your application needs or does not want. Demands and overrides are used in libraries to help protect resources from callers or to override default security behavior.

|  |
| --- |
| **Note Note** |
| In the .NET Framework 4, there have been important changes to the .NET Framework security model and terminology. For more information about these changes, see [Security Changes in the .NET Framework](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.110).aspx). |

In order to use declarative security calls, you must initialize the state data of the permission object so that it represents the particular form of permission you need. Every built-in permission has an attribute that is passed a [SecurityAction](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.110).aspx) enumeration to describe the type of security operation you want to perform. However, permissions also accept their own parameters that are exclusive to them.

The following code fragment shows declarative syntax for requesting that your code's callers have a custom permission calledMyPermission. This permission is a hypothetical custom permission and does not exist in the .NET Framework. In this example, the declarative call is placed directly before the class definition, specifying that this permission be applied to the class level. The attribute is passed a **SecurityAction.Demand** structure to specify that callers must have this permission in order to run.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/33tceax8(v=vs.110).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

[MyPermission(SecurityAction.Demand, Unrestricted = true)]

public class MyClass

{

public MyClass()

{

//The constructor is protected by the security call.

}

public void MyMethod()

{

//This method is protected by the security call.

}

public void YourMethod()

{

//This method is protected by the security call.

}

}

[**Imperative Security**](javascript:void(0))

Imperative security syntax issues a security call by creating a new instance of the permission object you want to invoke. You can use imperative syntax to perform demands and overrides, but not requests.

Before you make the security call, you must initialize the state data of the permission object so that it represents the particular form of the permission you need. For example, when creating a [FileIOPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.fileiopermission(v=vs.110).aspx) object, you can use the constructor to initialize the **FileIOPermission**object so that it represents either unrestricted access to all files or no access to files. Or, you can use a different **FileIOPermission**object, passing parameters that indicate the type of access you want the object to represent (that is, read, append, or write) and what files you want the object to protect.

In addition to using imperative security syntax to invoke a single security object, you can use it to initialize a group of permissions in a permission set. For example, this technique is the only way to reliably perform [assert](https://msdn.microsoft.com/en-us/library/91wteedy(v=vs.110).aspx) calls on multiple permissions in one method. Use the[PermissionSet](https://msdn.microsoft.com/en-us/library/system.security.permissionset(v=vs.110).aspx) and [NamedPermissionSet](https://msdn.microsoft.com/en-us/library/system.security.namedpermissionset(v=vs.110).aspx) classes to create a group of permissions and then call the appropriate method to invoke the desired security call.

You can use imperative syntax to perform demands and overrides, but not requests. You might use imperative syntax for demands and overrides instead of declarative syntax when information that you need in order to initialize the permission state becomes known only at run time. For example, if you want to ensure that callers have permission to read a certain file, but you do not know the name of that file until run time, use an imperative demand. You might also choose to use imperative checks instead of declarative checks when you need to determine at run time whether a condition holds and, based on the result of the test, make a security demand (or not).

The following code fragment shows imperative syntax for requesting that your code's callers have a custom permission calledMyPermission. This permission is a hypothetical custom permission and does not exist in the .NET Framework. A new instance ofMyPermision is created in MyMethod, guarding only this method with the security call.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/33tceax8(v=vs.110).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

public class MyClass {

public MyClass(){

}

public void MyMethod() {

//MyPermission is demanded using imperative syntax.

MyPermission Perm = new MyPermission();

Perm.Demand();

//This method is protected by the security call.

}

public void YourMethod() {

//This method is not protected by the security call.

}

}

[**Using Managed Wrapper Classes**](javascript:void(0))

Most applications and components (except secure libraries) should not directly call unmanaged code. There are several reasons for this. If code calls unmanaged code directly, it will not be allowed to run in many circumstances because code must be granted a high level of trust to call native code. If policy is modified to allow such an application to run, it can significantly weaken the security of the system, leaving the application free to perform almost any operation.

Additionally, code that has permission to access unmanaged code can probably perform almost any operation by calling into an unmanaged API. For example, code that has permission to call unmanaged code does not need [FileIOPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.fileiopermission(v=vs.110).aspx) to access a file; it can just call an unmanaged (Win32) file API directly, bypassing the managed file API that requires **FileIOPermission**. If managed code has permission to call into unmanaged code and does call directly into unmanaged code, the security system will be unable to reliably enforce security restrictions, since the runtime cannot enforce such restrictions on unmanaged code.

If you want your application to perform an operation that requires accessing unmanaged code, it should do so through a trusted managed class that wraps the required functionality (if such a class exists). Do not create a wrapper class yourself if one already exists in a secure class library. The wrapper class, which must be granted a high degree of trust to be allowed to make the call into unmanaged code, is responsible for demanding that its callers have the appropriate permissions. If you use the wrapper class, your code only needs to request and be granted the permissions that the wrapper class demands.

#### Writing Verifiably Type-Safe Code

**.NET Framework 1.1, 2.0, 3.0, 3.5**

Type-safecode is code that accesses types only in well-defined, allowable ways. For example, given a valid object reference, type-safe code can access memory at fixed offsets corresponding to actual field members. However, if the code accesses memory at arbitrary offsets outside the range of memory that belongs to that object's publicly exposed fields, it is not type-safe.

[JIT compilation](https://msdn.microsoft.com/en-us/library/ht8ecch6(v=vs.71).aspx) performs a process called verification that examines code and attempts to determine whether the code is type-safe. Code that is proven during verification to be type-safe is called verifiably type-safe code. Code can be type-safe, yet not be verifiably type-safe, due to the limitations of the verification process or of the compiler. Not all languages are type-safe, and some language compilers, such as Microsoft Visual C++, cannot generate verifiably type-safe managed code. To determine whether the language compiler you use generates verifiably type-safe code, consult the compiler's documentation. If you use a language compiler that generates verifiably type-safe code only when you avoid certain language constructs, you might want to use the .NET Framework SDK [PEVerify tool](https://msdn.microsoft.com/en-us/library/62bwd2yd(v=vs.71).aspx) to determine whether your code is verifiably type-safe.

Code that is not verifiably type-safe can attempt to execute if security policy allows the code to bypass verification. However, because type safety is an essential part of the runtime's mechanism for isolating assemblies, security cannot be reliably enforced if code violates the rules of type safety. By default, code that is not type-safe is allowed to run only if it originates from the local computer. Therefore, mobile code should be type-safe.

See Also

[Code Access Security](https://msdn.microsoft.com/en-us/library/930b76w0(v=vs.71).aspx) | [Code Access Security Basics](https://msdn.microsoft.com/en-us/library/33tceax8(v=vs.71).aspx) | [Compiling MSIL to Native Code](https://msdn.microsoft.com/en-us/library/ht8ecch6(v=vs.71).aspx) | [PEVerify Tool](https://msdn.microsoft.com/en-us/library/62bwd2yd(v=vs.71).aspx)

#### Writing Verifiably Type-Safe Code

**.NET Framework 4**

Type-safe code is code that accesses types only in well-defined, allowable ways. For example, given a valid object reference, type-safe code can access memory at fixed offsets corresponding to actual field members. However, if the code accesses memory at arbitrary offsets outside the range of memory that belongs to that object's publicly exposed fields, it is not type-safe.

Just-in-time (JIT) compilation performs a process called verification that examines code and attempts to determine whether the code is type-safe. Code that is proven during verification to be type-safe is called verifiably type-safe code. Code can be type-safe, yet not be verifiably type-safe, due to the limitations of the verification process or of the compiler. Not all languages are type-safe, and some language compilers, such as Microsoft Visual C++, cannot generate verifiably type-safe managed code. To determine whether the language compiler you use generates verifiably type-safe code, consult the compiler's documentation. If you use a language compiler that generates verifiably type-safe code only when you avoid certain language constructs, you might want to use the Windows Software Development Kit (SDK) [PEVerify tool](https://msdn.microsoft.com/en-us/library/62bwd2yd(v=vs.100).aspx) to determine whether your code is verifiably type-safe.

Code that is not verifiably type-safe can attempt to execute if security policy allows the code to bypass verification. However, because type safety is an essential part of the runtime's mechanism for isolating assemblies, security cannot be reliably enforced if code violates the rules of type safety. By default, code that is not type-safe is allowed to run only if it originates from the local computer. Therefore, mobile code should be type-safe.

[**See Also**](javascript:void(0))

Reference

[Peverify.exe (PEVerify Tool)](https://msdn.microsoft.com/en-us/library/62bwd2yd(v=vs.100).aspx)

Concepts

[Code Access Security Basics](https://msdn.microsoft.com/en-us/library/33tceax8(v=vs.100).aspx)

[Managed Execution Process](https://msdn.microsoft.com/en-us/library/k5532s8a(v=vs.100).aspx)

Other Resources

[Code Access Security](https://msdn.microsoft.com/en-us/library/c5tk9z76(v=vs.100).aspx)

#### Security Syntax

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4.0**

Code that targets the common language runtime can interact with the security system by requesting permissions, demanding that callers have specified permissions, and overriding certain security settings (given enough privileges). You use two different forms of syntax to programmatically interact with the .NET Framework security system: declarative syntax and imperative syntax. Some operations can be done using both forms of syntax while other operations can be performed using only declarative syntax. You should be familiar with both forms.

##### Declarative Security

**.NET Framework 1.1, 2.0, 3.0,**

Declarative security syntax uses [attributes](https://msdn.microsoft.com/en-us/library/5x6cd29c(v=vs.71).aspx) to place security information into the [metadata](https://msdn.microsoft.com/en-us/library/4y7k7c6k(v=vs.71).aspx) of your code. Attributes can be placed at the assembly, class, or member level, to indicate the type of request, demand, or override you want to use. Requests are used in applications that target the common language runtime to inform the runtime security system about the permissions that your application needs or does not want. Demands and overrides are used in libraries to help protect resources from callers or to override default security behavior.

In order to use declarative security calls, you must initialize the state data of the permission object so that it represents the particular form of permission you need. Every built-in permission has an attribute that is passed a [SecurityAction](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.71).aspx) enumeration to describe the type of security operation you want to perform. However, permissions also accept their own parameters that are exclusive to them. For a complete description of the parameters specific to permissions, see the section describing built-in [permissions](https://msdn.microsoft.com/en-us/library/5ba4k1c5(v=vs.71).aspx).

The following code fragment shows declarative syntax for requesting that your code's callers have a custom permission called MyPermission. This permission is a hypothetical custom permission and does not exist in the .NET Framework. In this example, the declarative call is placed directly before the class definition, specifying that this permission be applied to the class level. The attribute is passed a**SecurityAction.Demand** structure to specify that callers must have this permission in order to run.

[C#]

[MyPermission(SecurityAction.Demand, Unrestricted = true)]

public class MyClass

{

public MyClass()

{

//The constructor is protected by the security call.

}

public void MyMethod()

{

//This method is protected by the security call.

}

public void YourMethod()

{

//This method is protected by the security call.

}

}

###### Extending Metadata Using Attributes

**.NET Framework 1.1, 2.0, 3.0, 3.5**

The common language runtime allows you to add keyword-like descriptive declarations, called attributes, to annotate programming elements such as types, fields, methods, and properties. Attributes are saved with the metadata of a Microsoft .NET Framework file and can be used to describe your code to the runtime or to affect application behavior at run time. While the .NET Framework supplies many useful attributes, you can also design and deploy your own.

###### Extending Metadata Using Attributes

**.NET Framework 4**

If you have used C++, you are probably familiar with declarations that contain keywords, such as **public** and **private**, that provide additional information about class members. These keywords further define the behavior of class members by describing their accessibility to other classes. Because compilers are explicitly designed to recognize predefined keywords, you do not traditionally have the opportunity to create your own. The common language runtime, however, allows you to add keyword-like descriptive declarations, called attributes, to annotate programming elements such as types, fields, methods, and properties.

When you compile your code for the runtime, it is converted into Microsoft intermediate language (MSIL) and placed inside a portable executable (PE) file along with metadata generated by the compiler. Attributes allow you to place extra descriptive information into metadata that can be extracted using runtime reflection services. The compiler creates attributes when you declare instances of special classes that derive from [System.Attribute](https://msdn.microsoft.com/en-us/library/system.attribute(v=vs.100).aspx).

The .NET Framework uses attributes for a variety of reasons and to address a number of issues. Attributes describe how to serialize data, specify characteristics that are used to enforce security, and limit optimizations by the just-in-time (JIT) compiler so the code remains easy to debug. Attributes can also record the name of a file or the author of code, or control the visibility of controls and members during forms development.

You can use attributes to describe your code in practically any way conceivable and to affect run-time behavior in creative new ways. Attributes allow you to add your own descriptive elements to C#, Visual C++, Microsoft Visual Basic 2005, or any other language that targets the runtime, without having to rewrite your compiler.

###### Extending Metadata Using Attributes

**.NET Framework 4.6 and 4.5**

The common language runtime allows you to add keyword-like descriptive declarations, called attributes, to annotate programming elements such as types, fields, methods, and properties. When you compile your code for the runtime, it is converted into Microsoft intermediate language (MSIL) and placed inside a portable executable (PE) file along with metadata generated by the compiler. Attributes allow you to place extra descriptive information into metadata that can be extracted using runtime reflection services. The compiler creates attributes when you declare instances of special classes that derive from [System.Attribute](https://msdn.microsoft.com/en-us/library/system.attribute(v=vs.110).aspx).

The .NET Framework uses attributes for a variety of reasons and to address a number of issues. Attributes describe how to serialize data, specify characteristics that are used to enforce security, and limit optimizations by the just-in-time (JIT) compiler so the code remains easy to debug. Attributes can also record the name of a file or the author of code, or control the visibility of controls and members during forms development.

###### Metadata and Self-Describing Components

**.NET Framework 1.1, 2.0, 3.0, 3.6**

In the past, a software component (.exe or .dll) written in one language could not easily use a software component written in another language. COM provided a step forward in solving this problem. The .NET Framework makes component interoperation even easier by allowing compilers to emit additional declarative information into all modules and assemblies. This information, called metadata, helps components to seamlessly interact.

###### SecurityAction Enumeration

Has under gone some changes

https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.110).aspx

|  |  |  |
| --- | --- | --- |
| **Declaration of security action** | **Time of action** | **Targets supported** |
| **LinkDemand** (do not use in the .NET Framework 4) | Just-in-time compilation | Class, method |
| **InheritanceDemand** | Load time | Class, method |
| **Demand** | Run time | Class, method |
| **Assert** | Run time | Class, method |
| **Deny** (obsolete in the .NET Framework 4) | Run time | Class, method |
| **PermitOnly** | Run time | Class, method |
| **RequestMinimum** (obsolete in the .NET Framework 4) | Grant time | Assembly |
| **RequestOptional** (obsolete in the .NET Framework 4) | Grant time | Assembly |
| **RequestRefuse** (obsolete in the .NET Framework 4) | Grant time | Assembly |

For additional information about attribute targets, see [Attribute](https://msdn.microsoft.com/en-us/library/system.attribute(v=vs.110).aspx).

##### Imperative Security

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4**

Imperative security syntax issues a security call by creating a new instance of the permission object you want to invoke. You can use imperative syntax to perform demands and overrides, but not requests.

Before you make the security call, you must initialize the state data of the permission object so that it represents the particular form of the permission you need. For example, when creating a [FileIOPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.fileiopermission(v=vs.71).aspx) object, you can use the constructor to initialize the **FileIOPermission**object so that it represents either unrestricted access to all files or no access to files. Or, you can use a different **FileIOPermission**object, passing parameters that indicate the type of access you want the object to represent (that is, read, append, or write) and what files you want the object to protect.

In addition to using imperative security syntax to invoke a single security object, you can use it to initialize a group of permissions called a[permission set](https://msdn.microsoft.com/en-us/library/4652tyx7(v=vs.71).aspx). For example, this technique is the only way to reliably perform [assert](https://msdn.microsoft.com/en-us/library/91wteedy(v=vs.71).aspx) calls on multiple permissions in one method. Use the[PermissionSet](https://msdn.microsoft.com/en-us/library/system.security.permissionset(v=vs.71).aspx) and [NamedPermissionSet](https://msdn.microsoft.com/en-us/library/system.security.namedpermissionset(v=vs.71).aspx) classes to create a group of permissions and then call the appropriate method to invoke the desired security call.

You can use imperative syntax to perform demands and overrides, but not requests. You might use imperative syntax for demands and overrides instead of declarative syntax when information that you need in order to initialize the permission state becomes known only at run time. For example, if you want to ensure that callers have permission to read a certain file, but you do not know the name of that file until run time, use an imperative demand. You might also choose to use imperative checks instead of declarative checks when you need to determine at run time whether a condition holds and, based on the result of the test, make a security demand (or not).

The following code fragment shows imperative syntax for requesting that your code's callers have a custom permission called MyPermission. This permission is a hypothetical custom permission and does not exist in the .NET Framework. A new instance of MyPermision is created inMyMethod, guarding only this method with the security call.

[C#]

public class MyClass {

public MyClass(){

}

public void MyMethod() {

//MyPermission is demanded using imperative syntax.

MyPermission Perm = new MyPermission();

Perm.Demand();

//This method is protected by the security call.

}

public void YourMethod() {

//This method is not protected by the security call.

}

}

#### Requesting Permissions

**.NET Framework 1.1, 2.0, 3.0,**

Requesting permissions is the way you let the runtime know what your code needs to be allowed to do. You request permissions for an assembly by placing attributes (declarative syntax) in the assembly scope of your code. When the assembly is created, the language compiler stores the requested permissions in the [assembly manifest](https://msdn.microsoft.com/en-us/library/1w45z383(v=vs.71).aspx). At load time, the runtime examines the permission requests, and applies security policy rules to determine which permissions to grant to the assembly. Requests only influence the runtime to deny permissions to your code and never influence the runtime to give more permissions to your code. The local administration policy always has final control over the maximum permissions your code is granted.

Although your code does not have to request permissions in order to compile, there are important reasons your code should always request permissions:

* Requesting permissions increases the likelihood that your code will run properly if it is allowed to execute. Code that request a minimal set of permissions will not run unless it receives those permissions. If you do not identify a minimum set of permissions, your code must gracefully handle any and all situations where not being granted some permission might prevent it from executing properly.
* Requesting permissions helps ensure that your code is granted only the permissions it needs. If your code is not granted extra permissions, it cannot damage the resources protected by those extra permissions, even if it is exploited by malicious code or has bugs that can be leveraged to damage resources. You should request only those permissions that your code needs, and no more.
* Requesting permissions lets administrators know the minimum permissions that your application needs so that they can adjust security policy accordingly. Administrators use the [Permission View Tool (Permview.exe)](https://msdn.microsoft.com/en-us/library/06251f13(v=vs.71).aspx) to examine assemblies and set up security policy to issue required permissions. If you do not explicitly request the permissions that your application requires, the Permission View tool cannot return any information about the permissions that your application requires. If an administrator does not know this information, your application is difficult to administer.

Requesting permissions informs the runtime which permissions your application needs to function or specifically does not want. For example, if your application writes to the local hard disk without using isolated storage, your application must have [FileIOPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.fileiopermission(v=vs.71).aspx). If your code does not request **FileIOPermission** and the local security settings do not allow your application to have this permission, a security exception is raised when the application attempts to write to the disk. Even if the application can handle the exception, it will not be allowed to write to the disk. This behavior might be frustrating to users if your application is a text-editing program that they have been using for an extended period of time. On the other hand, if your application requests **FileIOPermission** and the local security settings do not allow your application to have **FileIOPermission**, the application will generate the exception when it starts and the user will not face the problem of losing any work. Additionally, if your application requests **FileIOPermission** and if it is a trusted application, the administrator can adjust security policy to allow it to execute from the remote share.

If your code does not access protected resources or perform protected operations, you do not need to request any permissions. For example, a permission request might not be necessary if the code simply computes a result based on inputs passed to it, without using any resources. If your code does access protected resources but does not request the necessary permissions, it might still be allowed to execute, but it could fail at some point during execution if it attempts to access a resource for which it does not have the necessary permission.

To request permissions, you must know which resources and protected operations your code uses, and you must also know which permissions protect those resources and operations. In addition, you need to keep track of any resources accessed by any class library methods that are called by your components. For a list of the code access permissions that are included with the .NET Framework, see the[Permissions](https://msdn.microsoft.com/en-us/library/5ba4k1c5(v=vs.71).aspx) topic.

The following table describes the types of permission requests.

|  |  |
| --- | --- |
| **Permission request** | **Description** |
| Minimum permissions ([RequestMinimum](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.71).aspx)) | Permissions your code must have in order to run. |
| Optional permissions ([RequestOptional](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.71).aspx)) | Permissions your code can use but can run effectively without. This request implicitly refuses all other permissions not specifically requested. |
| Refused permissions ([RequestRefuse](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.71).aspx)) | Permissions that you want to ensure will never be granted to your code, even if security policy allows them to be granted. |
| Perform any of the above requests on built-in permission sets ([Requesting Built-in Permission Sets](https://msdn.microsoft.com/en-us/library/0d005ted(v=vs.71).aspx)). | Built-in permission sets, including **Nothing**, **Execution**, **FullTrust**,**Internet**,**LocalIntranet**, and **SkipVerification**. |
| Perform any of the above requests on XML-encoded permission sets ([Requesting XML-Encoded Permissions](https://msdn.microsoft.com/en-us/library/cywyd064(v=vs.71).aspx)). | XML representation (either a string containing the XML-encoded permission set or the location of an XML file containing the encoded permission set) of a desired permission set. |

If you specify required permissions (using **RequestMinimum**), the code will be granted each required permission that [security policy](https://msdn.microsoft.com/en-us/library/tha13y5z(v=vs.71).aspx) allows. The code will be allowed to run only if it is granted all the permissions it requires.

Requesting optional permissions without also requesting required permissions can, in some cases, severely restrict the permissions granted to an assembly. For example, suppose security policy normally grants Assembly A the permissions associated with the **Everything** named permission set. If the developer of Assembly A requests Permission A as optional and does not request any required permissions, Assembly A will be granted either Permission A (if security policy allows it) or no permissions at all.

See Also

[Code Access Security](https://msdn.microsoft.com/en-us/library/930b76w0(v=vs.71).aspx) | [Code Access Security Basics](https://msdn.microsoft.com/en-us/library/33tceax8(v=vs.71).aspx) | [Assembly Manifest](https://msdn.microsoft.com/en-us/library/1w45z383(v=vs.71).aspx) | [Permissions](https://msdn.microsoft.com/en-us/library/5ba4k1c5(v=vs.71).aspx) | [FileIOPermission Class](https://msdn.microsoft.com/en-us/library/system.security.permissions.fileiopermission(v=vs.71).aspx) |[SecurityAction.RequestMinimum](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.71).aspx) | [SecurityAction.RequestOptional](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.71).aspx) | [Requesting Built-in Permission Sets](https://msdn.microsoft.com/en-us/library/0d005ted(v=vs.71).aspx) | [Requesting XML-Encoded Permissions](https://msdn.microsoft.com/en-us/library/cywyd064(v=vs.71).aspx) | [RequestRefuse](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.71).aspx) | [Security Policy](https://msdn.microsoft.com/en-us/library/tha13y5z(v=vs.71).aspx)

See Also 3.0

Tasks

[How to: Request Permission for a Named Permission Set](https://msdn.microsoft.com/en-us/library/0d005ted(v=vs.85).aspx)

Reference

[FileIOPermission Class](https://msdn.microsoft.com/en-us/library/system.security.permissions.fileiopermission(v=vs.85).aspx)  
[SecurityAction.RequestMinimum](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.85).aspx)  
[SecurityAction.RequestOptional](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.85).aspx)  
[RequestRefuse](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.85).aspx)

Concepts

[Code Access Security Basics](https://msdn.microsoft.com/en-us/library/33tceax8(v=vs.85).aspx)  
[Assembly Manifest](https://msdn.microsoft.com/en-us/library/1w45z383(v=vs.85).aspx)  
[Security Permissions](https://msdn.microsoft.com/en-us/library/5ba4k1c5(v=vs.85).aspx)  
[Requesting XML-Encoded Permissions](https://msdn.microsoft.com/en-us/library/cywyd064(v=vs.85).aspx)  
[Security Policy](https://msdn.microsoft.com/en-us/library/tha13y5z(v=vs.85).aspx)

Other Resources

[Code Access Security](https://msdn.microsoft.com/en-us/library/930b76w0(v=vs.85).aspx)

#### Requesting Permissions

**.NET Framework 4**

|  |
| --- |
| **Important noteImportant** |
| In the .NET Framework version 4, runtime support has been removed for the [RequestMinimum](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx), [RequestOptional](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx), and [RequestRefuse](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx)permission requests. This topic does not apply to code that is based on the .NET Framework 4 or later. For more information about this and other changes, see [Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx). |

Requesting permissions is the way you let the runtime know what your code needs to be allowed to do. You request permissions for an assembly by placing attributes (declarative syntax) in the assembly scope of your code. When the assembly is created, the language compiler stores the requested permissions in the [assembly manifest](https://msdn.microsoft.com/en-us/library/1w45z383(v=vs.100).aspx). At load time, the runtime examines the permission requests, and applies security policy rules to determine which permissions to grant to the assembly. Requests only influence the runtime to deny permissions to your code and never influence the runtime to give more permissions to your code than has been granted. The local administration policy always has final control over the maximum permissions your code is granted.

|  |
| --- |
| **NoteNote** |
| Code that is intended to reside on a computer runs in the **My\_Computer\_Zone**, which, by default, has full trust. Full trust causes all permission requests to be approved. Demands for permissions always succeed, even for identity permissions where the identity condition is not met. If your code is intended to run only in the **My\_Computer\_Zone**, there is no value in requesting permissions. If your code is intended to run in any of the other zones, requesting permissions is recommended. |
| **NoteNote** |
| In the .NET Framework 3.5 Service Pack 1 and later, applications in shares on the intranet run in full trust by default. If your application is intended to run from a share, it will run in full trust just like an application that is residing on a computer. For more information, see[Running Intranet Applications in Full Trust](https://msdn.microsoft.com/en-us/library/cc713717(v=vs.100).aspx). |

Although your code does not have to request permissions in order to run in partial trust, there are important reasons your code should always request permissions:

* Requesting permissions increases the likelihood that your code will run properly if it is allowed to execute. Code that request a minimal set of permissions will not run unless it receives those permissions. If you do not identify a minimum set of permissions, your code must gracefully handle any and all situations where not being granted some permission might prevent it from executing properly.
* Requesting permissions helps ensure that your code is granted only the permissions it needs. If your code is not granted extra permissions, it cannot damage the resources protected by those extra permissions, even if it is exploited by malicious code or has bugs that can be leveraged to damage resources. You should request only those permissions that your code needs, and no more.
* Requesting permissions lets administrators know the minimum permissions that your application needs so that they can adjust security policy accordingly. You can determine the permissions your application requires from the Security tab on the project properties page for a Visual Studio project. If an administrator does not know this information, your application is difficult to administer.

Requesting permissions informs the runtime which permissions your application needs to function or specifically does not want. For example, if your application writes to the local hard disk without using isolated storage, your application must have [FileIOPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.fileiopermission(v=vs.100).aspx). If your code does not request **FileIOPermission** and the local security settings do not allow your application to have this permission, a security exception is raised when the application attempts to write to the disk. Even if the application can handle the exception, it will not be allowed to write to the disk. This behavior might be frustrating to users if your application is a text-editing program that they have been using for an extended period of time. On the other hand, if your application requests **FileIOPermission** and the local security settings do not allow your application to have **FileIOPermission**, the application will generate the exception when it starts and the user will not face the problem of losing any work. Additionally, if your application requests **FileIOPermission** and if it is a trusted application, the administrator can adjust security policy to allow it to execute from the remote share.

If your code does not access protected resources or perform protected operations, you do not need to request any permissions. For example, a permission request might not be necessary if the code simply computes a result based on inputs passed to it, without using any resources. If your code does access protected resources but does not request the necessary permissions, it might still be allowed to execute, but it could fail at some point during execution if it attempts to access a resource for which it does not have the necessary permission.

To request permissions, you must know which resources and protected operations your code uses, and you must also know which permissions protect those resources and operations. In addition, you need to keep track of any resources accessed by any class library methods that are called by your components. For a list of the code access permissions that are included with the .NET Framework, see the[Permissions](https://msdn.microsoft.com/en-us/library/5ba4k1c5(v=vs.100).aspx) topic.

The following table describes the types of permission requests.

|  |  |
| --- | --- |
| **Permission request** | **Description** |
| Minimum permissions ([RequestMinimum](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx)) | Permissions your code must have in order to run. |
| Optional permissions ([RequestOptional](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx)) | Permissions your code can use but can run effectively without. This request implicitly refuses all other permissions not specifically requested. |
| Refused permissions ([RequestRefuse](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx)) | Permissions that you want to ensure will never be granted to your code, even if security policy allows them to be granted. |
| Perform any of the above requests on built-in permission sets ([Requesting Built-in Permission Sets](https://msdn.microsoft.com/en-us/library/0d005ted(v=vs.100).aspx)). | Built-in permission sets, including **Nothing**, **Execution**, **FullTrust**,**Internet**,**LocalIntranet**, and **SkipVerification**. |
| Perform any of the above requests on XML-encoded permission sets ([Requesting XML-Encoded Permissions](https://msdn.microsoft.com/en-us/library/cywyd064(v=vs.100).aspx)). | XML representation (either a string containing the XML-encoded permission set or the location of an XML file containing the encoded permission set) of a desired permission set. |

If you specify required permissions (using **RequestMinimum**), the code will be granted each required permission that [security policy](https://msdn.microsoft.com/en-us/library/tha13y5z(v=vs.100).aspx) allows. The code will be allowed to run only if it is granted all the permissions it requires.

Requesting optional permissions without also requesting required permissions can, in some cases, severely restrict the permissions granted to an assembly. For example, suppose security policy normally grants Assembly A the permissions associated with the **Everything** named permission set. If the developer of Assembly A requests Permission A as optional and does not request any required permissions, Assembly A will be granted either Permission A (if security policy allows it) or no permissions at all.

##### Requesting Minimum Permissions

**.NET Framework 1.1**

**Newer title**

##### How to: Request Permission to Access Unmanaged Code

**.NET Framework 2.0, 3.0, 3.5**

You can easily request permissions by applying [attributes](https://msdn.microsoft.com/en-us/library/5x6cd29c(v=vs.71).aspx) that represent the permissions you want to request to the assembly level of your code. The attributes you use can vary, depending on the permissions you are requesting. Requests are compiled into the [metadata](https://msdn.microsoft.com/en-us/library/4y7k7c6k(v=vs.71).aspx) of your application's [assembly manifest](https://msdn.microsoft.com/en-us/library/1w45z383(v=vs.71).aspx) and evaluated by the runtime when your code is loaded into memory during execution.

The following example shows how to request permission to access unmanaged code. Note that it uses a [SecurityPermissionAttribute](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionattribute(v=vs.71).aspx) and it specifies two values: a [SecurityAction](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.71).aspx) value that specifies the kind of permission request being made (**RequestMinimum**, in this case), and a flag that indicates which permission is being requested. In this case, [SecurityPermissionFlag.UnmanagedCode](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag(v=vs.71).aspx) specifies a request for unmanaged code permission. The assembly: syntax tells the compiler that the attribute is being applied at the assembly level.

[C#]

//The request is placed at the assembly level.

using System.Security.Permissions;

[assembly:SecurityPermissionAttribute(SecurityAction.RequestMinimum, Flags = SecurityPermissionFlag.UnmanagedCode)]

namespace MyNamespace {

using System;

using System.Runtime.InteropServices;

public class MyClass {

public MyClass() {

}

public void MyMethod() {

//Perform interoperation with unmanaged code here.

}

}

}

If the previous code does not receive **SecurityPermission** with the **UnmanagedCode** flag, the runtime throws a [PolicyException](https://msdn.microsoft.com/en-us/library/system.security.policy.policyexception(v=vs.71).aspx) and the code is not allowed to execute. However, if the code does receive that permission, then it is allowed to execute.

See Also

[Extending Metadata Using Attributes](https://msdn.microsoft.com/en-us/library/5x6cd29c(v=vs.71).aspx) | [Requesting Permissions](https://msdn.microsoft.com/en-us/library/yd267cce(v=vs.71).aspx) | [Code Access Security](https://msdn.microsoft.com/en-us/library/930b76w0(v=vs.71).aspx) | [Metadata and Self-Describing Components](https://msdn.microsoft.com/en-us/library/4y7k7c6k(v=vs.71).aspx) |[SecurityPermissionAttribute Class](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionattribute(v=vs.71).aspx) | [SecurityAction Enumeration](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.71).aspx) | [SecurityPermissionFlag.UnmanagedCode](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag(v=vs.71).aspx)

See Also

Reference

[SecurityPermissionAttribute Class](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionattribute(v=vs.85).aspx)  
[SecurityAction Enumeration](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.85).aspx)  
[SecurityPermissionFlag.UnmanagedCode](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag(v=vs.85).aspx)

Concepts

[Requesting Permissions](https://msdn.microsoft.com/en-us/library/yd267cce(v=vs.85).aspx)

Other Resources

[Extending Metadata Using Attributes](https://msdn.microsoft.com/en-us/library/5x6cd29c(v=vs.85).aspx)  
[Code Access Security](https://msdn.microsoft.com/en-us/library/930b76w0(v=vs.85).aspx)  
[Metadata and Self-Describing Components](https://msdn.microsoft.com/en-us/library/4y7k7c6k(v=vs.85).aspx)

##### How to: Request Minimum Permissions by Using the RequestMinimum Flag

**.NET Framework 2.0, 3.0, 3.5**

The

[RequestMinimum](https://msdn.microsoft.com/en-US/library/system.security.permissions.securityaction(v=vs.80).aspx) flag allows you to request a minimum set of permissions your code requires to execute. By contrast, the [RequestRefuse](https://msdn.microsoft.com/en-US/library/system.security.permissions.securityaction(v=vs.80).aspx) flag allows you to refuse permissions by explicitly specifying which ones your code should not be granted.

In contrast to using the **RequestMinimum** flag, your application will execute if it does not receive all the permissions that you request using the [RequestOptional](https://msdn.microsoft.com/en-US/library/system.security.permissions.securityaction(v=vs.80).aspx) flag, and a [SecurityException](https://msdn.microsoft.com/en-US/library/system.security.securityexception(v=vs.80).aspx) will be thrown when your application attempts to access a protected resource. If you use this type of request, you must enable your code to catch any exceptions that will be thrown if your code is not granted the optional permission.

The following example requests [FileIOPermission](https://msdn.microsoft.com/en-US/library/system.security.permissions.fileiopermission(v=vs.80).aspx) using the **RequestMinimum** flag. The example will not execute if it has not been granted the requested permission. This example assumes that a hypothetical class Log exists in LogNameSpace. The Log class contains the MakeLogmethod that creates a new log file on the local computer. This application creates a new instance of the Log class and executes the MakeLogmethod in the **try** block. Using the **catch** keyword, it intercepts any **SecurityException** thrown and displays a message.

Example

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229913(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

//The request is placed at the assembly level.

using System.Security.Permissions;

[assembly:FileIOPermission(SecurityAction.RequestMinimum, Unrestricted = true)]

namespace MyNamespace {

using System;

using System.Security;

//The hypothetical class log is in this namespace.

using LogNameSpace;

public class MyClass {

public MyClass() {

}

public static void Main(string[] args) {

//Put any code that requires optional permissions in the try block.

try {

Log MyLog = new Log();

MyLog.MakeLog();

Console.WriteLine("The Log has been created.");

}

//Catch the security exception and inform the user that the

//application was not granted FileIOPermission.

catch(SecurityException) {

Console.WriteLine("This application does not have permission to write to the disk.");

}

}

}

}

The previous code creates the log file and displays the following message to the console if it has sufficient permissions:

The Log has been created.

If the code is run from a share and the local security settings do not allow such code to have **FileIOPermission**, the code is not granted sufficient permission and displays the following message:

This application does not have permission to write to the disk.

##### Requesting Optional Permissions

**.NET Framework 1.1**

**The newer**

##### How to: Request Optional Permissions by Using the RequestOptional Flag

**.NET Framework 2.0, 3.0, 3.5**

The [SecurityAction.RequestOptional](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.71).aspx) flag allows you to request a set of permissions while refusing all other permissions the runtime otherwise might have been willing to give. By contrast, the **RequestRefuse** flag allows you to refuse permissions by explicitly specifying which ones your code should not be granted.

In contrast to using the **RequestMinimum** flag, your application will execute if it does not receive all the permissions that you request using the **RequestOptiona**l flag, and a **SecurityException** will be thrown when your application attempts to access a protected resource. If you use this type of request, you must enable your code to catch any exceptions that will be thrown if your code is not granted the optional permission.

The following example requests **FileIOPermission** using the **SecurityAction.RequestOptional** flag, indirectly refusing all other permissions. This example assumes that a hypothetical class Log exists in LogNameSpace. The Log class contains the MakeLog method that creates a new log file on the local computer. This application creates a new instance of the Log class and executes the MakeLog method in the **try** block. Using the **catch** keyword, it intercepts any **SecurityException** thrown and displays a message.

[C#]

//The request is placed at the assembly level.

using System.Security.Permissions;

[assembly:FileIOPermission(SecurityAction.RequestOptional, Unrestricted = true)]

namespace MyNamespace {

using System;

using System.Security;

//The hypothetical class log is in this namespace.

using LogNameSpace;

public class MyClass {

public MyClass() {

}

public static void Main(string[] args) {

//Put any code that requires optional permissions in the try block.

try {

Log MyLog = new Log();

MyLog.MakeLog();

Console.WriteLine("The Log has been created.");

}

//Catch the security exception and inform the user that the

//application was not granted FileIOPermission.

catch(SecurityException) {

Console.WriteLine("This application does not have permission to write to the disk.");

}

}

}

}

The previous code creates the log file and displays the following message to the console if it has sufficient permissions:

The Log has been created.

If the code is run from a share and the local security settings do not allow such code to have **FileIOPermission**, the code is not granted sufficient permission and displays the following message:

This application does not have permission to write to the disk.

See Also

[Extending Metadata Using Attributes](https://msdn.microsoft.com/en-us/library/5x6cd29c(v=vs.71).aspx) | [Requesting Permissions](https://msdn.microsoft.com/en-us/library/yd267cce(v=vs.71).aspx) | [Code Access Security](https://msdn.microsoft.com/en-us/library/930b76w0(v=vs.71).aspx) | [SecurityAction Enumeration](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.71).aspx) | [FileIOPermission Class](https://msdn.microsoft.com/en-us/library/system.security.permissions.fileiopermission(v=vs.71).aspx) |[UIPermission Class](https://msdn.microsoft.com/en-us/library/system.security.permissions.uipermission(v=vs.71).aspx)

##### How to: Request Optional Permissions by Using the RequestOptional Flag

**.NET Framework 3.5**

Updated: January 2010

The [RequestOptional](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.90).aspx) flag enables you to request a set of permissions and implicitly refuse all other permissions that the runtime otherwise might be willing to give. By contrast, the [RequestRefuse](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.90).aspx) flag enables you to refuse permissions by explicitly specifying which ones your code should not be granted.

You must use the [RequestMinimum](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.90).aspx) flag to identify permissions that you must have in addition to the optional permissions you identify with the [RequestOptional](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.90).aspx) flag. Your code loads and runs even if it does not have the permissions identified with the [RequestOptional](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.90).aspx) flag. A[SecurityException](https://msdn.microsoft.com/en-us/library/system.security.securityexception(v=vs.90).aspx) is thrown when your application tries to use a resource that it does not have permission to access. If you use[RequestOptional](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.90).aspx) permissions, you must enable your code to catch any exceptions that will be thrown if your code is not granted the optional permission.

The Test example in the following procedure requests [UIPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.uipermission(v=vs.90).aspx) by using the [RequestMinimum](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.90).aspx) flag, and requests [FileIOPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.fileiopermission(v=vs.90).aspx) by using the [RequestOptional](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.90).aspx) flag. Therefore, it indirectly refuses all other permissions. The code in the **try** block in the Main method tries to create a new file. If the attempt fails, the **catch** block intercepts the [SecurityException](https://msdn.microsoft.com/en-us/library/system.security.securityexception(v=vs.90).aspx) that is thrown and displays a message.

To run an application in a sandbox that demonstrates RequestOptional

1. In Visual Studio, create a console application project.
2. Copy the code from the [Example](https://msdn.microsoft.com/en-us/library/ea5yat38(v=vs.90).aspx#example) section into the application file. The code creates a sandbox that runs with the **LocalIntranet**permission set. The **LocalIntranet** permission set does not include [FileIOPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.fileiopermission(v=vs.90).aspx).
3. On the **Project** menu, click **Properties**, click the **Signing** tab, and sign the project with a strong name key.
4. Add a new console application project named Test to the solution.
5. Copy the following code into the application file for Test. The code requests [FileIOPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.fileiopermission(v=vs.90).aspx) as an optional requirement. The application can be started even if it does not have [FileIOPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.fileiopermission(v=vs.90).aspx).

C#

[**VB**](https://msdn.microsoft.com/en-us/library/ea5yat38(v=vs.90).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

using System;

using System.Security;

using System.Security.Permissions;

using System.IO;

[assembly:FileIOPermission(SecurityAction.RequestOptional, Unrestricted = true)]

[assembly: UIPermission(SecurityAction.RequestMinimum, Unrestricted = true)]

public class MyClass

{

public MyClass()

{

}

public static void Main(string[] args)

{

//Put any code that requires optional permissions in the try block.

try

{

File.Create("test.txt");

Console.WriteLine("The file has been created.");

}

//Catch the security exception and inform the user that the

//application was not granted FileIOPermission.

catch (SecurityException)

{

Console.WriteLine("This application does not have permission to write to the disk.");

}

}

}

1. Run the sandbox application. It will load and run the test application. When the application is run, an exception is thrown with the following message:
2. This application does not have permission to write to the disk.

For more information about how to run applications in a sandbox, see [How to: Run Partially Trusted Code in a Sandbox](https://msdn.microsoft.com/en-us/library/bb763046(v=vs.90).aspx).

[**Example**](javascript:void(0))

The following example creates a sandbox that runs with **LocalIntranet** permissions.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/ea5yat38(v=vs.90).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-3)

using System;

using System.Collections;

using System.Diagnostics;

using System.Security;

using System.Security.Permissions;

using System.Security.Policy;

using System.Reflection;

using System.IO;

[assembly: FileIOPermission(SecurityAction.RequestMinimum, Unrestricted=true)]

namespace SimpleSandboxing

{

class Program

{

static void Main(string[] args)

{

// Create the permission set to grant to other assemblies.

// In this case we are granting the permissions found in the LocalIntranet zone.

PermissionSet pset = GetNamedPermissionSet("LocalIntranet");

if (pset == null)

return;

AppDomainSetup ads = new AppDomainSetup();

// Identify the folder to use for the sandbox.

Directory.CreateDirectory("C:\\Sandbox");

ads.ApplicationBase = "C:\\Sandbox";

// Copy the application to be executed to the sandbox.

File.Copy(@"..\..\..\Test\Bin\Debug\Test.exe", "C:\\sandbox\\Test.exe", true);

File.Copy(@"..\..\..\Test\Bin\Debug\Test.pdb", "C:\\sandbox\\Test.pdb", true);

Evidence hostEvidence = new Evidence();

// Create the sandboxed domain.

AppDomain sandbox = AppDomain.CreateDomain(

"Sandboxed Domain",

hostEvidence,

ads,

pset,

GetStrongName(Assembly.GetExecutingAssembly()));

sandbox.ExecuteAssemblyByName("Test");

}

public static StrongName GetStrongName(Assembly assembly)

{

if (assembly == null)

throw new ArgumentNullException("assembly");

AssemblyName assemblyName = assembly.GetName();

Debug.Assert(assemblyName != null, "Could not get assembly name");

// Get the public key blob.

byte[] publicKey = assemblyName.GetPublicKey();

if (publicKey == null || publicKey.Length == 0)

throw new InvalidOperationException("Assembly is not strongly named");

StrongNamePublicKeyBlob keyBlob = new StrongNamePublicKeyBlob(publicKey);

// Return the strong name.

return new StrongName(keyBlob, assemblyName.Name, assemblyName.Version);

}

private static PermissionSet GetNamedPermissionSet(string name)

{

IEnumerator policyEnumerator = SecurityManager.PolicyHierarchy();

// Move through the policy levels to the machine policy level.

while (policyEnumerator.MoveNext())

{

PolicyLevel currentLevel = (PolicyLevel)policyEnumerator.Current;

if (currentLevel.Label == "Machine")

{

NamedPermissionSet copy = currentLevel.GetNamedPermissionSet(name);

return (PermissionSet)copy;

}

}

return null;

}

}

}

[**See Also**](javascript:void(0))

Concepts

[Requesting Permissions](https://msdn.microsoft.com/en-us/library/yd267cce(v=vs.90).aspx)

Reference

[SecurityAction](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.90).aspx)

[FileIOPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.fileiopermission(v=vs.90).aspx)

[UIPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.uipermission(v=vs.90).aspx)

Other Resources

[Extending Metadata Using Attributes](https://msdn.microsoft.com/en-us/library/5x6cd29c(v=vs.90).aspx)

[Code Access Security](https://msdn.microsoft.com/en-us/library/930b76w0(v=vs.90).aspx)

[**Change History**](javascript:void(0))

|  |  |  |
| --- | --- | --- |
| **Date** | **History** | **Reason** |
| January 2010 | Replaced the sample to be run in the sandbox. | Customer feedback. |
| September 2008 | Expanded the information. | Customer feedback. |

##### Refusing Permissions

**.NET Framework 1.1**

**Newer title**

##### How to: Refuse Permissions by Using the RequestRefuse Flag

**.NET Framework 2.0, 3.0, 3.5**

If you are concerned that your code might be used to access system resources maliciously, you can request that it never be granted a particular permission. For example, an application that browses data in a file but never modifies the data might refuse any file-write permissions. In the event of a bug or a malicious attack, this code cannot damage the data on which it operates.

**RequestRefuse** allows a large set of permissions to be requested as optional permissions, while ensuring that certain specific permissions are not in the grant.

The following example uses **RequestRefuse** to refuse **FileIOPermission**from the common language runtime security system:

[C#]

//The request is placed at the assembly level.

using System.Security.Permissions;

[assembly:FileIOPermission(SecurityAction.RequestRefuse ,Unrestricted = true)]

namespace MyNameSpace

{

using System;

using System.Security;

using System.Security.Permissions;

using System.IO;

public class MyClass

{

public MyClass()

{

}

public static int Main(string[] args)

{

//Creation of the log is attempted in the try block.

try

{

StreamWriter TextStream = new StreamWriter("Log.txt");

TextStream.WriteLine("This Log was created on {0}", DateTime.Now);

TextStream.Close();

Console.WriteLine("The Log was created");

}

//Catch the Security exception and inform the user that the

//application was not granted FileIOPermission.

catch(SecurityException)

{

Console.WriteLine("This application does not have permission to write to the disk.");

}

return 0;

}

}

}

The previous example is not granted permission to create the file and generates a security exception. The catch statement intercepts the exception and the application displays the following message to the console:

This application does not have permission to write to the disk.

See Also

[Extending Metadata Using Attributes](https://msdn.microsoft.com/en-us/library/5x6cd29c(v=vs.71).aspx) | [Requesting Permissions](https://msdn.microsoft.com/en-us/library/yd267cce(v=vs.71).aspx) | [Code Access Security](https://msdn.microsoft.com/en-us/library/930b76w0(v=vs.71).aspx)

##### Requesting Built-in Permission Sets

**.NET Framework 1.1**

**New title**

##### How to: Request Permission for a Named Permission Set

**.NET Framework 2.0, 3.0, 3.5, 4.0**

|  |
| --- |
| **Important noteImportant** |
| In the .NET Framework version 4, runtime support has been removed for enforcing the [Deny](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx), [RequestMinimum](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx), [RequestOptional](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx), and[RequestRefuse](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx) permission requests. Do not use these requests in code that is based on .NET Framework 4 or later. For more information about this and other changes, see [Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx). |

Instead of requesting individual permissions (using **RequestMinimum**, **RequestOptional**, or **RequestRefuse**), you can request any of the following built-in permission sets: **Nothing**, **Execution**, **FullTrust**,**Internet**, **LocalIntranet**, and **SkipVerification**. You cannot request custom named permission sets or the **Everything** modifiable built-in permission set because the permissions they represent can vary. The following example shows the syntax to request permission for a named permission set. It attaches a [PermissionSetAttribute](https://msdn.microsoft.com/en-us/library/system.security.permissions.permissionsetattribute(v=vs.71).aspx) with a **Name** value representing the name of the desired permission set.

[C#]

//The attribute is placed at the assembly level.

using System.Security.Permissions;

[assembly:PermissionSetAttribute(SecurityAction.RequestMinimum, Name = "FullTrust")]

namespace MyNamespace

{

using System;

using System.Runtime.InteropServices;

public class MyClass

{

public MyClass()

{

}

public void MyMethod()

{

//Perform operations that require permissions here.

}

}

}

See Also

[Extending Metadata Using Attributes](https://msdn.microsoft.com/en-us/library/5x6cd29c(v=vs.71).aspx) | [Requesting Permissions](https://msdn.microsoft.com/en-us/library/yd267cce(v=vs.71).aspx) | [Code Access Security](https://msdn.microsoft.com/en-us/library/930b76w0(v=vs.71).aspx)

##### Requesting XML-Encoded Permissions

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4.0**

|  |
| --- |
| **Important noteImportant** |
| In the .NET Framework version 4, runtime support has been removed for the [RequestMinimum](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx), [RequestOptional](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx), and [RequestRefuse](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx)permission requests. This topic does not apply to code that is based on the .NET Framework 4 or later. For more information about this and other changes, see [Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx). |

If you need to request a custom permission set (using **RequestMinimum**, **RequestOptional**, or **RequestRefuse**), you can use an XML representation of the desired permission set in one of two ways: either you can pass a string that contains the actual XML-encoded permission set or you can provide the location of an XML file containing the encoded permission set. The following example uses XML with the [PermissionSetAttribute](https://msdn.microsoft.com/en-us/library/system.security.permissions.permissionsetattribute(v=vs.71).aspx). The XML flag is a string containing an XML-encoded permission set, which in this case represents an unrestricted**UIPermission**and an unrestricted **RegistryPermission**.

[C#]

//The attribute is placed at the assembly level.

using System.Security.Permissions;

[assembly:PermissionSetAttribute(SecurityAction.RequestMinimum, XML="<PermissionSet class=\"System.Security.PermissionSet\" version=\"1\"><Permission class=\"System.Security.Permissions.UIPermission, mscorlib\" version=\"1\"><AllWindows/></Permission><Permission class=\"System.Security.Permissions.RegistryPermission, mscorlib\" version=\"1\"><Unrestricted/></Permission></PermissionSet>")]

namespace MyNamespace

{

using System;

using System.Runtime.InteropServices;

public class MyClass

{

public MyClass()

{

}

public void MyMethod()

{

//Perform user interface operations here.

}

}

}

The following example shows a request for a custom permission set by providing the location and name of the file that contains the same XML-encoded permission set. If you do not specify the location of your XML file, the runtime looks for it in the same directory as the application.

VB

Imports System

Imports System.Runtime.InteropServices

Imports System.Security.Permissions

'The attribute is placed at the assembly level.

<assembly: PermissionSetAttribute(SecurityAction.RequestMinimum, File := "pset.xml")>

Namespace MyNamespace

Public Class MyClass1

Public Sub New()

End Sub

Public Sub MyMethod()

'Perform operations that require permissions here.

End Sub

End Class

End Namespace

[C#]

//The attribute is placed at the assembly level.

using System.Security.Permissions;

[assembly:PermissionSetAttribute(SecurityAction.RequestMinimum, File = "pset.xml")]

namespace MyNamespace

{

using System;

using System.Runtime.InteropServices;

public class MyClass

{

public MyClass()

{

}

public void MyMethod()

{

//Perform operations that require permissions here.

}

}

}

Creating an XML-Encoded Permission Set

You can create an XML encoding of a permission set by creating an instance of the [PermissionSet](https://msdn.microsoft.com/en-us/library/system.security.permissionset(v=vs.71).aspx) object, adding instances of the permissions you want to the object, and then calling the **ToXml** method to return a **SecurityElement** object that represents the XML encoding or calling the **ToString** method to return a string representation of the XML encoding.

See Also

[Requesting Permissions](https://msdn.microsoft.com/en-us/library/yd267cce(v=vs.71).aspx) | [PermissionSetAttribute Class](https://msdn.microsoft.com/en-us/library/system.security.permissions.permissionsetattribute(v=vs.71).aspx) | [PermissionSet Class](https://msdn.microsoft.com/en-us/library/system.security.permissionset(v=vs.71).aspx) | [Metadata and Self-Describing Components](https://msdn.microsoft.com/en-us/library/4y7k7c6k(v=vs.71).aspx) | [Extending Metadata Using Attributes](https://msdn.microsoft.com/en-us/library/5x6cd29c(v=vs.71).aspx) | [Code Access Security](https://msdn.microsoft.com/en-us/library/930b76w0(v=vs.71).aspx)

#### Using Secure Class Libraries

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4**

A secure library is a class library that uses security demands to ensure that the library's callers have permission to access the resources that the library exposes. For example, a secure class library might have a method for creating files that would demand that its callers have permissions to create files. The .NET Framework comprises secure class libraries.

If your code requests and is granted the permissions required by the class library, it will be allowed to access the library and the resource will be protected from unauthorized access; if your code does not have the appropriate permissions, it will not be allowed to access the class library, and malicious code will not be able to use your code to indirectly access the resource. Even if your code has permission to access a library, it will not be allowed to run if code that calls your code does not also have permission to access the library.

Code access security does not eliminate the possibility of human error in writing code; however, if applications use secure class libraries to access protected resources, the security risk for application code is decreased because class libraries are closely scrutinized for potential security problems.

See Also

[Code Access Security](https://msdn.microsoft.com/en-us/library/930b76w0(v=vs.71).aspx) | [Code Access Security Basics](https://msdn.microsoft.com/en-us/library/33tceax8(v=vs.71).aspx)

#### Using Managed Wrapper Classes

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4.0**

Most applications and components (except secure libraries) should not directly call unmanaged code. There are several reasons for this. If code calls unmanaged code directly, it will not be allowed to run in many circumstances because code must be granted a high level of trust to call native code. If policy is modified to allow such an application to run, it can significantly weaken the security of the system, leaving the application free to perform almost any operation.

Additionally, code that has permission to access unmanaged code can probably perform almost any operation by calling into an unmanaged API. For example, code that has permission to call unmanaged code does not need [FileIOPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.fileiopermission(v=vs.71).aspx) to access a file; it can just call an unmanaged (Win32) file API directly, bypassing the managed file API that requires **FileIOPermission**. If managed code has permission to call into unmanaged code and does call directly into unmanaged code, the security system will be unable to reliably enforce security restrictions, since the runtime cannot enforce such restrictions on unmanaged code.

If you want your application to perform an operation that requires accessing unmanaged code, it should do so through a trusted managed class that wraps the required functionality (if such a class exists). Do not create a wrapper class yourself if one already exists in a secure class library. The wrapper class, which must be granted a high degree of trust to be allowed to make the call into unmanaged code, is responsible for demanding that its callers have the appropriate permissions. If you use the wrapper class, your code only needs to request and be granted the permissions that the wrapper class demands.

See Also

[Code Access Security](https://msdn.microsoft.com/en-us/library/930b76w0(v=vs.71).aspx) | [Using Secure Class Libraries](https://msdn.microsoft.com/en-us/library/32y7hb4c(v=vs.71).aspx) | [FileIOPermission Class](https://msdn.microsoft.com/en-us/library/system.security.permissions.fileiopermission(v=vs.71).aspx)

### Security-Transparent Code

**.NET Framework 4.6 and 4.5**

[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635736413806036745

Security involves three interacting pieces: sandboxing, permissions, and enforcement. Sandboxing refers to the practice of creating isolated domains where some code is treated as fully trusted and other code is restricted to the permissions in the grant set for the sandbox. The application code that runs within the grant set of the sandbox is considered to be transparent; that is, it cannot perform any operations that can affect security. The grant set for the sandbox is determined by evidence ([Evidence](https://msdn.microsoft.com/en-us/library/system.security.policy.evidence(v=vs.110).aspx) class). Evidence identifies what specific permissions are required by sandboxes, and what kinds of sandboxes can be created. Enforcement refers to allowing transparent code to execute only within its grant set.

|  |
| --- |
| **Important note Important** |
| Security policy was a key element in previous versions of the .NET Framework. Starting with the .NET Framework 4, security policy is obsolete. The elimination of security policy is separate from security transparency. For information about the effects of this change, see[Code Access Security Policy Compatibility and Migration](https://msdn.microsoft.com/en-us/library/ee191568(v=vs.110).aspx). |

This topic describes the transparency model in more detail. It contains the following sections:

* [Purpose of the Transparency Model](https://msdn.microsoft.com/en-us/library/ee191569(v=vs.110).aspx#purpose)
* [Specifying the Transparency Level](https://msdn.microsoft.com/en-us/library/ee191569(v=vs.110).aspx#level)
* [Transparency Enforcement](https://msdn.microsoft.com/en-us/library/ee191569(v=vs.110).aspx#enforcement)

[**Purpose of the Transparency Model**](javascript:void(0))

Transparency is an enforcement mechanism that separates code that runs as part of the application from code that runs as part of the infrastructure. Transparency draws a barrier between code that can do privileged things (critical code), such as calling native code, and code that cannot (transparent code). Transparent code can execute commands within the bounds of the permission set it is operating in, but cannot execute, derive from, or contain critical code.

The primary goal of transparency enforcement is to provide a simple, effective mechanism for isolating different groups of code based on privilege. Within the context of the sandboxing model, these privilege groups are either fully trusted (that is, not restricted) or partially trusted (that is, restricted to the permission set granted to the sandbox).

|  |
| --- |
| **Important note Important** |
| The transparency model transcends code access security. Transparency is enforced by the just-in-time compiler and remains in effect regardless of the grant set for an assembly, including full trust. |

Transparency was introduced in the .NET Framework version 2.0 to simplify the security model, and to make it easier to write and deploy secure libraries and applications. Transparent code is also used in Microsoft Silverlight, to simplify the development of partially trusted applications.

|  |
| --- |
| **Note Note** |
| When you develop a partially trusted application, you have to be aware of the permission requirements for your target hosts. You can develop an application that uses resources that are not allowed by some hosts. This application will compile without error, but will fail when it is loaded into the hosted environment. If you have developed your application using Visual Studio, you can enable debugging in partial trust or in a restricted permission set from the development environment. For more information, see [How to: Debug a ClickOnce Application with Restricted Permissions](https://msdn.microsoft.com/en-us/library/593zkfdf(v=vs.110).aspx). The Calculate Permissions feature provided for ClickOnce applications is also available for any partially trusted application. |

[Back to top](https://msdn.microsoft.com/en-us/library/ee191569(v=vs.110).aspx#top)

[**Specifying the Transparency Level**](javascript:void(0))

The assembly-level [SecurityRulesAttribute](https://msdn.microsoft.com/en-us/library/system.security.securityrulesattribute(v=vs.110).aspx) attribute explicitly selects the [SecurityRuleSet](https://msdn.microsoft.com/en-us/library/system.security.securityruleset(v=vs.110).aspx) rules that the assembly will follow. The rules are organized under a numeric level system, where higher levels mean tighter enforcement of security rules.

The levels are as follows:

* Level 2 ([Level2](https://msdn.microsoft.com/en-us/library/system.security.securityruleset(v=vs.110).aspx)) – the .NET Framework 4 transparency rules.
* Level 1 ([Level1](https://msdn.microsoft.com/en-us/library/system.security.securityruleset(v=vs.110).aspx)) – the .NET Framework 2.0 transparency rules.

The primary difference between the two transparency levels is that level 1 does not enforce transparency rules for calls from outside the assembly and is intended only for compatibility.

|  |
| --- |
| **Important note Important** |
| You should specify level 1 transparency for compatibility only; that is, specify level 1 only for code that was developed with the .NET Framework 3.5 or earlier that uses the [AllowPartiallyTrustedCallersAttribute](https://msdn.microsoft.com/en-us/library/system.security.allowpartiallytrustedcallersattribute(v=vs.110).aspx) attribute or does not use the transparency model. For example, use level 1 transparency for .NET Framework 2.0 assemblies that allow calls from partially trusted callers (APTCA). For code that is developed for the .NET Framework 4, always use level 2 transparency. |

[**Level 2 Transparency**](javascript:void(0))

Level 2 transparency was introduced in the .NET Framework 4. The three tenets of this model are transparent code, security-safe-critical code, and security-critical code.

* Transparent code, regardless of the permissions it is granted (including full trust), can call only other transparent code or security-safe-critical code. If the code is partially trusted, it can only perform actions that are allowed by the domain’s permission set. Transparent code cannot do the following:
  + Perform an [Assert](https://msdn.microsoft.com/en-us/library/system.security.codeaccesspermission.assert(v=vs.110).aspx) operation or elevation of privilege.
  + Contain unsafe or unverifiable code.
  + Directly call critical code.
  + Call native code or code that has the [SuppressUnmanagedCodeSecurityAttribute](https://msdn.microsoft.com/en-us/library/system.security.suppressunmanagedcodesecurityattribute(v=vs.110).aspx) attribute.
  + Call a member that is protected by a [LinkDemand](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.110).aspx).
  + Inherit from critical types.

In addition, transparent methods cannot override critical virtual methods or implement critical interface methods.

* Security-safe-critical code is fully trusted but is callable by transparent code. It exposes a limited surface area of full-trust code. Correctness and security verifications happen in safe-critical code.
* Security-critical code can call any code and is fully trusted, but it cannot be called by transparent code.

[**Level 1 Transparency**](javascript:void(0))

The level 1 transparency model was introduced in the .NET Framework version 2.0 to enable developers to reduce the amount of code that is subject to a security audit. Although level 1 transparency was publicly available in version 2.0, it was primarily used only within Microsoft for security auditing purposes. Through annotations, developers are able to declare which types and members can perform security elevations and other trusted actions (security-critical) and which cannot (security-transparent). Code that is identified as transparent does not require a high degree of security auditing. Level 1 transparency states that the transparency enforcement is limited to within the assembly. In other words, any public types or members that are identified as security-critical are security-critical only within the assembly. If you want to enforce security for those types and members when they are called from outside the assembly, you must use link demands for full trust. If you do not, publicly visible security-critical types and members are treated as security-safe-critical and can be called by partially trusted code outside the assembly.

The level 1 transparency model has the following limitations:

* Security-critical types and members that are public are accessible from security-transparent code.
* The transparency annotations are enforced only within an assembly.
* Security-critical types and members must use link demands to enforce security for calls from outside the assembly.
* Inheritance rules are not enforced.
* The potential exists for transparent code to do harmful things when run in full trust.

[Back to top](https://msdn.microsoft.com/en-us/library/ee191569(v=vs.110).aspx#top)

[**Transparency Enforcement**](javascript:void(0))

Transparency rules are not enforced until transparency is calculated. At that time, an [InvalidOperationException](https://msdn.microsoft.com/en-us/library/system.invalidoperationexception(v=vs.110).aspx) is thrown if a transparency rule is violated. The time that transparency is calculated depends on multiple factors and cannot be predicted. It is calculated as late as possible. In the .NET Framework 4, assembly-level transparency calculation occurs sooner than it does in the .NET Framework 2.0. The only guarantee is that transparency calculation will occur by the time it is needed. This is similar to how the just-in-time (JIT) compiler can change the point when a method is compiled and any errors in that method are detected. Transparency calculation is invisible if your code does not have any transparency errors.

#### Security-Transparent Code, Level 1

**.NET Framework 4.6 and 4.5**

Transparency helps developers write more secure .NET Framework libraries that expose functionality to partially trusted code. Level 1 transparency was introduced in the .NET Framework version 2.0 and was primarily used only within Microsoft. Starting with the .NET Framework 4, you can use [level 2 transparency](https://msdn.microsoft.com/en-us/library/dd233102(v=vs.110).aspx). However, level 1 transparency has been retained so that you can identify legacy code that must run with the earlier security rules.

|  |
| --- |
| **Important note Important** |
| You should specify level 1 transparency for compatibility only; that is, specify level 1 only for code that was developed with the .NET Framework 3.5 or earlier that uses the [AllowPartiallyTrustedCallersAttribute](https://msdn.microsoft.com/en-us/library/system.security.allowpartiallytrustedcallersattribute(v=vs.110).aspx) or does not use the transparency model. For example, use level 1 transparency for .NET Framework 2.0 assemblies that allow calls from partially trusted callers (APTCA). For code that is developed for the .NET Framework 4, always use level 2 transparency. |

This topic contains the following sections:

* [The Level 1 Transparency Model](https://msdn.microsoft.com/en-us/library/bb397858(v=vs.110).aspx#the_level_1_transparency_model)
* [Transparency Attributes](https://msdn.microsoft.com/en-us/library/bb397858(v=vs.110).aspx#transparency_attributes)
* [Security Transparency Examples](https://msdn.microsoft.com/en-us/library/bb397858(v=vs.110).aspx#security_transparency_examples)

[**The Level 1 Transparency Model**](javascript:void(0))

When you use Level 1 transparency, you are using a security model that separates code into security-transparent, security-safe-critical, and security-critical methods.

You can mark a whole assembly, some classes in an assembly, or some methods in a class as security-transparent. Security-transparent code cannot elevate privileges. This restriction has three consequences:

* Security-transparent code cannot perform [Assert](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.110).aspx) actions.
* Any link demand that would be satisfied by security-transparent code becomes a full demand.
* Any unsafe (unverifiable) code that must execute in security-transparent code causes a full demand for the [UnmanagedCode](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag(v=vs.110).aspx)security permission.

These rules are enforced during execution by the common language runtime (CLR). Security-transparent code passes all the security requirements of the code it calls back to its callers. Demands that flow through the security-transparent code cannot elevate privileges. If a low-trust application calls security-transparent code and causes a demand for high privilege, the demand will flow back to the low-trust code and fail. The security-transparent code cannot stop the demand because it cannot perform assert actions. The same security-transparent code called from full-trust code results in a successful demand.

Security-critical is the opposite of security-transparent. Security-critical code executes with full trust and can perform all privileged operations. Security-safe-critical code is privileged code that has been through an extensive security audit to confirm that it does not allow partially trusted callers to use resources they do not have permission to access.

You have to apply transparency explicitly. The majority of your code that handles data manipulation and logic can typically be marked as security-transparent, whereas the lesser amount of code that performs elevations of privileges is marked as security-critical or security-safe-critical.

|  |
| --- |
| **Important note Important** |
| Level 1 transparency is limited to assembly scope; it is not enforced between assemblies. Level 1 transparency was primarily used within Microsoft for security audit purposes. Security-critical types and members within a level 1 assembly can be accessed by security-transparent code in other assemblies. It is important that you perform link demands for full trust in all your level 1 security-critical types and members. Security-safe-critical types and members must also confirm that callers have permissions for protected resources that are accessed by the type or member. |

For backward compatibility with earlier versions of the .NET Framework, all members that are not annotated with transparency attributes are considered to be security-safe-critical. All types that are not annotated are considered to be transparent. There are no static analysis rules to validate transparency. Therefore, you may need to debug transparency errors at run time.

[**Transparency Attributes**](javascript:void(0))

The following table describes the three attributes that you use to annotate your code for transparency.

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| [SecurityTransparentAttribute](https://msdn.microsoft.com/en-us/library/system.security.securitytransparentattribute(v=vs.110).aspx) | Allowed only at the assembly level. Identifies all types and members in the assembly as security-transparent. The assembly cannot contain any security-critical code. |
| [SecurityCriticalAttribute](https://msdn.microsoft.com/en-us/library/system.security.securitycriticalattribute(v=vs.110).aspx) | When used at the assembly level without the [Scope](https://msdn.microsoft.com/en-us/library/ms147620(v=vs.110).aspx) property, identifies all code in the assembly as security-transparent by default, but indicates that the assembly may contain security-critical code.  When used at the class level, identifies the class or method as security-critical, but not the members of the class. To make all the members security-critical, set the [Scope](https://msdn.microsoft.com/en-us/library/ms147620(v=vs.110).aspx) property to [Everything](https://msdn.microsoft.com/en-us/library/system.security.securitycriticalscope(v=vs.110).aspx).  When used at the member level, the attribute applies only to that member.  The class or member identified as security-critical can perform elevations of privilege.   |  | | --- | | **Important note Important** | | In level 1 transparency, security-critical types and members are treated as security-safe-critical when they are called from outside the assembly. You should protect security-critical types and members with a link demand for full trust to avoid unauthorized elevation of privilege. | |
| [SecuritySafeCriticalAttribute](https://msdn.microsoft.com/en-us/library/system.security.securitysafecriticalattribute(v=vs.110).aspx) | Identifies security-critical code that can be accessed by security-transparent code in the assembly. Otherwise, security-transparent code cannot access private or internal security-critical members in the same assembly. Doing so would influence security-critical code and make unexpected elevations of privilege possible. Security-safe-critical code should undergo a rigorous security audit.   |  | | --- | | **Note Note** | | Security-safe-critical types and members must validate the permissions of callers to determine whether the caller has authority to access protected resources. | |

The [SecuritySafeCriticalAttribute](https://msdn.microsoft.com/en-us/library/system.security.securitysafecriticalattribute(v=vs.110).aspx) attribute enables security-transparent code to access security-critical members in the same assembly. Consider the security-transparent and security-critical code in your assembly as separated into two assemblies. The security-transparent code would not be able to see the private or internal members of the security-critical code. Additionally, the security-critical code is generally audited for access to its public interface. You would not expect a private or internal state to be accessible outside the assembly; you would want to keep the state isolated. The [SecuritySafeCriticalAttribute](https://msdn.microsoft.com/en-us/library/system.security.securitysafecriticalattribute(v=vs.110).aspx) attribute maintains the isolation of state between security-transparent and security-critical code while providing the ability to override the isolation when it is necessary. Security-transparent code cannot access private or internal security-critical code unless those members have been marked with [SecuritySafeCriticalAttribute](https://msdn.microsoft.com/en-us/library/system.security.securitysafecriticalattribute(v=vs.110).aspx). Before applying the [SecuritySafeCriticalAttribute](https://msdn.microsoft.com/en-us/library/system.security.securitysafecriticalattribute(v=vs.110).aspx), audit that member as if it were publicly exposed.

[**Assembly-wide Annotation**](javascript:void(0))

The following table describes the effects of using security attributes at the assembly level.

|  |  |
| --- | --- |
| **Assembly attribute** | **Assembly state** |
| No attribute on a partially trusted assembly | All types and members are transparent. |
| No attribute on a fully trusted assembly (in the global assembly cache or identified as full trust in the **AppDomain**) | All types are transparent and all members are security-safe-critical. |
| **SecurityTransparent** | All types and members are transparent. |
| **SecurityCritical(SecurityCriticalScope.Everything)** | All types and members are security-critical. |
| **SecurityCritical** | All code defaults to transparent. However, individual types and members can have other attributes. |

[**Security Transparency Examples**](javascript:void(0))

To use the .NET Framework 2.0 transparency rules (level 1 transparency), use the following assembly annotation:

[assembly: SecurityRules(SecurityRuleSet.Level1)]

If you want to make a whole assembly transparent to indicate that the assembly does not contain any critical code and does not elevate privileges in any way, you can explicitly add transparency to the assembly with the following attribute:

[assembly: SecurityTransparent]

If you want to mix critical and transparent code in the same assembly, start by marking the assembly with the [SecurityCriticalAttribute](https://msdn.microsoft.com/en-us/library/system.security.securitycriticalattribute(v=vs.110).aspx)attribute to indicate that the assembly can contain critical code, as follows:

[assembly: SecurityCritical]

If you want to perform security-critical actions, you must explicitly mark the code that will perform the critical action with another[SecurityCriticalAttribute](https://msdn.microsoft.com/en-us/library/system.security.securitycriticalattribute(v=vs.110).aspx) attribute, as shown in the following code example:

[assembly: SecurityCritical]

Public class A

{

[SecurityCritical]

private void Critical()

{

// critical

}

public int SomeProperty

{

get {/\* transparent \*/ }

set {/\* transparent \*/ }

}

}

public class B

{

internal string SomeOtherProperty

{

get { /\* transparent \*/ }

set { /\* transparent \*/ }

}

}

The previous code is transparent except for the Critical method, which is explicitly marked as security-critical. Transparency is the default setting, even with the assembly-level [SecurityCriticalAttribute](https://msdn.microsoft.com/en-us/library/system.security.securitycriticalattribute(v=vs.110).aspx) attribute.

#### Security-Transparent Code, Level 2

**.NET Framework 4.6 and 4.5**

[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635736413806036745

Level 2 transparency was introduced in the .NET Framework 4. The three tenets of this model are transparent code, security-safe-critical code, and security-critical code.

* Transparent code, including code that is running as full trust, can call other transparent code or security-safe-critical code only. It can only perform actions allowed by the domain’s partial trust permission set (if one exists). Transparent code cannot do the following:
  + Perform an [Assert](https://msdn.microsoft.com/en-us/library/system.security.codeaccesspermission.assert(v=vs.110).aspx) or elevation of privilege.
  + Contain unsafe or unverifiable code.
  + Directly call critical code.
  + Call native code or code with the [SuppressUnmanagedCodeSecurityAttribute](https://msdn.microsoft.com/en-us/library/system.security.suppressunmanagedcodesecurityattribute(v=vs.110).aspx) attribute.
  + Call a member that is protected by a [LinkDemand](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.110).aspx).
  + Inherit from critical types.

In addition, transparent methods cannot override critical virtual methods or implement critical interface methods.

* Safe-critical code is fully trusted but is callable by transparent code. It exposes a limited surface area of full-trust code; correctness and security verifications happen in safe-critical code.
* Security-critical code can call any code and is fully trusted, but it cannot be called by transparent code.

This topic contains the following sections:

* [Usage Examples and Behaviors](https://msdn.microsoft.com/en-us/library/dd233102(v=vs.110).aspx#examples)
* [Override Patterns](https://msdn.microsoft.com/en-us/library/dd233102(v=vs.110).aspx#override)
* [Inheritance Rules](https://msdn.microsoft.com/en-us/library/dd233102(v=vs.110).aspx#inheritance)
* [Additional Information and Rules](https://msdn.microsoft.com/en-us/library/dd233102(v=vs.110).aspx#additional)

[**Usage Examples and Behaviors**](javascript:void(0))

To specify .NET Framework 4 rules (level 2 transparency), use the following annotation for an assembly:

[assembly: SecurityRules(SecurityRuleSet.Level2)]

To lock into the .NET Framework 2.0 rules (level 1 transparency), use the following annotation:

[assembly: SecurityRules(SecurityRuleSet.Level1)]

If you do not annotate an assembly, the .NET Framework 4 rules are used by default. However, the recommended best practice is to use the[SecurityRulesAttribute](https://msdn.microsoft.com/en-us/library/system.security.securityrulesattribute(v=vs.110).aspx) attribute instead of depending on the default.

[**Assemblywide Annotation**](javascript:void(0))

The following rules apply to the use of attributes at the assembly level:

* No attributes: If you do not specify any attributes, the runtime interprets all code as security-critical, except where being security-critical violates an inheritance rule (for example, when overriding or implementing a transparent virtual or interface method). In those cases, the methods are safe-critical. Specifying no attribute causes the common language runtime to determine the transparency rules for you.
* **SecurityTransparent**: All code is transparent; the entire assembly will not do anything privileged or unsafe.
* **SecurityCritical**: All code that is introduced by types in this assembly is critical; all other code is transparent. This scenario is similar to not specifying any attributes; however, the common language runtime does not automatically determine the transparency rules. For example, if you override a virtual or abstract method or implement an interface method, by default, that method is transparent. You must explicitly annotate the method as **SecurityCritical** or **SecuritySafeCritical**; otherwise, a[TypeLoadException](https://msdn.microsoft.com/en-us/library/system.typeloadexception(v=vs.110).aspx) will be thrown at load time. This rule also applies when both the base class and the derived class are in the same assembly.
* **AllowPartiallyTrustedCallers** (level 2 only): All code defaults to transparent. However, individual types and members can have other attributes.

The following table compares the assembly level behavior for Level 2 with Level 1 .

|  |  |  |
| --- | --- | --- |
| **Assembly attribute** | **Level 2** | **Level 1** |
| No attribute on a partially trusted assembly | Types and members are by default transparent, but can be security-critical or security-safe-critical. | All types and members are transparent. |
| No attribute | Specifying no attribute causes the common language runtime to determine the transparency rules for you. All types and members are security-critical, except where being security-critical violates an inheritance rule. | On a fully trusted assembly (in the global assembly cache or identified as full trust in the **AppDomain**) all types are transparent and all members are security-safe-critical. |
| **SecurityTransparent** | All types and members are transparent. | All types and members are transparent. |
| **SecurityCritical(SecurityCriticalScope.Everything)** | Not applicable. | All types and members are security-critical. |
| **SecurityCritical** | All code that is introduced by types in this assembly is critical; all other code is transparent. If you override a virtual or abstract method or implement an interface method, you must explicitly annotate the method as **SecurityCritical** or**SecuritySafeCritical**. | All code defaults to transparent. However, individual types and members can have other attributes. |

[**Type and Member Annotation**](javascript:void(0))

The security attributes that are applied to a type also apply to the members that are introduced by the type. However, they do not apply to virtual or abstract overrides of the base class or interface implementations. The following rules apply to the use of attributes at the type and member level:

* **SecurityCritical**: The type or member is critical and can be called only by full-trust code. Methods that are introduced in a security-critical type are critical.

|  |
| --- |
| **Important noteImportant** |
| Virtual and abstract methods that are introduced in base classes or interfaces, and overridden or implemented in a security-critical class are transparent by default. They must be identified as either **SecuritySafeCritical** or **SecurityCritical**. |

* **SecuritySafeCritical**: The type or member is safe-critical. However, the type or member can be called from transparent (partially trusted) code and is as capable as any other critical code. The code must be audited for security.

[Back to top](https://msdn.microsoft.com/en-us/library/dd233102(v=vs.110).aspx#top)

[**Override Patterns**](javascript:void(0))

The following table shows the method overrides allowed for level 2 transparency.

|  |  |
| --- | --- |
| **Base virtual/interface member** | **Override/interface** |
| **Transparent** | **Transparent** |
| **Transparent** | **SafeCritical** |
| **SafeCritical** | **Transparent** |
| **SafeCritical** | **SafeCritical** |
| **Critical** | **Critical** |

[Back to top](https://msdn.microsoft.com/en-us/library/dd233102(v=vs.110).aspx#top)

[**Inheritance Rules**](javascript:void(0))

In this section, the following order is assigned to **Transparent**, **Critical**, and **SafeCritical** code based on access and capabilities:

**Transparent** < **SafeCritical** < **Critical**

* Rules for types: Going from left to right, access becomes more restrictive. Derived types must be at least as restrictive as the base type.
* Rules for methods: Derived methods cannot change accessibility from the base method. For default behavior, all derived methods that are not annotated are **Transparent**. Derivatives of critical types cause an exception to be thrown if the overridden method is not explicitly annotated as **SecurityCritical**.

The following table shows the allowed type inheritance patterns.

|  |  |
| --- | --- |
| **Base class** | **Derived class can be** |
| **Transparent** | **Transparent** |
| **Transparent** | **SafeCritical** |
| **Transparent** | **Critical** |
| **SafeCritical** | **SafeCritical** |
| **SafeCritical** | **Critical** |
| **Critical** | **Critical** |

The following table shows the disallowed type inheritance patterns.

|  |  |
| --- | --- |
| **Base class** | **Derived class cannot be** |
| **SafeCritical** | **Transparent** |
| **Critical** | **Transparent** |
| **Critical** | **SafeCritical** |

The following table shows the allowed method inheritance patterns.

|  |  |
| --- | --- |
| **Base method** | **Derived method can be** |
| **Transparent** | **Transparent** |
| **Transparent** | **SafeCritical** |
| **SafeCritical** | **Transparent** |
| **SafeCritical** | **SafeCritical** |
| **Critical** | **Critical** |

The following table shows the disallowed method inheritance patterns.

|  |  |  |
| --- | --- | --- |
| **Base method** | **Derived method cannot be** | |
| **Transparent** | **Critical** | |
| **SafeCritical** | **Critical** | |
| **Critical** | **Transparent** | |
| **Critical** | **SafeCritical** | |
| **Note Note** | |
| These inheritance rules apply to level 2 types and members. Types in level 1 assemblies can inherit from level 2 security-critical types and members. Therefore, level 2 types and members must have separate inheritance demands for level 1 inheritors. | |

[Back to top](https://msdn.microsoft.com/en-us/library/dd233102(v=vs.110).aspx#top)

[**Additional Information and Rules**](javascript:void(0))

[**LinkDemand Support**](javascript:void(0))

The level 2 transparency model replaces the [LinkDemand](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.110).aspx) with the [SecurityCriticalAttribute](https://msdn.microsoft.com/en-us/library/system.security.securitycriticalattribute(v=vs.110).aspx) attribute. In legacy (level 1) code, a[LinkDemand](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.110).aspx) is automatically treated as a [Demand](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.110).aspx).

[**Reflection**](javascript:void(0))

Invoking a critical method or reading a critical field triggers a demand for full trust (just as if you were invoking a private method or field). Therefore, full-trust code can invoke a critical method, whereas partial-trust code cannot.

The following properties have been added to the [System.Reflection](https://msdn.microsoft.com/en-us/library/system.reflection(v=vs.110).aspx) namespace to determine whether the type, method, or field is**SecurityCritical**, **SecuritySafeCritical**, or **SecurityTransparent**: [IsSecurityCritical](https://msdn.microsoft.com/en-us/library/dd267835(v=vs.110).aspx), [IsSecuritySafeCritical](https://msdn.microsoft.com/en-us/library/dd287589(v=vs.110).aspx), and [IsSecurityTransparent](https://msdn.microsoft.com/en-us/library/dd268420(v=vs.110).aspx). Use these properties to determine transparency by using reflection instead of checking for the presence of the attribute. The transparency rules are complex, and checking for the attribute may not be sufficient.

|  |
| --- |
| **Note Note** |
| A **SafeCritical** method returns **true** for both [IsSecurityCritical](https://msdn.microsoft.com/en-us/library/dd267835(v=vs.110).aspx)and [IsSecuritySafeCritical](https://msdn.microsoft.com/en-us/library/dd287589(v=vs.110).aspx), because **SafeCritical** is indeed critical (it has the same capabilities as critical code, but it can be called from transparent code). |

Dynamic methods inherit the transparency of the modules they are attached to; they do not inherit the transparency of the type (if they are attached to a type).

[**Skip Verification in Full Trust**](javascript:void(0))

You can skip verification for fully trusted transparent assemblies by setting the [SkipVerificationInFullTrust](https://msdn.microsoft.com/en-us/library/dd269628(v=vs.110).aspx) property to **true** in the[SecurityRulesAttribute](https://msdn.microsoft.com/en-us/library/system.security.securityrulesattribute(v=vs.110).aspx) attribute:

[assembly: SecurityRules(SecurityRuleSet.Level2, SkipVerificationInFullTrust = true)]

The [SkipVerificationInFullTrust](https://msdn.microsoft.com/en-us/library/dd269628(v=vs.110).aspx) property is **false** by default, so the property must be set to **true** to skip verification. This should be done for optimization purposes only. You should ensure that the transparent code in the assembly is verifiable by using the transparentoption in the [PEVerify tool](https://msdn.microsoft.com/en-us/library/62bwd2yd(v=vs.110).aspx).

#### Code Access Security Policy Compatibility and Migration

**.NET Framework 4.6 and 4.5**

[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635736413806036745

The policy portion of code access security (CAS) has been made obsolete in the .NET Framework 4. As a result, you may encounter compilation warnings and runtime exceptions if you call the obsolete policy types and members [explicitly](https://msdn.microsoft.com/en-us/library/ee191568(v=vs.110).aspx#explicit_use) or [implicitly](https://msdn.microsoft.com/en-us/library/ee191568(v=vs.110).aspx#implicit_use) (through other types and members).

You can avoid the warnings and errors by either:

* [Migrating](https://msdn.microsoft.com/en-us/library/ee191568(v=vs.110).aspx#migration) to the .NET Framework 4 replacements for the obsolete calls.

- or -

* Using the [<NetFx40\_LegacySecurityPolicy> configuration element](https://msdn.microsoft.com/en-us/library/dd409253(v=vs.110).aspx) to opt into the legacy CAS policy behavior.

This topic contains the following sections:

* [Explicit Use](https://msdn.microsoft.com/en-us/library/ee191568(v=vs.110).aspx#explicit_use)
* [Implicit Use](https://msdn.microsoft.com/en-us/library/ee191568(v=vs.110).aspx#implicit_use)
* [Errors and Warnings](https://msdn.microsoft.com/en-us/library/ee191568(v=vs.110).aspx#errors_and_warnings)
* [Migration: Replacement for Obsolete Calls](https://msdn.microsoft.com/en-us/library/ee191568(v=vs.110).aspx#migration)
* [Compatibility: Using the CAS Policy Legacy Option](https://msdn.microsoft.com/en-us/library/ee191568(v=vs.110).aspx#compatibility)

|  |
| --- |
| **Caution note Caution** |
| Code Access Security and Partially Trusted Code  The .NET Framework provides a mechanism for the enforcement of varying levels of trust on different code running in the same application called Code Access Security (CAS).  Code Access Security in .NET Framework should not  be used as a security boundary with partially trusted code, especially code of unknown origin. We advise against loading and executing code of unknown origins without putting alternative security measures in place.  This policy applies to all versions of .NET Framework, but does not apply to the .NET Framework included in Silverlight. |

[**Explicit Use**](javascript:void(0))

Members that directly manipulate security policy or require CAS policy to sandbox are obsolete and will produce errors by default.

Examples of these are:

* [AppDomain.SetAppDomainPolicy](https://msdn.microsoft.com/en-us/library/system.appdomain.setappdomainpolicy(v=vs.110).aspx)
* [HostSecurityManager.DomainPolicy](https://msdn.microsoft.com/en-us/library/system.security.hostsecuritymanager.domainpolicy(v=vs.110).aspx)
* [PolicyLevel.CreateAppDomainLevel](https://msdn.microsoft.com/en-us/library/system.security.policy.policylevel.createappdomainlevel(v=vs.110).aspx)
* [SecurityManager.LoadPolicyLevelFromString](https://msdn.microsoft.com/en-us/library/system.security.securitymanager.loadpolicylevelfromstring(v=vs.110).aspx)
* [SecurityManager.LoadPolicyLevelFromFile](https://msdn.microsoft.com/en-us/library/system.security.securitymanager.loadpolicylevelfromfile(v=vs.110).aspx)
* [SecurityManager.ResolvePolicy](https://msdn.microsoft.com/en-us/library/system.security.securitymanager.resolvepolicy(v=vs.110).aspx)
* [SecurityManager.ResolveSystemPolicy](https://msdn.microsoft.com/en-us/library/ms147633(v=vs.110).aspx)
* [SecurityManager.ResolvePolicyGroups](https://msdn.microsoft.com/en-us/library/system.security.securitymanager.resolvepolicygroups(v=vs.110).aspx)
* [SecurityManager.PolicyHierarchy](https://msdn.microsoft.com/en-us/library/system.security.securitymanager.policyhierarchy(v=vs.110).aspx)
* [SecurityManager.SavePolicy](https://msdn.microsoft.com/en-us/library/system.security.securitymanager.savepolicy(v=vs.110).aspx)

[**Implicit Use**](javascript:void(0))

Several assembly loading overloads produce errors because of their implicit use of CAS policy. These overloads take an [Evidence](https://msdn.microsoft.com/en-us/library/system.security.policy.evidence(v=vs.110).aspx) parameter that is used to resolve CAS policy and provide a permission grant set for an assembly.

Here are some examples. The obsolete overloads are those that take [Evidence](https://msdn.microsoft.com/en-us/library/system.security.policy.evidence(v=vs.110).aspx) as a parameter:

* [Activator.CreateInstanceFrom](https://msdn.microsoft.com/en-us/library/system.activator.createinstancefrom(v=vs.110).aspx)
* [AppDomain.CreateInstanceFrom](https://msdn.microsoft.com/en-us/library/system.appdomain.createinstancefrom(v=vs.110).aspx)
* [AppDomain.CreateInstanceAndUnwrap](https://msdn.microsoft.com/en-us/library/system.appdomain.createinstanceandunwrap(v=vs.110).aspx)
* [AppDomain.DefineDynamicAssembly](https://msdn.microsoft.com/en-us/library/system.appdomain.definedynamicassembly(v=vs.110).aspx)
* [AppDomain.ExecuteAssemblyByName](https://msdn.microsoft.com/en-us/library/system.appdomain.executeassemblybyname(v=vs.110).aspx)
* [AppDomain.Load](https://msdn.microsoft.com/en-us/library/system.appdomain.load(v=vs.110).aspx)
* [Assembly.LoadFrom](https://msdn.microsoft.com/en-us/library/system.reflection.assembly.loadfrom(v=vs.110).aspx)
* [Assembly.Load](https://msdn.microsoft.com/en-us/library/system.reflection.assembly.load(v=vs.110).aspx)
* [Assembly.LoadFrom](https://msdn.microsoft.com/en-us/library/system.reflection.assembly.loadfrom(v=vs.110).aspx)

[**Errors and Warnings**](javascript:void(0))

The obsolete types and members produce the following error messages when they are used. Note that the [System.Security.Policy.Evidence](https://msdn.microsoft.com/en-us/library/system.security.policy.evidence(v=vs.110).aspx)type itself is not obsolete.

Compile-time warning:

warning CS0618: '<API Name>' is obsolete: 'This method is obsolete and will be removed in a future release of the .NET Framework. Please use <suggested alternate API>. See <link> for more information.'

Run-time exception:

[NotSupportedException](https://msdn.microsoft.com/en-us/library/system.notsupportedexception(v=vs.110).aspx) : This method uses CAS policy, which has been obsoleted by the .NET Framework. In order to enable CAS policy for compatibility reasons, please use the <NetFx40\_LegacySecurityPolicy> configuration switch. Please see <link> for more information.

[**Migration: Replacement for Obsolete Calls**](javascript:void(0))

[**Determining an Assembly’s Trust Level**](javascript:void(0))

CAS policy is often used to determine an assembly’s or application domain’s permission grant set or trust level. The .NET Framework 4 exposes the following useful properties that do not need to resolve security policy:

* [Assembly.PermissionSet](https://msdn.microsoft.com/en-us/library/dd413895(v=vs.110).aspx)
* [Assembly.IsFullyTrusted](https://msdn.microsoft.com/en-us/library/dd413989(v=vs.110).aspx)
* [AppDomain.PermissionSet](https://msdn.microsoft.com/en-us/library/dd383500(v=vs.110).aspx)
* [AppDomain.IsFullyTrusted](https://msdn.microsoft.com/en-us/library/dd414841(v=vs.110).aspx)

[**Application Domain Sandboxing**](javascript:void(0))

The [AppDomain.SetAppDomainPolicy](https://msdn.microsoft.com/en-us/library/system.appdomain.setappdomainpolicy(v=vs.110).aspx) method is typically used for sandboxing the assemblies in an application domain. The .NET Framework 4 exposes members that do not have to use [PolicyLevel](https://msdn.microsoft.com/en-us/library/system.security.policy.policylevel(v=vs.110).aspx) for this purpose. For more information, see [How to: Run Partially Trusted Code in a Sandbox](https://msdn.microsoft.com/en-us/library/bb763046(v=vs.110).aspx).

[**Determining a Safe or Reasonable Permission Set for Partially Trusted Code**](javascript:void(0))

Hosts often need to determine the permissions that are appropriate for sandboxing hosted code. Before the .NET Framework 4, CAS policy provided a way to do this with the [SecurityManager.ResolvePolicy](https://msdn.microsoft.com/en-us/library/system.security.securitymanager.resolvepolicy(v=vs.110).aspx) method. As a replacement, .NET Framework 4 provides the[SecurityManager.GetStandardSandbox](https://msdn.microsoft.com/en-us/library/dd414122(v=vs.110).aspx) method, which returns a safe, standard permission set for the provided evidence.

[**Non-Sandboxing Scenarios: Overloads for Assembly Loads**](javascript:void(0))

The reason for using an assembly load overload might be to use parameters that are not otherwise available, instead of sandboxing the assembly. Starting with the .NET Framework 4, assembly load overloads that do not require a [System.Security.Policy.Evidence](https://msdn.microsoft.com/en-us/library/system.security.policy.evidence(v=vs.110).aspx) object as a parameter, for example, [AppDomain.ExecuteAssembly(String, String[], Byte[], AssemblyHashAlgorithm)](https://msdn.microsoft.com/en-us/library/dd414750(v=vs.110).aspx), enable this scenario.

If you want to sandbox an assembly, use the [AppDomain.CreateDomain(String, Evidence, AppDomainSetup, PermissionSet,StrongName[])](https://msdn.microsoft.com/en-us/library/ms130766(v=vs.110).aspx) overload.

[**Compatibility: Using the CAS Policy Legacy Option**](javascript:void(0))

The [<NetFx40\_LegacySecurityPolicy> configuration element](https://msdn.microsoft.com/en-us/library/dd409253(v=vs.110).aspx) lets you specify that a process or library uses legacy CAS policy. When you enable this element, the policy and evidence overloads will work as they did in previous versions of the framework.

|  |
| --- |
| **Note Note** |
| CAS policy behavior is specified on a runtime version basis, so modifying CAS policy for one runtime version does not affect the CAS policy of another version. |

<configuration>

<runtime>

<NetFx40\_LegacySecurityPolicy enabled="true"/>

</runtime>

</configuration>

### Using Libraries from Partially Trusted Code

**.NET Framework 1.1, 2.0, 3.0, 3.5**

Applications that receive less than full trust by the runtime code access security system are not allowed to call shared managed libraries unless the library writer specifically allows them to through the use of the [AllowPartiallyTrustedCallersAttribute Class](https://msdn.microsoft.com/en-us/library/system.security.allowpartiallytrustedcallersattribute(v=vs.71).aspx). Therefore, application writers must be aware that some libraries will not be available to them from a partially trusted context. By default, all code that executes from the local intranet or Internet zones is partially trusted. If you do not expect your code to be executed from a partially trusted context or to be called by partially trusted code, you do not need to be concerned about the information in this section. However, if you write code that must interact with partially trusted code or operate from a partially trusted context, you should consider the following factors:

* Libraries must be signed with a strong name in order to be shared by multiple applications. Strong names allow your code to be placed in the global assembly cache and allow consumers to verify that a particular piece of mobile code actually originates from you.
* By default, strong-named shared libraries perform an implicit [LinkDemand](https://msdn.microsoft.com/en-us/library/hzsc022c(v=vs.71).aspx) for full trust automatically, without the library writer having to do anything.
* If a caller does not have full trust but still tries to call such a library, the runtime throws a [SecurityException](https://msdn.microsoft.com/en-us/library/system.security.securityexception(v=vs.71).aspx) and the caller is not allowed to link to the library.
* In order to disable the automatic **LinkDemand** and prevent the exception from being thrown, you can place the**AllowPartiallyTrustedCallersAttribute** attribute on the assembly scope of a shared library. This attribute allows your libraries to be called from partially trusted managed code.
* Partially trusted code that is granted access to a library with this attribute is still subject to further restrictions defined by local machine policy.
* There is no programmatic way for partially trusted code to call a library that does not have the **AllowPartiallyTrustedCallersAttribute**attribute. If an application does not receive full trust by default, an administrator must choose to modify security policy and grant the application full trust before it can call such a library.

Libraries that are private to a specific application do not require a strong name or the **AllowPartiallyTrustedCallersAttribute** attribute and cannot be referenced by potentially malicious code outside the application. Such code is protected against intentional or unintentional misuse by partially trusted mobile code without the developer or administrator having to do anything extra.

You should consider explicitly enabling use by partially trusted code for the following types of code:

* Code that has been diligently tested for security vulnerabilities and is in compliance with the guidelines described in [Secure Coding Guidelines](https://msdn.microsoft.com/en-us/library/d55zzx87(v=vs.71).aspx).
* Strong-named code libraries that are specifically written for partially trusted scenarios.
* Any components (whether partially or fully trusted) signed with a strong name that will be called by mobile code downloaded from the Internet or the local intranet. These components are affected because under default security policy mobile code receives partial trust.
* If default policy is modified, any code that security policy grants less than full trust.

**Note**Some classes shipped in the .NET Framework class library do not have the**AllowPartiallyTrustedCallersAttribute** attribute and cannot be called by partially trusted code. See [.NET Framework Assemblies Marked with AllowPartiallyTrustedCallersAttribute](https://msdn.microsoft.com/en-us/library/ys8yafkf(v=vs.71).aspx) for a list of classes that are callable by partially trusted code.

### Using Libraries from Partially Trusted Code

**.NET Framework 4, 4.6 and 4.5**

|  |
| --- |
| **Note Note** |
| This topic addresses the behavior of strong-named assemblies and applies only to [Level 1](https://msdn.microsoft.com/en-us/library/bb397858(v=vs.110).aspx) assemblies. [Security-Transparent Code, Level 2](https://msdn.microsoft.com/en-us/library/dd233102(v=vs.110).aspx)assemblies in the .NET Framework 4 or later are not affected by strong names. For more information about changes to the security system, see [Security Changes in the .NET Framework](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.110).aspx). |

Applications that receive less than full trust from their host or sandbox are not allowed to call shared managed libraries unless the library writer specifically allows them to through the use of the [AllowPartiallyTrustedCallersAttribute attribute](https://msdn.microsoft.com/en-us/library/system.security.allowpartiallytrustedcallersattribute(v=vs.110).aspx). Therefore, application writers must be aware that some libraries will not be available to them from a partially trusted context. By default, all code that executes in a partial-trust[sandbox](https://msdn.microsoft.com/en-us/library/bb763046(v=vs.110).aspx) and is not in the list of full-trust assemblies is partially trusted. If you do not expect your code to be executed from a partially trusted context or to be called by partially trusted code, you do not have to be concerned about the information in this section. However, if you write code that must interact with partially trusted code or operate from a partially trusted context, you should consider the following factors:

* Libraries must be signed with a strong name in order to be shared by multiple applications. Strong names allow your code to be placed in the global assembly cache or added to the full-trust list of a sandboxing [AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.110).aspx), and allow consumers to verify that a particular piece of mobile code actually originates from you.
* By default, strong-named [Level 1](https://msdn.microsoft.com/en-us/library/bb397858(v=vs.110).aspx) shared libraries perform an implicit [LinkDemand](https://msdn.microsoft.com/en-us/library/hzsc022c(v=vs.110).aspx) for full trust automatically, without the library writer having to do anything.
* If a caller does not have full trust but still tries to call such a library, the runtime throws a [SecurityException](https://msdn.microsoft.com/en-us/library/system.security.securityexception(v=vs.110).aspx) and the caller is not allowed to link to the library.
* In order to disable the automatic **LinkDemand** and prevent the exception from being thrown, you can place the**AllowPartiallyTrustedCallersAttribute** attribute on the assembly scope of a shared library. This attribute allows your libraries to be called from partially trusted managed code.
* Partially trusted code that is granted access to a library with this attribute is still subject to further restrictions defined by the[AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.110).aspx).
* There is no programmatic way for partially trusted code to call a library that does not have the **AllowPartiallyTrustedCallersAttribute**attribute.

Libraries that are private to a specific application do not require a strong name or the **AllowPartiallyTrustedCallersAttribute** attribute and cannot be referenced by potentially malicious code outside the application. Such code is protected against intentional or unintentional misuse by partially trusted mobile code without the developer having to do anything extra.

You should consider explicitly enabling use by partially trusted code for the following types of code:

* Code that has been diligently tested for security vulnerabilities and is in compliance with the guidelines described in [Secure Coding Guidelines](https://msdn.microsoft.com/en-us/library/8a3x2b7f(v=vs.110).aspx).
* Strong-named code libraries that are specifically written for partially trusted scenarios.
* Any components (whether partially or fully trusted) signed with a strong name that will be called by code that is downloaded from the Internet.

|  |
| --- |
| **Note Note** |
| Some classes in the .NET Framework class library do not have the **AllowPartiallyTrustedCallersAttribute** attribute and cannot be called by partially trusted code. |

#### Deciding When To Enable Partially Trusted Callers

**.NET Framework 1.1, 2.0, 3.0, .3.5**

The implicit [LinkDemand](https://msdn.microsoft.com/en-us/library/hzsc022c(v=vs.71).aspx) for full trust that shared libraries perform by default helps provide your code a high level of protection by not allowing untrusted, potentially malicious code to call your code and exploit weaknesses. If you do not expect that your code will be called from partially trusted code, you should consider leaving the [AllowPartiallyTrustedCallersAttribute Class](https://msdn.microsoft.com/en-us/library/system.security.allowpartiallytrustedcallersattribute(v=vs.71).aspx) off your shared libraries.

However, there are certain scenarios in which you might want your shared library to be called by mobile or partially trusted code. For example, enterprise applications might want to support managed controls hosted in Microsoft Internet Explorer that run from Web sites on a local intranet. The **AllowPartiallyTrustedCallersAttribute** attribute should be applied only after you have considered the security implications and taken the necessary precautions, including code review against the secure coding guidelines for managed code. This attribute should be applied to assemblies only if the following criteria are met:

* Partially trusted code use is important to support.
* The assemblies have been designed and built with explicit attention to security considerations to make them robust against all callers, including potentially malicious callers.
* Appropriate security testing with partially trusted code is done before releasing the code.

#### Deciding When To Enable Partially Trusted Callers

**.NET Framework 4**

The implicit [LinkDemand](https://msdn.microsoft.com/en-us/library/hzsc022c(v=vs.100).aspx) for full trust that shared libraries perform by default helps provide your code a high level of protection by not allowing untrusted, potentially malicious code to call your code and exploit weaknesses. If you do not expect that your code will be called from partially trusted code, you should consider leaving the [AllowPartiallyTrustedCallersAttribute Class](https://msdn.microsoft.com/en-us/library/system.security.allowpartiallytrustedcallersattribute(v=vs.100).aspx) off your shared libraries.

|  |
| --- |
| **Important noteImportant** |
| In the .NET Framework version 4, partially trusted code has been redefined as transparent code. The transparency model draws a barrier between code that can do privileged things (critical code), such as calling native code, and code that cannot (transparent code). Transparency eliminates the use of the [LinkDemand](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx) to identify fully trusted code, and affects the rules for running partially trusted code. For more information about this and other changes, see [Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx). |
| **NoteNote** |
| The .NET Framework 4 introduces the conditional use of the [AllowPartiallyTrustedCallersAttribute](https://msdn.microsoft.com/en-us/library/system.security.allowpartiallytrustedcallersattribute(v=vs.100).aspx) (APTCA) attribute. Conditional APTCA enables hosts to identify which assemblies they want to expose to partial-trust callers that are loaded within the context of the host. The candidate assemblies must already be designed for partial trust; that is, they must either be APCTA (security-safe-critical in the transparency model) or fully transparent. A new constructor for the [AllowPartiallyTrustedCallersAttribute](https://msdn.microsoft.com/en-us/library/system.security.allowpartiallytrustedcallersattribute(v=vs.100).aspx) class enables the host to specify the level of visibility for an APTCA assembly by using the [PartialTrustVisibilityLevel](https://msdn.microsoft.com/en-us/library/system.security.partialtrustvisibilitylevel(v=vs.100).aspx) enumeration in the constructor call. |

However, there are certain scenarios in which you might want your shared library to be called by mobile or partially trusted code. For example, enterprise applications might want to support managed controls hosted in Microsoft Internet Explorer that run from Web sites on a local intranet. The [AllowPartiallyTrustedCallersAttribute](https://msdn.microsoft.com/en-us/library/system.security.allowpartiallytrustedcallersattribute(v=vs.100).aspx) attribute should be applied only after you have considered the security implications and taken the necessary precautions, including code review against the secure coding guidelines for managed code. This attribute should be applied to assemblies only if the following criteria are met:

* Partially trusted code use is important to support.
* The assemblies have been designed and built with explicit attention to security considerations to make them robust against all callers, including potentially malicious callers.
* Appropriate security testing with partially trusted code is done before releasing the code.

#### Sharing a Library with Partially Trusted Code

**.NET Framework 1.1, 2.0, 3.0, 3.5**

Several steps must be completed before shared libraries can be called by partially trusted code. For an overview of this issue, see [Using Libraries from Partially Trusted Code](https://msdn.microsoft.com/en-us/library/8skskf63(v=vs.71).aspx). You can enable partially trusted access to your libraries by completing the three following tasks:

**Strong Name the Assembly**

Before your library can be shared through the global assembly cache or used as a downloaded component, you must create a strong name. Complete the following tasks described in [Creating and Using Strong-Named Assemblies](https://msdn.microsoft.com/en-us/library/xwb8f617(v=vs.71).aspx) to strong name your library:

1. Create a key pair.
2. Use the [Assembly Linker (Al.exe)](https://msdn.microsoft.com/en-us/library/c405shex(v=vs.71).aspx) or an assembly attribute to apply the key pair to your library.

**Disable the LinkDemand**

To disable the implicit [LinkDemand](https://msdn.microsoft.com/en-us/library/hzsc022c(v=vs.71).aspx) that blocks partially trusted code from calling shared libraries, apply the[AllowPartiallyTrustedCallersAttribute](https://msdn.microsoft.com/en-us/library/system.security.allowpartiallytrustedcallersattribute(v=vs.71).aspx) at the assembly level of your code.

The following code example demonstrates how to apply the **AllowPartiallyTrustedCallersAttribute** attribute to a shared library.

VB

<assembly:AllowPartiallyTrustedCallersAttribute()>

[C#]

[assembly:AllowPartiallyTrustedCallersAttribute()]

When this attribute is present, all other security checks in your code still work as before, including any class-level or method-level declarative security attributes that are present. This attribute blocks only the implicit **LinkDemand** enforcement.

Share the Library

After the previous tasks are completed, you can share the library by placing it in the global assembly cache. For more information, see[Installing an Assembly into the Global Assembly Cache](https://msdn.microsoft.com/en-us/library/dkkx7f79(v=vs.71).aspx).

**Note**   If you want partially trusted callers to be able to call select classes or members in your library, you can place demands on those classes or members. For more information, see Requiring Full Trust for Types Within an AllowPartiallyTrustedCallersAttribute Assembly

See Also

[Code Access Security](https://msdn.microsoft.com/en-us/library/930b76w0(v=vs.71).aspx) | [Using Libraries from Partially Trusted Code](https://msdn.microsoft.com/en-us/library/8skskf63(v=vs.71).aspx)

#### Sharing a Library with Partially Trusted Code

**.NET Framework 4**

|  |
| --- |
| **NoteNote** |
| This topic addresses the use of strong-named assemblies. The following information applies only to [Level 1](https://msdn.microsoft.com/en-us/library/bb397858(v=vs.100).aspx) assemblies or assemblies that use legacy security policy. [Security-transparent code, level 2](https://msdn.microsoft.com/en-us/library/dd233102(v=vs.100).aspx) assemblies in the .NET Framework version 4 or later are not affected by strong names. For more information about changes to the security system, see [Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx). Several steps must be completed before shared libraries can be called by partially trusted code. For an overview of this issue, see [Using Libraries from Partially Trusted Code](https://msdn.microsoft.com/en-us/library/8skskf63(v=vs.100).aspx). You can enable partially trusted access to your libraries by completing the three tasks described in the following sections. |

[Assign a Strong Name to the Assembly](javascript:void(0))

Before your library can be shared through the global assembly cache or used as a downloaded component, you must create a strong name. Complete the following tasks described in [Creating and Using Strong-Named Assemblies](https://msdn.microsoft.com/en-us/library/xwb8f617(v=vs.100).aspx) to assign a strong name to your library:

1. Create a key pair.
2. Use the [Assembly Linker (Al.exe)](https://msdn.microsoft.com/en-us/library/c405shex(v=vs.100).aspx) or an assembly attribute to apply the key pair to your library.

[Disable the LinkDemand](javascript:void(0))

To disable the implicit [LinkDemand](https://msdn.microsoft.com/en-us/library/hzsc022c(v=vs.100).aspx) that blocks partially trusted code from calling shared libraries, apply the[AllowPartiallyTrustedCallersAttribute](https://msdn.microsoft.com/en-us/library/system.security.allowpartiallytrustedcallersattribute(v=vs.100).aspx) at the assembly level of your code.

The following code example demonstrates how to apply the [AllowPartiallyTrustedCallersAttribute](https://msdn.microsoft.com/en-us/library/system.security.allowpartiallytrustedcallersattribute(v=vs.100).aspx) attribute to a shared library.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/9ykfdse9(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

[assembly:AllowPartiallyTrustedCallersAttribute()]

When this attribute is present, all other security checks in your code still work as before, including any class-level or method-level declarative security attributes that are present. This attribute blocks only the implicit **LinkDemand** enforcement.

|  |
| --- |
| **NoteNote** |
| In the .NET Framework version 4, a new property has been added to the [AllowPartiallyTrustedCallersAttribute](https://msdn.microsoft.com/en-us/library/system.security.allowpartiallytrustedcallersattribute(v=vs.100).aspx) class that allows the attribute to be conditionally activated. For more information, see the [PartialTrustVisibilityLevel](https://msdn.microsoft.com/en-us/library/dd414118(v=vs.100).aspx) property. |

[Share the Library](javascript:void(0))

After the previous tasks are completed, you can share the library by placing it in the global assembly cache. For more information, see[Installing an Assembly into the Global Assembly Cache](https://msdn.microsoft.com/en-us/library/dkkx7f79(v=vs.100).aspx).

|  |
| --- |
| **NoteNote** |
| If you want to prevent partially trusted callers from calling select classes or members in your library, you can place demands on those classes or members. For more information, see [Requiring Full Trust for Types Within an AllowPartiallyTrustedCallersAttribute Assembly](https://msdn.microsoft.com/en-us/library/970x52db(v=vs.100).aspx). |

[**See Also**](javascript:void(0))

Concepts

[Using Libraries from Partially Trusted Code](https://msdn.microsoft.com/en-us/library/8skskf63(v=vs.100).aspx)

[Security-Transparent Code, Level 2](https://msdn.microsoft.com/en-us/library/dd233102(v=vs.100).aspx)

Other Resources

[Code Access Security](https://msdn.microsoft.com/en-us/library/c5tk9z76(v=vs.100).aspx)

#### Requiring Full Trust for Types Within an AllowPartiallyTrustedCallersAttribute Assembly

**.NET Framework 1.1, 2.0, 3.0, 3.5**

When the [AllowPartiallyTrustedCallersAttribute](https://msdn.microsoft.com/en-us/library/system.security.allowpartiallytrustedcallersattribute(v=vs.71).aspx) is placed on an assembly, the implicit [LinkDemand](https://msdn.microsoft.com/en-us/library/hzsc022c(v=vs.71).aspx) requiring all callers to be fully trusted is disabled. In some situations, you might want certain members or classes in an assembly to be callable by partially trusted code, but want other members or classes in the same assembly be protected from partially trusted code. The following steps allow you to provide access to certain members or classes from partially trusted code, while still denying access to other members and classes from partially trusted code:

1. Sign your code with a strong name.
2. Apply the **AllowPartiallyTrustedCallersAttribute** attribute to your assembly so that partially trusted callers can use it.
3. Apply a demand for full trust to the specific member or class within the assembly from step 2 that you want protected from partially trusted callers.

The following are demands that can be placed on your code in order to limit access by partially trusted code:

* To cause a **LinkDemand** for full trust, apply the following to a class or member.

VB

<PermissionSetAttribute(SecurityAction.LinkDemand, Name:="FullTrust")>

[C#]

[PermissionSetAttribute(SecurityAction.LinkDemand, Name="FullTrust")]

* To cause a link demand requiring that callers have the ability to call unmanaged code, apply the following to a class or member.

VB

<SecurityPermissionAttribute(SecurityAction.LinkDemand, UnmanagedCode:=True)>

[C#]

[SecurityPermissionAttribute(SecurityAction.LinkDemand, UnmanagedCode=true)]

* To cause an inheritance demand for full trust, apply the following to a class or member.

VB

<PermissionSet(SecurityAction.InheritanceDemand, Name:="FullTrust")>

[C#]

[PermissionSet(SecurityAction.InheritanceDemand, Name="FullTrust")]

#### Requiring Full Trust for Types Within an AllowPartiallyTrustedCallersAttribute Assembly

**.NET Framework 1.1., 2.0, 3.0, 3.5, 4**

When the [AllowPartiallyTrustedCallersAttribute](https://msdn.microsoft.com/en-us/library/system.security.allowpartiallytrustedcallersattribute(v=vs.100).aspx) (APTCA) is placed on an assembly, the implicit [LinkDemand](https://msdn.microsoft.com/en-us/library/hzsc022c(v=vs.100).aspx) requiring all callers to be fully trusted is disabled. In some situations, you might want certain members or classes in an assembly to be callable by partially trusted code, but want other members or classes in the same assembly be protected from partially trusted code. The following steps allow you to provide access to certain members or classes from partially trusted code, while still denying access to other members and classes from partially trusted code:

1. Sign your code with a strong name.
2. Apply the **AllowPartiallyTrustedCallersAttribute** attribute to your assembly so that partially trusted callers can use it.
3. Apply a demand for full trust to the specific member or class within the assembly from step 2 that you want protected from partially trusted callers.

|  |
| --- |
| **NoteNote** |
| In the .NET Framework version 4, applying the [SecurityCriticalAttribute](https://msdn.microsoft.com/en-us/library/system.security.securitycriticalattribute(v=vs.100).aspx) attribute to a type or member in an APTCA assembly has the same effect as applying a link demand for full trust on a type or member. For more information, see [Security-Transparent Code, Level 2](https://msdn.microsoft.com/en-us/library/dd233102(v=vs.100).aspx). |

The following are demands that can be placed on your code in order to limit access by partially trusted code:

* To cause a **LinkDemand** for full trust, apply the following to a class or member.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/970x52db(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

[PermissionSetAttribute(SecurityAction.LinkDemand, Name="FullTrust")]

* To cause a link demand requiring that callers have the ability to call unmanaged code, apply the following to a class or member.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/970x52db(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

[SecurityPermissionAttribute(SecurityAction.LinkDemand, UnmanagedCode=true)]

* To cause an inheritance demand for full trust, apply the following to a class or member.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/970x52db(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-3)

[PermissionSet(SecurityAction.InheritanceDemand, Name="FullTrust")]

[**See Also**](javascript:void(0))

Concepts

[Using Libraries from Partially Trusted Code](https://msdn.microsoft.com/en-us/library/8skskf63(v=vs.100).aspx)

Other Resources

[Code Access Security](https://msdn.microsoft.com/en-us/library/c5tk9z76(v=vs.100).aspx)

#### .NET Framework Assemblies Marked with AllowPartiallyTrustedCallersAttribute

**.NET Framework 1.1**

To enable key scenarios for partially trusted code and minimize unintended use by partially trusted callers, several .NET Framework assemblies are marked for use by partially trusted code. Some code within these assemblies might be subject to security restrictions, including requiring callers to be fully trusted in some cases. All other assemblies that are part of the .NET Framework are intended for use only by fully trusted code.

The following .NET Framework assemblies have the [AllowPartiallyTrustedCallersAttribute](https://msdn.microsoft.com/en-us/library/system.security.allowpartiallytrustedcallersattribute(v=vs.71).aspx) attributeapplied:

* Accessibility.dll
* IEExecRemote.dll
* Microsoft.VisualBasic.dll
* Mscorlib.dll
* System.dll
* System.Data.dll
* System.Drawing.dll
* System.Web.dll (available only in version 1.1)
* System.Web.Mobile.dll (available only in version 1.1)
* System.Web.Services.dll
* System.Web.RegularExpressions.dll (available only in version 1.1)
* System.Windows.Forms.dll
* System.XML.dll

**.NET Framework 2.0**

The following .NET Framework assemblies have the [AllowPartiallyTrustedCallersAttribute](https://msdn.microsoft.com/en-us/library/system.security.allowpartiallytrustedcallersattribute(v=vs.80).aspx) attribute applied:

* Accessibility.dll
* IEExecRemote.dll
* Microsoft.JScript.dll (applied in version 2.0)
* Microsoft.VisualBasic.dll
* Mscorlib.dll
* System.dll
* System.Configuration.dll (applied in version 2.0)
* System.Data.dll
* System.Deployment.dll (applied in version 2.0)
* System.DirectoryServices.dll (applied in version 2.0)
* System.DirectoryServices.Protocols.dll (applied in version 2.0)
* System.Drawing.dll
* System.Security.dll (applied in version 2.0)
* System.Transactions.dll (applied in version 2.0)
* System.Web.dll (applied in versions 1.1 and 2.0)
* System.Web.Mobile.dll (applied in versions 1.1 and 2.0)
* System.Web.RegularExpressions.dll (applied in versions 1.1 and 2.0)
* System.Web.Services.dll
* System.Windows.Forms.dll
* System.XML.dll

**.NET Framework 3.0**

The following .NET Framework assemblies have the [AllowPartiallyTrustedCallersAttribute](https://msdn.microsoft.com/en-us/library/system.security.allowpartiallytrustedcallersattribute(v=vs.85).aspx) attribute applied:

* Accessibility.dll
* IEExecRemote.dll
* Microsoft.JScript.dll (applied in version 2.0)
* Microsoft.VisualBasic.dll
* Mscorlib.dll
* System.dll
* System.Configuration.dll (applied in version 2.0)
* System.Data.dll
* System.Deployment.dll (applied in version 2.0)
* System.DirectoryServices.dll (applied in version 2.0)
* System.DirectoryServices.Protocols.dll (applied in version 2.0)
* System.Drawing.dll
* System.Security.dll (applied in version 2.0)
* System.Transactions.dll (applied in version 2.0)
* System.Web.dll (applied in versions 1.1 and 2.0)
* System.Web.Mobile.dll (applied in versions 1.1 and 2.0)
* System.Web.RegularExpressions.dll (applied in versions 1.1 and 2.0)
* System.Web.Services.dll
* System.Windows.Forms.dll
* System.XML.dll

**.NET Framework 4.0**

The following .NET Framework assemblies have the [AllowPartiallyTrustedCallersAttribute](https://msdn.microsoft.com/en-us/library/system.security.allowpartiallytrustedcallersattribute(v=vs.100).aspx) (APTCA) or the [SecurityTransparentAttribute](https://msdn.microsoft.com/en-us/library/system.security.securitytransparentattribute(v=vs.100).aspx) attribute applied:

* Accessibility.dll (APTCA)
* Microsoft.JScript.dll (APTCA)
* Microsoft.VisualBasic.dll (APTCA)
* Mscorlib.dll (APTCA)
* System.dll (APTCA)
* System.Configuration.dll (APTCA)
* System.Data.dll (APTCA)
* System.Data.Linq.dll (security-transparent attribute)
* System.Deployment.dll (APTCA)
* System.DirectoryServices.dll (APTCA)
* System.DirectoryServices.Protocols.dll (APTCA)
* System.Drawing.dll (APTCA)
* System.Security.dll (APTCA)
* System.Transactions.dll (APTCA)
* System.Web.dll (APTCA)
* System.Web.Mobile.dll (APTCA)
* System.Web.RegularExpressions.dll (security-transparent attribute)
* System.Web.Services.dll (APTCA)
* System.Windows.Forms.dll (APTCA)
* System.XML.dll (APTCA)

#### .NET Framework Assemblies Marked with AllowPartiallyTrustedCallersAttribute

**.NET Framework 3.5**

To enable key scenarios for partially trusted code and minimize unintended use by partially trusted callers, several .NET Framework assemblies are marked for use by partially trusted code. Some code within these assemblies might be subject to security restrictions, including requiring callers to be fully trusted in some cases. All other assemblies that are part of the .NET Framework are intended for use only by fully trusted code.

The following .NET Framework assemblies have the [AllowPartiallyTrustedCallersAttribute](https://msdn.microsoft.com/en-us/library/system.security.allowpartiallytrustedcallersattribute(v=vs.90).aspx) attribute applied:

* Accessibility.dll
* IEExecRemote.dll
* Microsoft.JScript.dll (applied in version 2.0)
* Microsoft.VisualBasic.dll
* Mscorlib.dll
* System.dll
* System.Configuration.dll (applied in version 2.0)
* System.Data.dll
* System.Data.Linq.dll
* System.Deployment.dll (applied in version 2.0)
* System.DirectoryServices.dll (applied in version 2.0)
* System.DirectoryServices.Protocols.dll (applied in version 2.0)
* System.Drawing.dll
* System.Linq.dll
* System.Security.dll (applied in version 2.0)
* System.Transactions.dll (applied in version 2.0)
* System.Web.dll (applied in versions 1.1 and 2.0)
* System.Web.Mobile.dll (applied in versions 1.1 and 2.0)
* System.Web.RegularExpressions.dll (applied in versions 1.1 and 2.0)
* System.Web.Services.dll
* System.Windows.Forms.dll
* System.XML.dll

#### .NET Framework Assemblies Callable by Partially Trusted Code

**.NET Framework 4**

To enable key scenarios several .NET Framework assemblies can be called by partially trusted code. Some code within these assemblies might be subject to security restrictions, including requiring callers to be fully trusted in some cases. All other assemblies that are part of the .NET Framework are intended for use only by fully trusted code.

The following .NET Framework assemblies have the [AllowPartiallyTrustedCallersAttribute](https://msdn.microsoft.com/en-us/library/system.security.allowpartiallytrustedcallersattribute(v=vs.100).aspx) (APTCA) or the [SecurityTransparentAttribute](https://msdn.microsoft.com/en-us/library/system.security.securitytransparentattribute(v=vs.100).aspx) attribute applied:

* Accessibility.dll (APTCA)
* Microsoft.JScript.dll (APTCA)
* Microsoft.VisualBasic.dll (APTCA)
* Mscorlib.dll (APTCA)
* System.dll (APTCA)
* System.Configuration.dll (APTCA)
* System.Data.dll (APTCA)
* System.Data.Linq.dll (security-transparent attribute)
* System.Deployment.dll (APTCA)
* System.DirectoryServices.dll (APTCA)
* System.DirectoryServices.Protocols.dll (APTCA)
* System.Drawing.dll (APTCA)
* System.Security.dll (APTCA)
* System.Transactions.dll (APTCA)
* System.Web.dll (APTCA)
* System.Web.Mobile.dll (APTCA)
* System.Web.RegularExpressions.dll (security-transparent attribute)
* System.Web.Services.dll (APTCA)
* System.Windows.Forms.dll (APTCA)
* System.XML.dll (APTCA)

### Writing Secure Class Libraries

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4**

Programming errors in class libraries can expose security vulnerabilities because class libraries often access protected resources and unmanaged code. If you design class libraries, you need to understand code access security and be careful to secure your class library.

The following table describes the three main elements you need to consider when securing a class library.

|  |  |
| --- | --- |
| **Security element** | **Description** |
| Security demand | Demands are applied at the class and method level as a mechanism for requiring that callers of your code have the permissions that you want them to have. Demands invoke a stack walk, in which all callers that directly or indirectly call your code are checked on the stack when your code is called. Demands are usually used in class libraries to help protect resources. |
| Security override | Overrides are applied on the class and method scope as a way to overrule certain security decisions made by the runtime. They are invoked when callers use your code. They are used to stop stack walks and limit the access of callers who have already been granted certain permissions.  **Caution**   Overrides can be dangerous and should be used with care. |
| Security optimization | A combination of demands and overrides can enhance performance during the interaction of your code and the security system. |

See Also

[Security Demands](https://msdn.microsoft.com/en-us/library/60zfc754(v=vs.71).aspx) | [Overriding Security Checks](https://msdn.microsoft.com/en-us/library/c2f7a0y2(v=vs.71).aspx) | [Declarative Security Used with Class and Member Scope](https://msdn.microsoft.com/en-us/library/cz02ke7h(v=vs.71).aspx) | [Security Optimizations](https://msdn.microsoft.com/en-us/library/ett3th5b(v=vs.71).aspx) | [Code Access Security](https://msdn.microsoft.com/en-us/library/930b76w0(v=vs.71).aspx)

#### Security Demands

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4.0**

To ensure that only callers that have been granted a specified permission can call your code, you can declaratively or imperatively demand that callers of your code have a specific permission or set of permissions. A demand causes the runtime to perform a security check to enforce restrictions on calling code. During a security check, the runtime walks the call stack, examining the permissions of each caller in the stack and determining whether the permission being demanded has been granted to each caller. If a caller that does not have the demanded permission is found, the security check fails and a [SecurityException](https://msdn.microsoft.com/en-us/library/system.security.securityexception(v=vs.71).aspx) is thrown. The only demands that do not result in a stack walk are [link demands](https://msdn.microsoft.com/en-us/library/hzsc022c(v=vs.71).aspx), which check only the immediate caller.

You can cause a security check to take place every time a particular method is called or before a particular block of code is executed. If you want the security check to occur when any member of a particular class is called, you can place a demand before the class so that it applies to every member of the class. The remainder of this topic explains how you make security demands, when you should do so, and why you might choose one type of security demand over another.

If you are writing a library that directly accesses a protected resource and if this access is exposed to the caller, you must make a security demand in the library to help verify that all callers in the call stack have permission to access that resource. Your demands can be [declarative](https://msdn.microsoft.com/en-us/library/9kc0c6st(v=vs.71).aspx)or [imperative](https://msdn.microsoft.com/en-us/library/9kc0c6st(v=vs.71).aspx). Note that demands should never be made in a class constructor because class constructor code is not guaranteed to execute at any particular point or in any particular context. Because the state of the call stack in a class constructor is not well defined, demands placed on class constructors can produce unexpected and undesired results.

You should use the following guidelines, regardless of the type of demand you make:

* Ensure that the caller originated from a particular site or zone, or was signed by a particular publisher, by demanding that callers have a particular identity permission. However, you should do this only when you are giving additional access based on matching an identity, not when you are denying access based on matching an identity. Because it is relatively simple to modify or mask code's identity, denying access based on identity alone is not a reliable way of protecting your code and the resources it accesses from unauthorized access.
* Ensure that an object can be created only by callers who have a specific permission by placing the demand on the class level of that object. For example, suppose you have a class called myFileStream, which derives from the [FileStream](https://msdn.microsoft.com/en-us/library/system.io.filestream(v=vs.71).aspx) class, and you want to ensure that only authorized callers can create instances of myFileStream. You would place a declarative demand for a [FileIOPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.fileiopermission(v=vs.71).aspx) object that represents the right to access the stream created by myFileStream on the class level of the myFileStream class.
* You can also put demands in code that set or get a property. In general, you put demands for less restrictive permissions on the get accessor rather than on the set accessor, unless the property holds sensitive information, such as a password.

**Note**   Role-based security checks have slightly different semantics than code access security checks. For more information, see [Role-Based Security](https://msdn.microsoft.com/en-us/library/52kd59t0(v=vs.71).aspx).

**Note**Demands can be applied at the class, method, event, and property levels only; assemblies and individual non-private variable members are not protected by demands. Demands placed on the assembly or non-private variable level will not produce a compiler warning. Therefore, it is important to use properties instead of public members in order to insure the protection that demands provide.

##### Demands

**.NET Framework 1.1**

You can use the security demand call declaratively or imperatively to specify the permissions that direct or indirect callers must have to access your library. Direct callers explicitly call static or instance methods of your library, while indirect callers call static or instance methods of another library that calls your library. When you use a demand, any application that includes your code will execute only if all direct and indirect callers have the permissions that the demand specifies. Demands are particularly useful in situations in which your class library uses protected resources that you do not want to be accessed by untrusted code. Demands can be placed in code using either imperative or declarative syntax.

Note that most classes in the .NET Framework already have demands associated with them, so you do not need to make an additional demand whenever you use a class that accesses a protected resource. For example, the **StreamWriter** class automatically makes a security demand for **FileIOPermission** whenever it is opened. If you make a demand for **FileIOPermission** when you use the **StreamWriter** class, you will cause a redundant and inefficient stack walk to occur. You should use demands to protect custom resources that require custom permissions.

Demands can be either declarative or imperative.

**Declarative Demands**

Declarative demands place information into your code's metadata using attributes. You can use declarative syntax to place a demand at either the class or the method level of your code.

If you place a declarative security check at the class level, it applies to each class member. However, if you place a declarative security check at the member level, it applies to only that member and overrides the permission specified at the class level, if one exists. For example, suppose you specify at the class level that PermissionA is required, and for that class's Method1 you indicate that PermissionB is required. When Method1 is called, a security check will look only for PermissionB, but other methods of the class will still require PermissionA.

The following example places a declarative demand for a custom permission called CustomPermission on all callers of the ReadDatamethod. This permission is a hypothetical custom permission and does not exist in the .NET Framework. The custom permission has a separately defined CustomPermissionAttribute that makes the demand. In this case, it takes a **SecurityAction.Demand** flag in order to specify the type of demand the attribute will perform.

VB

<CustomPermissionAttribute(SecurityAction.Demand, Unrestricted := True)>Public Shared Function ReadData() As String

'Read from a custom resource.

End Function

[C#]

[CustomPermissionAttribute(SecurityAction.Demand, Unrestricted = true)]

public static string ReadData()

{

//Read from a custom resource.

}

**Imperative Demands**

Imperative demands are placed at the method level of your code by creating a new instance of a permission object and calling that object's**Demand** method. Imperative syntax cannot be used to place demands at the class level.

The imperative demand that you place in your code effectively helps protect all the remaining code in the method in which the **Demand**method is called. The security check is performed when the **Demand** is executed; if the security check fails, a [SecurityException](https://msdn.microsoft.com/en-us/library/system.security.securityexception(v=vs.71).aspx) is thrown and the rest of the code in that method or member is never executed unless the **SecurityException** is caught and handled.

The following example uses imperative syntax to place a demand on all callers for the custom permission CustomPermission. This code creates a new instance of the CustomPermission class, passing the **PermissionState.Unrestricted** flag to the constructor. The **Demand**method is then called.

VB

Public Shared Sub ReadData()

Dim MyPermission As New CustomPermission(PermissionState.Unrestricted)

MyPermission.Demand()

'Read from a custom resource.

End Sub

[C#]

public static void ReadData()

{

CustomPermission MyPermission = new CustomPermission(PermissionState.Unrestricted);

MyPermission.Demand();

//Read from a custom resource.

}

See Also

[Extending Metadata Using Attributes](https://msdn.microsoft.com/en-us/library/5x6cd29c(v=vs.71).aspx) | [Security Demands](https://msdn.microsoft.com/en-us/library/60zfc754(v=vs.71).aspx) | [Code Access Security](https://msdn.microsoft.com/en-us/library/930b76w0(v=vs.71).aspx) | [Creating Your Own Code Access Permissions](https://msdn.microsoft.com/en-us/library/yctbsyf4(v=vs.71).aspx) | [Adding Declarative Security Support](https://msdn.microsoft.com/en-us/library/84kh7ht8(v=vs.71).aspx) | [Writing Secure Class Libraries](https://msdn.microsoft.com/en-us/library/e942ksxt(v=vs.71).aspx) | [SecurityException Class](https://msdn.microsoft.com/en-us/library/system.security.securityexception(v=vs.71).aspx)

##### Demands

**.NET Framework 2.0, 3.0, 4.0**

You can use the security demand call declaratively or imperatively to specify the permissions that direct or indirect callers must have to access your library. Direct callers explicitly call static or instance methods of your library, while indirect callers call static or instance methods of another library that calls your library. When you use a demand, any application that includes your code will execute only if all direct and indirect callers have the permissions that the demand specifies. Demands are particularly useful in situations in which your class library uses protected resources that you do not want to be accessed by untrusted code. Demands can be placed in code using either imperative or declarative syntax.

Note that most classes in the .NET Framework already have demands associated with them, so you do not need to make an additional demand whenever you use a class that accesses a protected resource. For example, the **StreamWriter** class automatically makes a security demand for **FileIOPermission** whenever it is opened. If you make a demand for **FileIOPermission** when you use the **StreamWriter** class, you will cause a redundant and inefficient stack walk to occur. You should use demands to protect custom resources that require custom permissions.

Demands can be either declarative or imperative.

**Stack Walks**

Demands enforce security by performing an analysis (called a stack walk) in which every calling function (or stack frame) in the current call stack is examined for the specified permission. When a demand is triggered the following occurs.

* The stack walk begins at the callers stack frame not the current stack where the demand occurs. For example, if Method A calls Method B and method B has a demand, the stack walk begins at method A's stack frame. Method B is never evaluated as part of the stack walk.
* The stack walk proceeds through the call stack until it reaches the program entry point of the stack (usually the **Main** method) or until a stack walk modifier like an assert is found. For information on stack walk modifiers see, [Overriding Security Checks](https://msdn.microsoft.com/en-us/library/c2f7a0y2(v=vs.80).aspx).
* When a demand and a stack walk modifier (an assert, for example) for the same permission appear on the same stack frame, the demand takes precedence.
* Declarative and imperative syntax exhibit no difference in behavior.
* Note that a demand placed on your program entry point never gets evaluated because the stack walks always begins at calling stack frame, but in this case, there is no such calling frame to evaluate. Therefore demands placed on a program entry point always succeed.

**Declarative Demands**

Declarative demands place information into your code's metadata using attributes. You can use declarative syntax to place a demand at either the class or the method level of your code.

If you place a declarative security check at the class level, it applies to each class member. However, if you place a declarative security check at the member level, it applies to only that member and overrides the permission specified at the class level, if one exists. For example, suppose you specify at the class level that PermissionA is required, and for that class's Method1 you indicate that PermissionB is required. When Method1 is called, a security check will look only for PermissionB, but other methods of the class will still require PermissionA.

The following example places a declarative demand for a custom permission called CustomPermission on all callers of the ReadDatamethod. This permission is a hypothetical custom permission and does not exist in the .NET Framework. The custom permission has a separately defined CustomPermissionAttribute that makes the demand. In this case, it takes a **SecurityAction.Demand** flag in order to specify the type of demand the attribute will perform.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/9kc0c6st(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

[CustomPermissionAttribute(SecurityAction.Demand, Unrestricted = true)]

public static string ReadData()

{

//Read from a custom resource.

}

**Imperative Demands**

Imperative demands are placed at the method level of your code by creating a new instance of a permission object and calling that object's**Demand** method. Imperative syntax cannot be used to place demands at the class level.

The imperative demand that you place in your code effectively helps protect all the remaining code in the method in which the **Demand**method is called. The security check is performed when the **Demand** is executed; if the security check fails, a [SecurityException](https://msdn.microsoft.com/en-us/library/system.security.securityexception(v=vs.80).aspx) is thrown and the rest of the code in that method or member is never executed unless the **SecurityException** is caught and handled.

The following example uses imperative syntax to place a demand on all callers for the custom permission CustomPermission. This code creates a new instance of the CustomPermission class, passing the **PermissionState.Unrestricted** flag to the constructor. The **Demand**method is then called.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/9kc0c6st(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

public static void ReadData()

{

CustomPermission MyPermission = new CustomPermission(PermissionState.Unrestricted);

MyPermission.Demand();

//Read from a custom resource.

}

|  |
| --- |
| **NoteNote** |
| The optimization behavior for the demand operation differs between 64 bit and 32 bit platforms. On 64 bit platforms, a demand will not check the grant set of the assembly containing the demand in cases where no other calling assemblies are present. However, this optimization does not cause an elevation of privilege because a stack walk is still performed when calling assemblies are present. On 32 bit platforms the demand operation checks the grant set of the assembly containing the demand and all calling assemblies. |

See Also

Reference

[SecurityException Class](https://msdn.microsoft.com/en-us/library/system.security.securityexception(v=vs.80).aspx)

Concepts

[Security Demands](https://msdn.microsoft.com/en-us/library/60zfc754(v=vs.80).aspx)  
[Creating Your Own Code Access Permissions](https://msdn.microsoft.com/en-us/library/yctbsyf4(v=vs.80).aspx)  
[Adding Declarative Security Support](https://msdn.microsoft.com/en-us/library/84kh7ht8(v=vs.80).aspx)  
[Writing Secure Class Libraries](https://msdn.microsoft.com/en-us/library/e942ksxt(v=vs.80).aspx)

Other Resources

[Extending Metadata Using Attributes](https://msdn.microsoft.com/en-us/library/5x6cd29c(v=vs.80).aspx)  
[Code Access Security](https://msdn.microsoft.com/en-us/library/930b76w0(v=vs.80).aspx)

##### Link Demands

**.NET Framework 1.1**

A link demand causes a security check during just-in-time compilation and checks only the immediate caller of your code. Linking occurs when your code is bound to a type reference, including function pointer references and method calls. If the caller does not have sufficient permission to link to your code, the link is not allowed and a runtime exception is thrown when the code is loaded and run. Link demands can be overridden in classes that inherit from your code.

Note that a full stack walk is not performed with this type of demand and that your code is still susceptible to luring attacks. The link demand specifies only the permissions direct callers must have to link to your code. It does not specify the permissions all callers must have to run your code.

If a method protected by a link demand is accessed through [reflection](https://msdn.microsoft.com/en-us/library/cxz4wk15(v=vs.71).aspx), than a link demand checks the immediate caller of the code accessed through reflection. This is true both for method discovery and for method invocation performed using reflection. For example, suppose code uses reflection to return a [MethodInfo](https://msdn.microsoft.com/en-us/library/system.reflection.methodinfo(v=vs.71).aspx) object representing a method protected by a link demand and then passes that **MethodInfo** object to some other code that uses the object to invoke the original method. In this case the link demand check occurs twice: once for the code that returns the **MethodInfo** object and once for the code that invokes it.

**Note**   A link demand performed on a static class constructor does not protect the constructor because static constructors are called by the system, outside the application's code execution path. As a result, when a link demand is applied to an entire class, it cannot protect access to a static constructor, although it does protect the rest of the class.

The following code fragment declaratively specifies that any code linking to the ReadData method must have the CustomPermissionpermission. This permission is a hypothetical custom permission and does not exist in the .NET Framework. The demand is made by passing a**SecurityAction.LinkDemand** flag to the CustomPermissionAttribute.

VB

<CustomPermissionAttribute(SecurityAction.LinkDemand)> Public Shared Function ReadData() As String

'Access a custom resource.

End Function

[C#]

[CustomPermissionAttribute(SecurityAction.LinkDemand)]

public static string ReadData()

{

//Access a custom resource.

}

##### Link Demands

**.NET Framework 2.0, 3.0, 3.5, 4.0, 4.5, 4.6**

A link demand causes a security check during just-in-time compilation and checks only the immediate calling assembly of your code. Linking occurs when your code is bound to a type reference, including function pointer references and method calls. If the calling assembly does not have sufficient permission to link to your code, the link is not allowed and a runtime exception is thrown when the code is loaded and run. Link demands can be overridden in classes that inherit from your code.

Note that a full stack walk is not performed with this type of demand and that your code is still susceptible to luring attacks. For example, if a method in assembly A is protected by a link demand, a direct caller in assembly B is evaluated based on the permissions of Assembly B. However, the link demand will not evaluate a method in assembly C if it indirectly calls the method in assembly A using the method in assembly B. The link demand specifies only the permissions direct callers in the immediate calling assembly must have to link to your code. It does not specify the permissions all callers must have to run your code.

The [Assert](https://msdn.microsoft.com/en-us/library/c82hh6x8(v=vs.80).aspx), [Deny](https://msdn.microsoft.com/en-us/library/0ebw73h9(v=vs.80).aspx), and [PermitOnly](https://msdn.microsoft.com/en-us/library/a3z81ks0(v=vs.80).aspx) stack walk modifiers do not affect the evaluation of link demands. Because link demands do not perform a stack walk, the stack walk modifiers have no effect on link demands.

If a method protected by a link demand is accessed through [reflection](https://msdn.microsoft.com/en-us/library/cxz4wk15(v=vs.80).aspx), than a link demand checks the immediate caller of the code accessed through reflection. This is true both for method discovery and for method invocation performed using reflection. For example, suppose code uses reflection to return a [MethodInfo](https://msdn.microsoft.com/en-us/library/system.reflection.methodinfo(v=vs.80).aspx) object representing a method protected by a link demand and then passes that **MethodInfo** object to some other code that uses the object to invoke the original method. In this case the link demand check occurs twice: once for the code that returns the **MethodInfo** object and once for the code that invokes it.

|  |
| --- |
| **NoteNote** |
| A link demand performed on a static class constructor does not protect the constructor because static constructors are called by the system, outside the application's code execution path. As a result, when a link demand is applied to an entire class, it cannot protect access to a static constructor, although it does protect the rest of the class. |

The following code fragment declaratively specifies that any code linking to the ReadData method must have the CustomPermissionpermission. This permission is a hypothetical custom permission and does not exist in the .NET Framework. The demand is made by passing a**SecurityAction.LinkDemand** flag to the CustomPermissionAttribute.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/hzsc022c(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

[CustomPermissionAttribute(SecurityAction.LinkDemand)]

public static string ReadData()

{

// Access a custom resource.

}

See Also

Concepts

[Security Demands](https://msdn.microsoft.com/en-us/library/60zfc754(v=vs.80).aspx)  
[Creating Your Own Code Access Permissions](https://msdn.microsoft.com/en-us/library/yctbsyf4(v=vs.80).aspx)  
[Adding Declarative Security Support](https://msdn.microsoft.com/en-us/library/84kh7ht8(v=vs.80).aspx)

Other Resources

[Extending Metadata Using Attributes](https://msdn.microsoft.com/en-us/library/5x6cd29c(v=vs.80).aspx)  
[Code Access Security](https://msdn.microsoft.com/en-us/library/930b76w0(v=vs.80).aspx)

##### Inheritance Demands

**.NET Framework 1.1, 2.0, 3.0, 3.5**

Inheritance demands applied to classes have a different meaning than inheritance demands applied to methods. You can place inheritance demands at the class level to ensure that only code with the specified permission can inherit from your class. Inheritance demands placed on methods require that code have the specified permission to override the method.

**Class Inheritance Demands**

An inherited demand applied to a class has the effect of demanding that all classes derived from the parent class have the specified permission. For example, if class B is to inherit from class A and class A is protected by an inheritance demand, then B must be granted that permission in order to run. If class B is granted that permission and derives from class A, then class C must also have the permission demanded by A, if it is to derive from B. This demand can be applied only declaratively.

The following example uses an inheritance demand to require that any class that inherits from this class must have the custom permissionCustomPermissionAttribute. This permission is a hypothetical custom permission and does not exist in the .NET Framework. This demand is made by passing the CustomPermissionAttribute a **SecurityAction.InheritanceDemand** enumeration.

VB

<CustomPermissionAttribute(SecurityAction.InheritanceDemand)> Public Class MyClass1

Public Sub New()

End Sub

Public Overridable Function ReadData() As String

'Access a custom resource.

End Function

End Class

[C#]

[CustomPermissionAttribute(SecurityAction.InheritanceDemand)]

public class MyClass

{

public MyClass()

{

}

public virtual string ReadData()

{

//Access a custom resource.

}

}

**Method Inheritance Demands**

If you place an inheritance demand at the method level, the specified permission will be applied to all overridden methods in a derived class. By using this declaration on specific methods, it is possible to selectively control the ability of derived classes to override the methods.

The following example specifies that all classes that derive from MyClass must have the CustomPermission permission to override theReadData method. In this case, the attribute and **SecurityAction.InheritanceDemand** are applied at the method level instead of at the class level

VB

Public Class MyClass

Public Sub New()

End Sub

<CustomPermissionAttribute(SecurityAction.InheritanceDemand)> Public Overridable Function

ReadData() As String

'Access a custom resource.

End Function

End Class

[C#]

public class MyClass

{

public MyClass()

{

}

[CustomPermissionAttribute(SecurityAction.InheritanceDemand)]

public virtual string ReadData()

{

//Access a custom resource.

}

}

See Also

[Extending Metadata Using Attributes](https://msdn.microsoft.com/en-us/library/5x6cd29c(v=vs.71).aspx) | [Security Demands](https://msdn.microsoft.com/en-us/library/60zfc754(v=vs.71).aspx) | [Creating Your Own Code Access Permissions](https://msdn.microsoft.com/en-us/library/yctbsyf4(v=vs.71).aspx) | [Adding Declarative Security Support](https://msdn.microsoft.com/en-us/library/84kh7ht8(v=vs.71).aspx)| [Code Access Security](https://msdn.microsoft.com/en-us/library/930b76w0(v=vs.71).aspx)

##### Inheritance Demands

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4**

Inheritance demands applied to classes have a different meaning than inheritance demands applied to methods. You can place inheritance demands at the class level to ensure that only code with the specified permission can inherit from your class. Inheritance demands placed on methods require that code have the specified permission to override the method.

|  |
| --- |
| **NoteNote** |
| A new transparency model has been introduced in the .NET Framework version 4. The [Security-Transparent Code, Level 2](https://msdn.microsoft.com/en-us/library/dd233102(v=vs.100).aspx) model identifies secure code with the [SecurityCriticalAttribute](https://msdn.microsoft.com/en-us/library/system.security.securitycriticalattribute(v=vs.100).aspx) or the [SecuritySafeCriticalAttribute](https://msdn.microsoft.com/en-us/library/system.security.securitysafecriticalattribute(v=vs.100).aspx) attribute. Security-critical code requires both callers and inheritors to be fully trusted. Any type or member that inherits from a security-critical type or member must be security-critical or security-safe-critical. Assemblies that use earlier code access security rules (level 1) can call level 2 security-critical types and members if they are full trusted. However, the level 2 types and members must specifically identify inheritance demands, because level 2 implicit inheritance rules do not apply to level 1 callers. |

[**Class Inheritance Demands**](javascript:void(0))

An inherited demand applied to a class has the effect of demanding that all classes derived from the parent class have the specified permission. For example, if class B is to inherit from class A and class A is protected by an inheritance demand, then B must be granted that permission in order to run. If class B is granted that permission and derives from class A, then class C must also have the permission demanded by A, if it is to derive from B. This demand can be applied only declaratively.

The following code example uses an inheritance demand to require that any class that inherits from the MyClass1 class must have the custom permission CustomPermissionAttribute. This permission is a hypothetical custom permission and does not exist in the .NET Framework. The demand is made by passing the CustomPermissionAttribute a [SecurityAction.InheritanceDemand](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx) enumeration value.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/x4yx82e6(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

[CustomPermissionAttribute(SecurityAction.InheritanceDemand)]

public class MyClass

{

public MyClass()

{

}

public virtual string ReadData()

{

// Access a custom resource.

}

}

[**Method Inheritance Demands**](javascript:void(0))

Placing an inheritance demand on a static method in the base class has no impact on derived classes because the static methods are unrelated. However, placing an inheritance demand on any nonstatic method in the base class has the same effect as an inheritance demand on the class. All methods in the derived class, including the constructor for the class, must meet the inheritance demand.

[**See Also**](javascript:void(0))

Concepts

[Security Demands](https://msdn.microsoft.com/en-us/library/60zfc754(v=vs.100).aspx)

[Creating Your Own Code Access Permissions](https://msdn.microsoft.com/en-us/library/yctbsyf4(v=vs.100).aspx)

[Adding Declarative Security Support](https://msdn.microsoft.com/en-us/library/84kh7ht8(v=vs.100).aspx)

Other Resources

[Extending Metadata Using Attributes](https://msdn.microsoft.com/en-us/library/5x6cd29c(v=vs.100).aspx)

[Code Access Security](https://msdn.microsoft.com/en-us/library/c5tk9z76(v=vs.100).aspx)

#### Overriding Security Checks

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4, 4.5, 4.6**

Normally, a security check examines every caller in the call stack to ensure that each caller has been granted the specified permission. However, you can override the outcome of security checks by calling [Assert](https://msdn.microsoft.com/en-us/library/c82hh6x8(v=vs.71).aspx), [Deny](https://msdn.microsoft.com/en-us/library/0ebw73h9(v=vs.71).aspx), or [PermitOnly](https://msdn.microsoft.com/en-us/library/a3z81ks0(v=vs.71).aspx) on an individual permission object or a permission set object. Depending on which of these methods you call, you can cause the security check to succeed or fail, even though the permissions of all callers on the stack might not have been checked.

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

* If, during the stack walk, the runtime discovers more than one override of the same type (that is, two calls to **Assert**) in one stack frame, the second override causes an exception to be thrown.
* When different overrides are present in the same stack frame, the runtime processes these overrides in the following order:**PermitOnly**, then **Deny**, and finally **Assert**.

To replace an override, first call the appropriate revert method (for example, [RevertAssert](https://msdn.microsoft.com/en-us/library/fw7655b1(v=vs.71).aspx)) and then apply the new override.

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

Application developers do not usually need to use **Assert**, **Deny**, or **PermitOnly**, and component and class library developers rarely need to use them. However, security overrides are appropriate in some situations, which are described in the [Assert](https://msdn.microsoft.com/en-us/library/91wteedy(v=vs.71).aspx), [Deny](https://msdn.microsoft.com/en-us/library/hk3b9142(v=vs.71).aspx), and [PermitOnly](https://msdn.microsoft.com/en-us/library/y6abcbh4(v=vs.71).aspx) topics.

**Note**   If you perform an override (**Deny**, **Assert**, or **PermitOnly**), you must revert the permission before you can perform the same kind of override in the same stack frame (that is, method). Otherwise, a [SecurityException](https://msdn.microsoft.com/en-us/library/system.security.securityexception(v=vs.71).aspx) is thrown. For example, if you deny a permission, P, you must revert that permission before you can deny another permission, Q, in the same method.

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

|  |  |
| --- | --- |
| **Method** | **Method Action** |
| [CodeAccessPermission.RevertAll](https://msdn.microsoft.com/en-us/library/2t7w266z(v=vs.71).aspx) | Causes all previous overrides for the current frame to be removed and no longer in effect. |
| [CodeAccessPermission.RevertAssert](https://msdn.microsoft.com/en-us/library/fw7655b1(v=vs.71).aspx) | Causes any previous **Assert** for the current frame to be removed and no longer in effect. |
| [CodeAccessPermission.RevertDeny](https://msdn.microsoft.com/en-us/library/3exz858d(v=vs.71).aspx) | Causes any previous **Deny** for the current frame to be removed and no longer in effect. |
| [CodeAccessPermission.RevertPermitOnly](https://msdn.microsoft.com/en-us/library/s5dtth18(v=vs.71).aspx) | Causes any previous **PermitOnly** for the current frame to be removed and no longer in effect. |

See Also

[Code Access Security](https://msdn.microsoft.com/en-us/library/930b76w0(v=vs.71).aspx) | [Writing Secure Class Libraries](https://msdn.microsoft.com/en-us/library/e942ksxt(v=vs.71).aspx) | [Assert Method](https://msdn.microsoft.com/en-us/library/c82hh6x8(v=vs.71).aspx) | [Deny Method](https://msdn.microsoft.com/en-us/library/0ebw73h9(v=vs.71).aspx) | [PermitOnly Method](https://msdn.microsoft.com/en-us/library/a3z81ks0(v=vs.71).aspx)

###### Assert

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4, 4.5, 4.6**

[Assert](https://msdn.microsoft.com/en-us/library/c82hh6x8(v=vs.71).aspx) is a method that can be called on code access permission classes and on the [PermissionSet](https://msdn.microsoft.com/en-us/library/system.security.permissionset(v=vs.71).aspx) class. You can use **Assert** to enable your code (and downstream callers) to perform actions that your code has permission to do but its callers might not have permission to do. A security assertion changes the normal process that the runtime performs during a security check. When you assert a permission, it tells the security system not to check the callers of your code for the asserted permission.

**Caution**   Use assertions carefully because they can open security holes and undermine the runtime's mechanism for enforcing security restrictions.

Assertions are useful in situations in which a library calls into unmanaged code or makes a call that requires a permission that is not obviously related to the library's intended use. For example, all managed code that calls into unmanaged code must have **SecurityPermission**with the**UnmanagedCode**flag specified. Code that does not originate from the local computer, such as code that is downloaded from the local intranet, will not be granted this permission by default. Therefore, in order for code that is downloaded from the local intranet to be able to call a library that uses unmanaged code, it must have the permission asserted by the library. Additionally, some libraries might make calls that are unseen to callers and require special permissions.

You can also use assertions in situations in which your code accesses a resource in a way that is completely hidden from callers. For example, suppose your library acquires information from a database but in the process also reads information from the computer registry. Because developers using your library do not have access to your source, they have no way of knowing that their code requires **RegistryPermission**in order to use your code. In this case, if you decide that it is not reasonable or necessary to require that callers of your code have permission to access the registry, you can assert permission for reading the registry. In this situation, it is appropriate for the library to assert the permission so that callers without **RegistryPermission**can use the library.

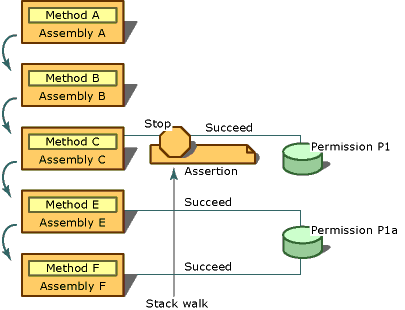
The assertion affects the stack walk only if the asserted permission and a permission demanded by a downstream caller are of the same type and if the demanded permission is a subset of the asserted permission. For example, if you assert **FileIOPermission** to read all files on the C drive, and a downstream demand is made for **FileIOPermission** to read files in C:\Temp, the assertion could affect the stack walk; however, if the demand was for **FileIOPermission** to write to the C drive, the assertion would have no effect.

To perform assertions, your code must be granted both the permission you are asserting and the [SecurityPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermission(v=vs.71).aspx) that represents the right to make assertions. Although you could assert a permission that your code has not been granted, the assertion would be pointless because the security check would fail before the assertion could cause it to succeed.

The following illustration shows what happens when you use **Assert**. Assume that the following statements are true about assemblies A, B, C, E, and F, and two permissions, P1 and P1A:

* P1A represents the right to read .txt files on the C drive.
* P1 represents the right to read all files on the C drive.
* P1A and P1 are both **FileIOPermission**types, and P1A is a subset of P1.
* Assemblies E and F have been granted P1A permission.
* Assembly C has been granted P1 permission.
* Assemblies A and B have been granted neither P1 nor P1A permissions.
* Method A is contained in assembly A, method B is contained in assembly B, and so on.

**Using Assert**



In this scenario, method A calls B, B calls C, C calls E, and E calls F. Method C asserts permission to read files on the C drive (permission P1), and method E demands permission to read .txt files on the C drive (permission P1A). When the demand in F is encountered at run time, a stack walk is performed to check the permissions of all callers of F, starting with E. E has been granted P1A permission, so the stack walk proceeds to examine the permissions of C, where C's assertion is discovered. Because the demanded permission (P1A) is a subset of the asserted permission (P1), the stack walk stops and the security check automatically succeeds. It does not matter that assemblies A and B have not been granted permission P1A. By asserting P1, method C ensures that its callers can access the resource protected by P1, even if the callers have not been granted permission to access that resource.

If you design a class library and a class accesses a protected resource, you should, in most cases, make a [security demand](https://msdn.microsoft.com/en-us/library/60zfc754(v=vs.71).aspx) requiring that the callers of the class have the appropriate permission. If the class then performs an operation for which you know most of its callers will not have permission, and if you are willing to take the responsibility for letting these callers call your code, you can assert the permission by calling the **Assert** method on a permission object that represents the operation the code is performing. Using **Assert** in this way lets callers that normally could not do so call your code. Therefore, if you assert a permission, you should be sure to perform appropriate security checks beforehand to prevent your component from being misused.

For example, suppose your highly trusted library class has a method that deletes files. It accesses the file by calling an unmanaged Win32 function. A caller invokes your code's **Delete** method, passing in the name of the file to be deleted, C:\Test.txt. Within the **Delete** method, your code creates a [FileIOPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.fileiopermission(v=vs.71).aspx) object representing write access to C:\Test.txt. (Write access is required to delete a file.) Your code then invokes an imperative security check by calling the **FileIOPermission** object's **Demand** method. If one of the callers in the call stack does not have this permission, a [SecurityException](https://msdn.microsoft.com/en-us/library/system.security.securityexception(v=vs.71).aspx) is thrown. If no exception is thrown, you know that all callers have the right to access C:\Test.txt. Because you believe that most of your callers will not have permission to access unmanaged code, your code then creates a[SecurityPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermission(v=vs.71).aspx)object that represents the right to call unmanaged code and calls the object's **Assert** method. Finally, it calls the unmanaged Win32 function to delete C:\Text.txt and returns control to the caller.

**Caution**   You must be sure that your code does not use assertions in situations where your code can be used by other code to access a resource that is protected by the permission you are asserting. For example, in code that writes to a file whose name is specified by the caller as a parameter, you would not assert the **FileIOPermission** for writing to files because your code would be open to misuse by a third party.

If you call the **Assert** method on multiple permissions in the same method, unpredictable and undesired behaviors might result. Instead, you should create a **PermissionSet** object, pass it the individual permissions you want to invoke, and then call the **Assert** method on the**PermissionSet** object.

The following example shows declarative syntax for overriding security checks using the **Assert** method. Notice that the**FileIOPermissionAttribute** syntax takes two values: a [SecurityAction](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.71).aspx) enumeration and the location of the file or directory to which permission is to be granted. The call to **Assert** causes demands for access to C:\Log.txt to succeed, even though callers are not checked for permission to access the file.

VB

Option Explicit

Option Strict

Imports System

Imports System.IO

Imports System.Security.Permissions

Namespace LogUtil

Public Class Log

Public Sub New()

End Sub

<FileIOPermission(SecurityAction.Assert, All := "C:\Log.txt")> Public Sub

MakeLog()

Dim TextStream As New StreamWriter("C:\Log.txt")

TextStream.WriteLine("This Log was created on {0}", DateTime.Now) '

TextStream.Close()

End Sub

End Class

End Namespace

[C#]

namespace LogUtil

{

using System;

using System.IO;

using System.Security.Permissions;

public class Log

{

public Log()

{

}

[FileIOPermission(SecurityAction.Assert, All = @"C:\Log.txt")]

public void MakeLog()

{

StreamWriter TextStream = new StreamWriter(@"C:\Log.txt");

TextStream.WriteLine("This Log was created on {0}", DateTime.Now);

TextStream.Close();

}

}

}

The following code fragments show imperative syntax for overriding security checks using the **Assert** method. In this example, an instance of the **FileIOPermission** object is declared. Its constructor is passed **FileIOPermissionAccess.Read** to define the type of access allowed, followed by a string describing the file's location. Once the **FileIOPermission** object is defined, you only need to call its **Assert** method to override the security check.

VB

Option Explicit

Option Strict

Imports System

Imports System.IO

Imports System.Security.Permissions

Namespace LogUtil

Public Class Log

Public Sub New()

End Sub 'New

Public Sub MakeLog()

Dim FilePermission As New FileIOPermission(FileIOPermissionAccess.AllAccess, "C:\Log.txt")

FilePermission.Assert()

Dim TextStream As New StreamWriter("C:\Log.txt")

TextStream.WriteLine("This Log was created on {0}", DateTime.Now)

TextStream.Close()

End Sub

End Class

End Namespace

[C#]

namespace LogUtil

{

using System;

using System.IO;

using System.Security.Permissions;

public class Log

{

public Log()

{

}

public void MakeLog()

{

FileIOPermission FilePermission = new FileIOPermission(FileIOPermissionAccess.AllAccess,@"C:\Log.txt");

FilePermission.Assert();

StreamWriter TextStream = new StreamWriter(@"C:\Log.txt");

TextStream.WriteLine("This Log was created on {0}", DateTime.Now);

TextStream.Close();

}

}

}

See Also

[Extending Metadata Using Attributes](https://msdn.microsoft.com/en-us/library/5x6cd29c(v=vs.71).aspx) | [Overriding Security Checks](https://msdn.microsoft.com/en-us/library/c2f7a0y2(v=vs.71).aspx) | [Code Access Security](https://msdn.microsoft.com/en-us/library/930b76w0(v=vs.71).aspx) | [Assert Method](https://msdn.microsoft.com/en-us/library/c82hh6x8(v=vs.71).aspx) | [PermissionSet Class](https://msdn.microsoft.com/en-us/library/system.security.permissionset(v=vs.71).aspx) |[SecurityPermission Class](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermission(v=vs.71).aspx) | [Make a Security Demand](https://msdn.microsoft.com/en-us/library/60zfc754(v=vs.71).aspx) | [FileIOPermission Class](https://msdn.microsoft.com/en-us/library/system.security.permissions.fileiopermission(v=vs.71).aspx) | [SecurityAction Enumeration](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.71).aspx)

###### Deny

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4**

|  |
| --- |
| **Important noteImportant** |
| In the .NET Framework version 4, runtime support has been removed for enforcing the [Deny](https://msdn.microsoft.com/en-US/library/system.security.permissions.securityaction(v=vs.100).aspx), [RequestMinimum](https://msdn.microsoft.com/en-US/library/system.security.permissions.securityaction(v=vs.100).aspx), [RequestOptional](https://msdn.microsoft.com/en-US/library/system.security.permissions.securityaction(v=vs.100).aspx), and[RequestRefuse](https://msdn.microsoft.com/en-US/library/system.security.permissions.securityaction(v=vs.100).aspx) permission requests. These requests should not be used in code that is based on .NET Framework 4 or later. For more information about this and other changes, see [Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-US/library/dd233103(v=vs.100).aspx). |

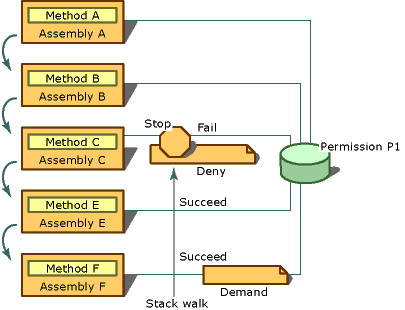
Calling [Deny](https://msdn.microsoft.com/en-US/library/0ebw73h9(v=vs.71).aspx) prevents access to the resource specified by the denied permission. If your code calls **Deny** and a downstream caller subsequently demands the denied permission, the security check will fail, even if all callers have permission to access that resource. The permission being demanded and the permission being denied do not have to match exactly for the **Deny** to take effect, and the demanded permission does not have to be a subset of the denied permission. However, if the intersection of the two permissions is empty (that is, if they have nothing in common), the call to **Deny** will have no effect. Note that **Deny** cannot override deeper code on the call stack that performs an[Assert](https://msdn.microsoft.com/en-US/library/91wteedy(v=vs.71).aspx). If deeper code on the call stack performs an **Assert**, the deeper code can access the resource that code higher on the call stack denies.

You can use calls to **Deny** in your code to protect yourself from liability because **Deny** makes it impossible for your code to be used to access the denied resource. However, the call to **Deny** does not block future security assertions by downstream callers.

The following illustration shows what happens when you use **Deny**. Assume the following statements are true about assemblies A, B, C, D, and E, and permission P1:

* P1 represents the right to read all files on the C drive.
* Assemblies A, B, C, D, and E have been granted P1.
* Method F places a demand on Permission P1.
* Method C creates an instance of the P1 class and then calls P1's **Deny** method.
* Method A is contained in assembly A, method B is contained in assembly B, and so on.

**Using Deny**



Method C's call to **Deny** can affect the outcome of demands for P1. For example, suppose that method A calls B, B calls C, C calls E, and E calls F. Because method F directly accesses the resource that P1 protects, method F invokes a security check for P1 by calling P1's **Demand** method (or by using a declarative demand). This demand causes the runtime to check the permissions of all callers in the call stack, starting with assembly E. Because assembly E has been granted P1 permission, the runtime proceeds to examine the permissions of assembly C. But because method C has denied P1, the security check invoked by method E fails at that point, and a [SecurityException](https://msdn.microsoft.com/en-US/library/system.security.securityexception(v=vs.71).aspx) is thrown. It does not matter whether assembly C and its callers (assemblies A and B) have been granted P1; the security check still fails. Because method C called**Deny**, code in assemblies A and B cannot access the resource protected by P1.

The following code shows declarative syntax for overriding security checks using the **Deny** method. In this example, the**ReflectionPermission** syntax specifies two values: a [SecurityAction](https://msdn.microsoft.com/en-US/library/system.security.permissions.securityaction(v=vs.71).aspx) enumeration and the setting for the **TypeInformation** property.**TypeInformation** is set to true to specify that this permission represents the right to view private members through reflection and**SecurityAction.Deny** is passed to deny that permission. See the description of [ReflectionPermission](https://msdn.microsoft.com/en-US/library/system.security.permissions.reflectionpermission(v=vs.71).aspx) for a complete list of values you can specify. With this security declaration, the method cannot read private members of a type through reflection.

VB

Option Strict

Option Explicit

Imports System

Imports System.Security.Permissions

<ReflectionPermissionAttribute(SecurityAction.Deny, TypeInformation = true ")> Public Class

MyClass1

Public Sub New()

End Sub

Public Sub GetPublicMembers ()

' Access public members through reflection.

End Sub

End Class

[C#]

using System;

using System.Security.Permissions;

[ReflectionPermissionAttribute(SecurityAction.Deny, TypeInformation = true)]

public class MyClass

{

public MyClass()

{

}

public void GetPublicMembers()

{

//Access public members through reflection.

}

}

The following code shows imperative syntax for overriding security checks using the **Deny** method. In this example, the **ReflectionPermission**object is declared and its constructor is passed **ReflectionPermissionFlag.TypeInformation** to initialize the current permission. When the**Deny** method is called, the code and callers can never be used to read private fields through reflection.

VB

Option Explicit

Option Strict

Imports System

Imports System.Security.Permissions

Public Class MyClass1

Public Sub New()

End Sub

Public Sub ReadRegistry()

Dim MyPermission As New ReflectionPermission (ReflectionPermissionFlag.TypeInformation)

MyPermission.Deny()

' Access public members through reflection.

End Sub

End Class

[C#]

using System;

using System.Security.Permissions;

public class MyClass {

public MyClass() {

}

public void ReadRegistry() {

ReflectionPermission MyPermission = new ReflectionPermission (ReflectionPermissionFlag.TypeInformation);

MyPermission.Deny();

// Access public members through reflection.

}

}

**Canonicalization Problems Using Deny**

You should be extremely careful when denying [FileIOPermission](https://msdn.microsoft.com/en-US/library/system.security.permissions.fileiopermission(v=vs.71).aspx), [RegistryPermission](https://msdn.microsoft.com/en-US/library/system.security.permissions.registrypermission(v=vs.71).aspx), [WebPermission](https://msdn.microsoft.com/en-US/library/system.net.webpermission(v=vs.71).aspx), [UrlIdentityPermission](https://msdn.microsoft.com/en-US/library/system.security.permissions.urlidentitypermission(v=vs.71).aspx),[SiteIdentityPermission](https://msdn.microsoft.com/en-US/library/system.security.permissions.siteidentitypermission(v=vs.71).aspx), and [EnvironmentPermission](https://msdn.microsoft.com/en-US/library/system.security.permissions.environmentpermission(v=vs.71).aspx) because single files, registry entries, URLs, and system paths can be described using multiple names. For example, a single file, MyFile.log, can be referenced a number of ways, including "c:\MyFile.log" and "\\MyMachineName\c$\MyFile.log". If you create a permission that represents access to "c:\MyFile.txt" and then deny that permission to your code, your code might still be able to access the file using the alternate path "\\MyMachineName\c$\MyFile.log".

You can use a combination of [PermitOnly](https://msdn.microsoft.com/en-US/library/y6abcbh4(v=vs.71).aspx) and **Deny** to avoid canonization problems. **PermitOnly** gives you the ability to specify only one of several possible names for a resource and has the side effect of denying access to that resource using any other name. After you use**PermitOnly** to specify the one allowed name for a resource, you should use **Deny** to disallow access to the resource using that name.

The following code uses a combination of **Deny** and **PermitOnly** to prevent your code from accessing a resource called MyLog.log. This code also blocks access to the resource using all alternative names or paths.

VB

<FileIOPermissionAttribute(SecurityAction.PermitOnly, All := "C:\ "), FileIOPermissionAttribute(SecurityAction.Deny, All := "C:\MyLog.log")>

[C#]

[FileIOPermissionAttribute(SecurityAction.PermitOnly, All = @"C:\ ")]

[FileIOPermissionAttribute(SecurityAction.Deny, All = @"C:\MyLog.log")]

See Also

[Extending Metadata Using Attributes](https://msdn.microsoft.com/en-US/library/5x6cd29c(v=vs.71).aspx) | [Overriding Security Checks](https://msdn.microsoft.com/en-US/library/c2f7a0y2(v=vs.71).aspx) | [Code Access Security](https://msdn.microsoft.com/en-US/library/930b76w0(v=vs.71).aspx) | [SecurityAction Enumeration](https://msdn.microsoft.com/en-US/library/system.security.permissions.securityaction(v=vs.71).aspx) |[RegistryPermissionAttribute Class](https://msdn.microsoft.com/en-US/library/system.security.permissions.registrypermissionattribute(v=vs.71).aspx)

###### PermitOnly

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4.0**

Calling [PermitOnly](https://msdn.microsoft.com/en-us/library/a3z81ks0(v=vs.71).aspx) has essentially the same effect as calling [Deny](https://msdn.microsoft.com/en-us/library/0ebw73h9(v=vs.71).aspx) but is a different way of specifying the conditions under which the security check should fail. Instead of saying that a specified resource cannot be accessed, as **Deny** does, **PermitOnly** says that only the resources you specify can be accessed. Therefore, calling **PermitOnly** on permission X is the same as calling **Deny** on all permissions except permission X. If you call **PermitOnly**, your code can be used to access only the resources protected by the permissions that you specify when you call**PermitOnly**. You use **PermitOnly** instead of **Deny** when it is more convenient to describe resources that can be accessed instead of resources that cannot be accessed.

If your code calls **PermitOnly** on a permission P1, and downstream callers then demand permission P2, the **PermitOnly** call affects the result of the stack walk only if P1 and P2 are of different types and if P2 is not a subset of P1.

The following code fragments show declarative syntax for overriding security checks using **PermitOnly**. Callers cannot use this code to access any protected resources except user interface resources. .

VB

Option Explicit

Option Strict

Imports System

Imports System.Security.Permissions

Public Class MyClass1

Public Sub New()

End Sub

<UIPermissionAttribute(SecurityAction.PermitOnly, Unrestricted := True)> Public Sub

ReadRegistry()

'Access a UI resource.

End Sub

End Class

[C#]

using System;

using System.Security.Permissions;

public class MyClass

{

public MyClass() {

}

[UIPermissionAttribute(SecurityAction.PermitOnly, Unrestricted=true)]

public void ReadRegistry() {

//Access a UI resource.

}

}

The following code example shows imperative syntax for overriding security checks using the **PermitOnly** method. The **UIPermission**constructor is passed a [PermissionState](https://msdn.microsoft.com/en-us/library/system.security.permissions.permissionstate(v=vs.71).aspx) object that specifies the user interface resources to which access is to be granted. Once the**PermitOnly** method is called, the code and all callers can be used only to access user interface resources.

VB

Option Explicit

Option Strict

Imports System

Imports System.Security.Permissions

Public Class MyClass1

Public Sub New()

End Sub

Public Sub ReadRegistry()

Dim MyPermission As New UIPermission(PermissionState.Unrestricted)

MyPermission.PermitOnly()

'Access a UI resource.

End Sub

End Class

[C#]

using System;

using System.Security.Permissions;

public class MyClass {

public MyClass() {

}

public void ReadRegistry() {

UIPermission MyPermission = new UIPermission(PermissionState.Unrestricted);

MyPermission.PermitOnly();

//Access a UI resource.

}

}

#### Declarative Security Used with Class and Member Scope

**.NET Framework 1.1**

Declarative security can be performed on classes, members, and nested classes. This section outlines the rules used to evaluate declarative security when applied to multiple levels of the same class.

**Classes, Members, and Declarative Security**

When using declarative security, method-level declarations override class-level declarations for demands and asserts. However, when you use[Inheritance Demands](https://msdn.microsoft.com/en-us/library/x4yx82e6(v=vs.71).aspx) and [Link Demands](https://msdn.microsoft.com/en-us/library/hzsc022c(v=vs.71).aspx), the following rules apply:

* **Inheritance Demands.**When applied at the class level, an inheritance demand requires the specified permission in order to derive from the class. When applied at the method level, an inheritance demand requires the specified permission in order to override the method in a derived class. Because inheritance demands have different meanings for classes and methods, declarations can be applied to both the member and class levels independently.
* **Link Demands.** When applied to both the class and member levels, link demands are evaluated sequentially. The end result is a union of both the permissions demanded.

For all other actions, the method level completely overrides and replaces the class level. If the security actions are different (for example, a demand at class level with an assert at method level), there is no interaction whatsoever and both are evaluated.

**Nested Classes and Declarative Security**

When you apply declarative security to classes, it does not propagate to any nested classes or methods of nested classes. Conversely, when you apply declarative security to nested classes or methods of a nested class, it does not propagate to the parent classes either. You should apply declarative security to nested classes as if they were separate classes.

The following example shows a hypothetical permission demanded on the class level of a class called Main. Within that class, a nested class called Nested is defined. In this example, the demand does not apply to the nested class.

VB

<SomePermissionAttribute(SecurityAction.Demand, Unrestricted:=True)> Public Class Main

' This nested class is not influenced by the demand.

Public Class Nested

' This method is not influenced by the demand.

Public Sub MyMethod()

End Sub

End Class

End Class

[C#]

[SomePermissionAttribute(SecurityAction.Demand, Unrestricted = true)]

class Main

{

// This nested class is not influenced by the demand.

class Nested

{

// This method is not influenced by the demand.

public void MyMethod()

{

}

}

}

#### Declarative Security Used with Class and Member Scope

**.NET Framework 2.0, 3.0**

Declarative security can be performed on classes, members, and nested classes. This section outlines the rules used to evaluate declarative security when applied to multiple levels of the same class.

Classes, Members, and Declarative Security

When there is declarative security for the same security action on both the class level and the method level, the declarative security will be applied according to the following table.

|  |  |  |
| --- | --- | --- |
| **Security action** | **.NET Framework version 1.0 and 1.1 behavior** | **NET Framework version 2.0 behavior** |
| Demand | Method-level attributes override class-level attributes. (If a declarative demand is placed on the method level, a class-level declarative demand will be ignored.) | Method-level attributes and class-level attributes are unioned together into a single permission set for both levels. |
| Link demand | Method-level attributes and class-level attributes are unioned. | No change in behavior. |
| Inheritance demand | Class-level attributes require the specified permission in order to derive from the class.  Method-level attributes require the specified permission in order to override the method in a derived class.  Because inheritance demands have different meanings for classes and methods, declarations can be applied to both the class and method levels independently. | No change in behavior. |
| Assert | Method-level attributes override class-level attributes. | Method-level attributes and class-level attributes are unioned together into a single permission set for both levels. |
| Deny | Method-level attributes override class-level attributes. | Method-level attributes and class-level attributes are unioned together into a single permission set for both levels. |
| Permit only | Method-level attributes override class-level attributes. | Method-level attributes and class-level attributes are intersected together into a single permission set for both levels. |

If the security actions are different (for example, a demand at class level with an assert at method level), there is no interaction whatsoever and both are evaluated.

Nested Classes and Declarative Security

When you apply declarative security to classes, it does not propagate to any nested classes or methods of nested classes. Conversely, when you apply declarative security to nested classes or methods of a nested class, it does not propagate to the parent classes either. You should apply declarative security to nested classes as if they were separate classes.

The following example shows a hypothetical permission demanded on the class level of a class called Main. Within that class, a nested class called Nested is defined. In this example, the demand does not apply to the nested class.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/cz02ke7h(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

[SomePermissionAttribute(SecurityAction.Demand, Unrestricted = true)]

class Main

{

// This nested class is not influenced by the demand.

class Nested

{

// This method is not influenced by the demand.

public void MyMethod()

{

}

}

}

See Also

Concepts

[Security Demands](https://msdn.microsoft.com/en-us/library/60zfc754(v=vs.80).aspx)

Other Resources

[Code Access Security](https://msdn.microsoft.com/en-us/library/930b76w0(v=vs.80).aspx)

#### Declarative Security Used with Class and Member Scope

**.NET Framework 4**

[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635736413806036745

Declarative security can be performed on classes, members, and nested classes. This section outlines the rules used to evaluate declarative security when applied to multiple levels of the same class.

|  |
| --- |
| **Important noteImportant** |
| In the .NET Framework version 4, runtime support has been removed for enforcing the [Deny](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx), [RequestMinimum](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx), [RequestOptional](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx), and[RequestRefuse](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.100).aspx) permission requests. These requests should not be used in code that is based on .NET Framework 4 or later. For more information about this and other changes, see [Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx). |

[Classes, Members, and Declarative Security](javascript:void(0))

When there is declarative security for the same security action on both the class level and the method level, the declarative security will be applied according to the following table.

|  |  |  |
| --- | --- | --- |
| **Security action** | **NET Framework version 2.0 behavior** | **.NET Framework version 4 behavior** |
| Demand | Method-level attributes and class-level attributes are unioned together into a single permission set for both levels. | No change in behavior. |
| Link demand | Method-level attributes and class-level attributes are unioned. | No change in behavior. |
| Inheritance demand | Class-level attributes require the specified permission in order to derive from the class.  Method-level attributes require the specified permission in order to override the method in a derived class.  Because inheritance demands have different meanings for classes and methods, declarations can be applied to both the class and method levels independently. | No change in behavior. |
| Assert | Method-level attributes and class-level attributes are unioned together into a single permission set for both levels. | No change in behavior. |
| Deny | Method-level attributes and class-level attributes are unioned together into a single permission set for both levels. | Obsolete in the .NET Framework 4. |
| Permit only | Method-level attributes and class-level attributes are intersected together into a single permission set for both levels. | No change in behavior. |

If the security actions are different (for example, a demand at class level with an assert at method level), there is no interaction whatsoever and both are evaluated.

[**Nested Classes and Declarative Security**](javascript:void(0))

When you apply declarative security to classes, it does not propagate to any nested classes or methods of nested classes. Conversely, when you apply declarative security to nested classes or methods of a nested class, it does not propagate to the parent classes either. You should apply declarative security to nested classes as if they were separate classes.

The following example shows a hypothetical permission demanded on the class level of a class called Main. Within that class, a nested class called Nested is defined. In this example, the demand does not apply to the nested class.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/cz02ke7h(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

[SomePermissionAttribute(SecurityAction.Demand, Unrestricted = true)]

class Main

{

// This nested class is not influenced by the demand.

class Nested

{

// This method is not influenced by the demand.

public void MyMethod()

{

}

}

}

#### Security Optimizations

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4.0**

Security checks can cause performance problems for some applications. There are two optimization techniques that you can use to improve performance. One technique combines security demands; the other suppresses demands for permission to call into unmanaged code. Although these techniques might improve the performance of your application, they can also make your application vulnerable to security exploits. Before using these optimization techniques, you should take the following precautions:

* Follow the [Secure Coding Guidelines](https://msdn.microsoft.com/en-us/library/d55zzx87(v=vs.90).aspx) for managed code.
* Understand the security implications of the optimizations and use other methods to protect your application as appropriate.
* Implement the minimum security optimizations required to improve application performance.

After optimizing your code, you should test the optimized code to determine whether its performance actually improved. If not, you should remove the security optimizations to help prevent inadvertent security weaknesses.

**Caution**Security optimization requires you to change the standard code access security. To avoid introducing security vulnerabilities into your code, be sure you understand the security implications of the optimization techniques before using them.

**Combining Security Demands**

To optimize code that makes security demands, you can, in some situations, use a technique for combining demands.

For example, if:

* your code performs a number of operations within a single method, and
* when performing each of those operations, your code calls into a managed class library that demands that your code have the same permission on each call to the library,

then:

* you can modify your code to perform a **Demand** and an **Assert** of that permission to reduce the overhead incurred by the security demands.

If the call stack depth above the method is large, using this technique can result in a significant performance gain.

To illustrate how this works, suppose method M performs 100 operations. Each operation calls into a library that makes a security demand requiring your code and all its callers to have the X permission. Because of the security demands, each operation causes the runtime to walk the entire call stack examining each caller's permissions, to determine whether X permission was actually granted to each caller. If the call stack above method M isnlevels deep, 100n comparisons are required.

To optimize, you can do the following in method M:

* Demand X, which results in the runtime performing a stack walk (of depth n) to ensure that all callers indeed have permission X.
* Then, assert permission X, which causes subsequent stack walks to stop at method M and succeed, thereby reducing the number of permission comparisons by 99n.

In the following code example, the GetFileCreationTime method takes a string representation of a directory as a parameter and displays the name and creation date of every file in that directory. The static [File.GetCreationTime](https://msdn.microsoft.com/en-us/library/66x02450(v=vs.71).aspx) method reads information from the files but requires a demand and stack walk for every file it reads. The method creates a new instance of the **FileIOPermission** object, performs a demand to check the permissions of all callers on the stack, and then asserts the permission if the demand is successful. If the demand succeeds, only one stack walk is performed and the method reads the creation time from every file in the passed directory.

VB

Option Explicit

Option Strict

Imports System

Imports System.IO

Imports System.Security

Imports System.Security.Permissions

Namespace OptimizedSecurity

Public Class FileUtil

Public Sub New()

End Sub

Public Sub GetFileCreationTime(Directory As String)

'Initialize DirectoryInfo object to the passed directory.

Dim DirFiles As New DirectoryInfo(Directory)

'Create a DateTime object to be initialized below.

Dim TheTime As DateTime

'Get a list of files for the current directory.

Dim Files As FileInfo() = DirFiles.GetFiles()

'Create a new instance of FileIOPermission with read

'permission to the current directory.

Dim FilePermission As New FileIOPermission(FileIOPermissionAccess.Read, Directory)

Try

'Check the stack by making a demand.

FilePermission.Demand()

'If the demand succeeded, assert permission and

'perform the operation.

FilePermission.Assert()

Dim x As Integer

For x = 0 To Files.Length - 1

TheTime = File.GetCreationTime(Files(x).FullName)

Console.WriteLine("File: {0} Created: {1:G}", Files(x).Name, TheTime)

Next x

'Catch a security exception and display an error.

Catch

Console.WriteLine("You do not have permission to read this directory.")

End Try

End Sub

End Class

End Namespace

If the demand in the previous example succeeds, every file and its creation date and time are displayed for the passed directory. If the demand fails, the security exception is intercepted and the following message is displayed to the console:

You do not have permission to read this directory.

**Suppressing Demands for Unmanaged Code Permission**

A special optimization is available to code that has permission to call unmanaged code. This optimization enables your managed code to call into unmanaged code without incurring the overhead of a stack walk. Asserting unmanaged code permission can shorten the stack walk, but the optimization described in this topic can eliminate it entirely. (See [SecurityPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermission(v=vs.71).aspx) for more information about the permission to call into unmanaged code.)

Normally, a call into unmanaged code raises a demand of unmanaged code permission, which results in a stack walk that determines whether all callers have permission to call into unmanaged code. Applying the custom attribute [SuppressUnmanagedCodeSecurityAttribute](https://msdn.microsoft.com/en-us/library/system.security.suppressunmanagedcodesecurityattribute(v=vs.71).aspx) to the method that calls into unmanaged code suppresses the demand. This attribute replaces the full stack walk at run time with a check that only verifies the permissions of the immediate caller at link time. In effect, using this attribute creates an open door into unmanaged code. Only code that has unmanaged code permission can use this attribute; otherwise, it has no effect.

**Caution**   Use the **SuppressUnmanagedCodeSecurityAttribute** attribute only with extreme care. Incorrect use of this attribute can create security weaknesses. The **SuppressUnmanagedCodeSecurityAttribute** attribute should never be used to allow less-trusted code (code that does not have unmanaged code permission) to call into unmanaged code.

This attribute is best applied only to privately declared entry points to unmanaged code so that code in other assemblies cannot access and take advantage of the security suppression. Typically, the highly trusted managed code that uses this attribute first demands some permission of callers before invoking the unmanaged code on the caller's behalf.

The following example shows the **SuppressUnmanagedCodeSecurityAttribute** attribute applied to a private entry point.

VB

<SuppressUnmanagedCodeSecurityAttribute()> Private Declare Sub

EntryPoint Lib "some.dll"(args As String)

[C#]

[SuppressUnmanagedCodeSecurityAttribute()]

[DllImport("some.dll")]

private static extern void EntryPoint(string args);

In the rare case of unmanaged code that is completely safe for all possible circumstances, a method with the**SuppressUnmanagedCodeSecurityAttribute** attribute can be exposed to other managed code directly by making it public instead of private. If you choose to expose a method that has the **SuppressUnmanagedCodeSecurityAttribute** attribute, the functionality of the unmanaged code must be not only safe but also impervious to attack by malicious callers. For example, the code must operate appropriately even when unintended arguments are fabricated to specifically cause the code to malfunction.

**Using Declarative Overrides and Imperative Demands**

Asserts and other overrides are fastest when made declaratively, while demands are fastest when made imperatively. Although the performance gains might not be dramatic, using declarative overrides and imperative demands can help you improve code performance.

See Also

[Code Access Security](https://msdn.microsoft.com/en-us/library/930b76w0(v=vs.71).aspx) | [Writing Secure Class Libraries](https://msdn.microsoft.com/en-us/library/e942ksxt(v=vs.71).aspx) | [File.GetCreationTime Method](https://msdn.microsoft.com/en-us/library/66x02450(v=vs.71).aspx) | [SecurityPermission Class](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermission(v=vs.71).aspx) |[SuppressUnmanagedCodeSecurityAttribute Class](https://msdn.microsoft.com/en-us/library/system.security.suppressunmanagedcodesecurityattribute(v=vs.71).aspx)

### Writing Secure Managed Controls

**.NET Framework 1.1, 2.0, 3.0, 3.5**

Managed controls are assemblies referenced from Web pages that are downloaded to the user's computer and executed upon demand. From a code access security perspective, there are two types of managed controls: those that are run under the default security policy and those that require higher trust.

To write managed controls that are intended to run under the default security policy, you only need to know what operations are allowed by the default security policy for the intranet or Internet zones. As long as a managed control does not require more permission to execute than it receives as a result of its zone of origin, it will run. (Keep in mind that an administrator or user might decide not to grant as much permission to code from the Internet or intranet zones.) To execute managed controls that demand higher trust, the administrator or user must adjust the security policy of any computer that will run the code.

Whenever possible, managed controls should be written so that they do not require permissions that are not granted to Internet or intranet code by default. For the Internet zone, this means that code is restricted to displaying SafeTopLevelWindows and SafeSubWindows (adorned by the security system so as to prevent them from impersonating system dialogs), communicating back only to its site of origin, and using limited, isolated storage.

Code on the intranet has slightly greater permissions. For complete details, see [Default Security Policy](https://msdn.microsoft.com/en-us/library/03kwzyfc(v=vs.71).aspx). If your control needs to access files, use databases, collect information about the client computer, and so on, it needs some higher degree of trust.

**Development**

High-trust controls are intended to run under a less-restrictive security policy than their origin (intranet or the Internet) would typically warrant. Most permission demands made by secure libraries, such as.NET Framework classes, perform stack walks that check all callers to ensure that they have been granted the demanded permission and that Web pages, although not managed code, are treated as callers for the purposes of security. Stack walks are performed to help prevent less-trusted code from luring highly trusted code into performing malicious operations.

Because managed controls hosted in a browser can be manipulated by active script on a Web page, that Web page is considered a caller and checked during a security stack walk to help prevent malicious Web-page authors from exploiting more highly trusted code. The consequence of treating a Web page as a caller is that a control granted a high level of trust based on its strong name or publisher certificate and run from a Web page will be prevented from performing operations not normally allowed to code originating from the same zone as the Web page itself (intranet or Internet). For more information about deployment considerations, see the next section. On the face of it, this would seem to make writing high-trust controls impossible, but code access security provides for this scenario by enabling you to selectively override the security stack walk behavior.

High-trust controls must make judicious use of **Asserts** to short-circuit stack walks for permissions that their callers (the Web pages from which they were run) would not normally have. When you use **Asserts**, you must be careful not to expose dangerous APIs that would allow malicious Web pages to perform inappropriate operations. For this reason, the level of care and security consciousness that must go into writing a high-trust control approaches that required to write a secure class library.

The following are some tips for writing securely managed controls:

* When possible, encapsulate the operations requiring high trust so that they are not exposed by the control. In this manner, you can assert the permissions required by those operations and have confidence Web pages using your control will not be able to abuse the functionality.
* If the design of your control requires that the high-trust operations it performs be exposed, consider making a site or URL identity permission demand to ensure that it can be called only by the Web pages from which you intended it to run.

**Deployment**

High-trust controls should always be strong-named or signed with a publisher certificate (X.509). This allows policy administrators to grant higher trust to these controls without reducing their security with regard to other intranet or Internet code. After the assembly is signed, the user must create a new code group associated with sufficient permissions, and specify that only code signed by the user's company or organization be allowed membership to the code group. After the security policy is modified in this manner, the highly trusted control will receive sufficient permission to execute.

Because security policy must be modified to allow high-trust, downloaded controls to function, deploying this type of control is much easier on a corporate intranet for which there is typically an enterprise administrator who can deploy the described policy changes to multiple client computers. In order for high-trust controls to be used over the Internet by general users with no common corporate or organizational affiliation, there must be a trust relationship between the control publisher and the user. Ultimately, the user must be comfortable using the publisher's instructions to modify the policy and to allow the high-trust control to execute. Otherwise, the control will not be allowed to run.

See Also

[Introduction to Code Access Security](https://msdn.microsoft.com/en-us/library/c5tk9z76(v=vs.71).aspx) | [Writing Secure Class Libraries](https://msdn.microsoft.com/en-us/library/e942ksxt(v=vs.71).aspx) | [Creating and Using Strong-Named Assemblies](https://msdn.microsoft.com/en-us/library/xwb8f617(v=vs.71).aspx) | [Security Policy Management](https://msdn.microsoft.com/en-us/library/c1k0eed6(v=vs.71).aspx) | [Default Security Policy](https://msdn.microsoft.com/en-us/library/03kwzyfc(v=vs.71).aspx)

### Creating Your Own Code Access Permissions

**.NET Framework 1.1, 2.0, 3.0, 3.5**

The .NET Framework supplies a set of code access permission classes designed to help protect a specific set of resources and operations, focusing on those resources exposed by the .NET Framework. These permission classes are described briefly in the [Permissions](https://msdn.microsoft.com/en-us/library/5ba4k1c5(v=vs.71).aspx) topic and in detail in the reference documentation for each permission class. For most environments, the built-in code access permissions are adequate. However, in some situations, it might make sense to define your own code access permission class. This topic discusses when, why, and how to define custom code access permission classes.

If you are defining a component or class library that accesses a resource that is not covered by the built-in permission classes but needs to be protected from unauthorized code, you should consider creating a custom code access permission class. If you want to be able to make declarative demands for your custom permission, you must also define an [attribute](https://msdn.microsoft.com/en-us/library/5x6cd29c(v=vs.71).aspx) class for the permission. Providing these classes and making demands for the permission from within your class library enables the runtime to prevent unauthorized code from accessing that resource and enables an administrator to configure access rights.

There are other situations in which a custom permission might be appropriate. When a built-in code access permission class protects a resource but does not sufficiently control access to that resource, you might need a custom code access permission. For example, an application might use personnel records for which each employee record is stored in a separate file; in such a case, read and write access could be controlled independently for different types of employee data. An internal management tool could be authorized to read certain sections of an employee's personnel file but not to modify those sections. In fact, it might not even be allowed to read some sections.

Custom code access permissions are also appropriate in cases where a built-in permission exists but is not defined in a way that enables it to protect the resource appropriately. For example, there might be a case in which there is UI functionality, such as the ability to create menus, that must be protected but is not protected by the built-in [UIPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.uipermission(v=vs.71).aspx) class. In that case, you could create a custom permission to protect the ability to create menus.

Wherever possible, permissions should not overlap. Having more than one permission protecting a resource presents a significant problem for administrators, who must then be sure to deal appropriately with all the overlapping permissions every time they configure the rights to access that resource.

Implementing a custom code access permission involves the following steps, some of which are optional. Each step is described in a separate topic.

1. [Design the Permission class](https://msdn.microsoft.com/en-us/library/fskfdsy4(v=vs.71).aspx).
2. Implement the [IPermission](https://msdn.microsoft.com/en-us/library/system.security.ipermission(v=vs.71).aspx) and [IUnrestrictedPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.iunrestrictedpermission(v=vs.71).aspx) interfaces.
3. Implement the [ISerializable](https://msdn.microsoft.com/en-us/library/system.runtime.serialization.iserializable(v=vs.71).aspx) interface, if necessary for performance or to support special data types.
4. Handle XML encoding and decoding.
5. [Add support for declarative security](https://msdn.microsoft.com/en-us/library/84kh7ht8(v=vs.71).aspx), by implementing an **Attribute** class.
6. [Demand custom permission](https://msdn.microsoft.com/en-us/library/d8tb8sc9(v=vs.71).aspx) for your permission, where appropriate.
7. [Update security policy](https://msdn.microsoft.com/en-us/library/t0385k7y(v=vs.71).aspx) to be aware of the custom permission.

### Creating Your Own Code Access Permissions

**.NET Framework 4**

The .NET Framework supplies a set of code access permission classes designed to help protect a specific set of resources and operations, focusing on those resources exposed by the .NET Framework. These permission classes are described briefly in the [Permissions](https://msdn.microsoft.com/en-us/library/5ba4k1c5(v=vs.100).aspx) topic and in detail in the reference documentation for each permission class. For most environments, the built-in code access permissions are adequate. However, in some situations, it might make sense to define your own code access permission class. This topic discusses when, why, and how to define custom code access permission classes.

If you are defining a component or class library that accesses a resource that is not covered by the built-in permission classes but needs to be protected from unauthorized code, you should consider creating a custom code access permission class. If you want to be able to make declarative demands for your custom permission, you must also define an [attribute](https://msdn.microsoft.com/en-us/library/5x6cd29c(v=vs.100).aspx) class for the permission. Providing these classes and making demands for the permission from within your class library enables the runtime to prevent unauthorized code from accessing that resource and enables an administrator to configure access rights.

There are other situations in which a custom permission might be appropriate. When a built-in code access permission class protects a resource but does not sufficiently control access to that resource, you might need a custom code access permission. For example, an application might use personnel records for which each employee record is stored in a separate file; in such a case, read and write access could be controlled independently for different types of employee data. An internal management tool could be authorized to read certain sections of an employee's personnel file but not to modify those sections. In fact, it might not even be allowed to read some sections.

Custom code access permissions are also appropriate in cases where a built-in permission exists but is not defined in a way that enables it to protect the resource appropriately. For example, there might be a case in which there is UI functionality, such as the ability to create menus, that must be protected but is not protected by the built-in [UIPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.uipermission(v=vs.100).aspx) class. In that case, you could create a custom permission to protect the ability to create menus.

Wherever possible, permissions should not overlap. Having more than one permission protecting a resource presents a significant problem for administrators, who must then be sure to deal appropriately with all the overlapping permissions every time they configure the rights to access that resource.

Implementing a custom code access permission involves the following steps, some of which are optional. Each step is described in a separate topic.

1. [Design the Permission class](https://msdn.microsoft.com/en-us/library/fskfdsy4(v=vs.100).aspx).
2. Implement the [IPermission](https://msdn.microsoft.com/en-us/library/system.security.ipermission(v=vs.100).aspx) and [IUnrestrictedPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.iunrestrictedpermission(v=vs.100).aspx) interfaces.
3. Implement the [ISerializable](https://msdn.microsoft.com/en-us/library/system.runtime.serialization.iserializable(v=vs.100).aspx) interface, if necessary for performance or to support special data types.
4. Handle XML encoding and decoding.
5. [Add support for declarative security](https://msdn.microsoft.com/en-us/library/84kh7ht8(v=vs.100).aspx), by implementing an **Attribute** class.
6. [Demand custom permission](https://msdn.microsoft.com/en-us/library/d8tb8sc9(v=vs.100).aspx) for your permission, where appropriate.

[**See Also**](javascript:void(0))

Reference

[FileIOPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.fileiopermission(v=vs.100).aspx)

[UIPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.uipermission(v=vs.100).aspx)

[IPermission](https://msdn.microsoft.com/en-us/library/system.security.ipermission(v=vs.100).aspx)

[IUnrestrictedPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.iunrestrictedpermission(v=vs.100).aspx)

[ISerializable](https://msdn.microsoft.com/en-us/library/system.runtime.serialization.iserializable(v=vs.100).aspx)

Concepts

[Security Permissions](https://msdn.microsoft.com/en-us/library/5ba4k1c5(v=vs.100).aspx)

[Designing a Permission](https://msdn.microsoft.com/en-us/library/fskfdsy4(v=vs.100).aspx)

[Adding Declarative Security Support](https://msdn.microsoft.com/en-us/library/84kh7ht8(v=vs.100).aspx)

[Demanding a Custom Permission](https://msdn.microsoft.com/en-us/library/d8tb8sc9(v=vs.100).aspx)

Other Resources

[Code Access Security](https://msdn.microsoft.com/en-us/library/c5tk9z76(v=vs.100).aspx)

[Extending Metadata Using Attributes](https://msdn.microsoft.com/en-us/library/5x6cd29c(v=vs.100).aspx)

#### Designing a Permission

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4.0**

A permission represents the ability to access a protected resource or perform a protected operation. When you are implementing your own permission class, you must make several high-level design decisions. One of the first steps is to determine exactly what resource your custom permission is designed to protect.

You next want to decide whether there are any concerns about overlapping permissions. Although you want to avoid having two permissions that protect the same resource, in some situations you cannot reasonably avoid it. For example, the permission to access unmanaged code can also encompass other permissions, because code that is granted permission to access unmanaged code can do almost anything through an unmanaged API. However, when permission to access unmanaged code is not granted, you still need to grant permissions to access other specific resources. Therefore, it makes sense for permission to access unmanaged code to be separate from other permissions.

How do you know whether an overlap in permission coverage is manageable? There is no absolute answer, but one thing to think about is whether one of the permissions represents more fine-grained access than the other permission so that it will typically be more readily granted than the other permission. When this is the case, granting of access rights is easy to do in many cases, which makes the administrator's task easier.

After you have decided what resource your permission will protect and have resolved any issues about overlap of permissions, you must decide how finely grained access control should be. The answer to this question affects the way you design the variables that represent the state of the permission and determines whether administrators can configure access to the protected resource. It might also affect performance, ease of use, and other factors.

To illustrate some of these design issues, consider a few of the designs that could have been chosen for the [FileIOPermission Class](https://msdn.microsoft.com/en-us/library/system.security.permissions.fileiopermission(v=vs.71).aspx) that the .NET Framework provides. Each design choice affects the variables that represent the permission's state.

* A single bit that means "use all files" or "use no files", depending on its value.
* Two bits meaning "read all files" and "write all files", or not, depending on their values.
* 26 bits meaning "use all files on the specified drive".
* An array of strings listing all files to which access is given.

Clearly, there are various tradeoffs to consider. For example, the single-bit permission is very simple, fast, and easy to understand, but it presents an all-or-nothing choice for administrators, which might not be desirable. Other choices that specify a more complex representation of permission state could slow performance to some degree. You must weigh these tradeoffs, and consider that you should not create more than one permission to protect the same resource. In general, you should design your permission class so that the state of the permission is as complex as necessary, without greatly impacting performance.

Although other designs are possible, most permissions follow one of the following standard design patterns or a combination thereof:

* Boolean permissions. This simplest kind of permission object holds one or more bits, each of which corresponds to "permission to do X". Either you have the permission or you do not. An example of this type of permission is the [SecurityPermission class](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermission(v=vs.71).aspx), whose state contains Boolean variables representing the right to do different things, such as permission to call into unmanaged code, each of which is either allowed or not.
* Levels of permissions. This more detailed form of permission has variables that represent each kind of access as a number from zero (meaning no access at all) to some higher number (meaning totally unrestricted access), with a few levels between the two. For example, you can use the [UIPermission class](https://msdn.microsoft.com/en-us/library/system.security.permissions.uipermission(v=vs.71).aspx) to express varying levels of permission for using windows, from no UI permission to unrestricted UI permission, with a few gradations between the two.
* Object list permissions. This kind of permission provides a very detailed specification for what is and is not allowed. The[FileIOPermission class](https://msdn.microsoft.com/en-us/library/system.security.permissions.fileiopermission(v=vs.71).aspx) is a good example of this type of permission because its state is represented by lists of files on which certain kinds of access are allowed. Permissions with lists are most useful for protecting resources that contain a large number of named objects.

In general, it is a good idea to minimize external dependencies in your custom permission class because every assembly your permission depends on will have to be loaded when the security system needs your permission class. Another reason to minimize dependencies is that every assembly used by your custom permission class (except Mscorlib) must be added to the full-trust list. (For more information, see[Updating Security Policy](https://msdn.microsoft.com/en-us/library/t0385k7y(v=vs.71).aspx).) If possible, you should place your custom permission and any attribute classes associated with it in a separate assembly, to reduce the likelihood that other assemblies will be loaded unnecessarily.

**Note**Custom permissions should either be marked as sealed (NotInheritable in Visual Basic) or have an inheritance demand placed on them. Otherwise, malicious callers are able to derive from your permission, potentially causing security vulnerabilities.

#### Implementing a Custom Permission

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4.0**

All permission objects must implement the [IPermission](https://msdn.microsoft.com/en-us/library/system.security.ipermission(v=vs.71).aspx) interface. Inheriting from the [CodeAccessPermission](https://msdn.microsoft.com/en-us/library/system.security.codeaccesspermission(v=vs.71).aspx) class is the easiest way to create a custom permission because **CodeAccessPermission** implements **IPermission** and provides most of the methods required for a permission. Additionally, you must implement the [IUnrestrictedPermision](https://msdn.microsoft.com/en-us/library/system.security.permissions.iunrestrictedpermission(v=vs.71).aspx) interface for all custom code access permissions. The custom permission class is required for both imperative and declarative security support, so you should create it even if you plan to use only declarative security.

|  |
| --- |
| **NoteNote:** |
| The custom permission should be defined in an assembly other than the assembly in which it is referenced. If the custom permission includes a security attribute for declarative security, the custom permission and the attribute must be defined in a separate assembly. This is because the security attribute is executed when the assembly is loaded, and the attribute may not have been created at the time the reference to it is encountered. Attempting to use a declarative permission in the same assembly in which it is defined results in a[TypeLoadException](https://msdn.microsoft.com/en-us/library/system.typeloadexception(v=vs.90).aspx) being thrown. |

**Defining the Permission Class**

To derive from the **CodeAccessPermission** class, you must override the following five key methods and provide your own implementation:

* **Copy** creates a duplicate of the current permission object.
* **Intersect** returns the intersection of allowed permissions of the current class and a passed class.
* **IsSubsetOf** returns **true** if a passed permission includes everything allowed by the current permission.
* **FromXml** decodes an XML representation of your custom permission.
* **ToXml** encodes an XML representation of your custom permission.

The **IUnrestrictedPermission** interface requires you to override and implement a single method called **IsUnrestrictedPermission**. In order to support the **IUnrestrictedPermission** interface, you must implement some system, such as a Boolean value that represents the state of restriction in the current object, to define whether the current instance of the permission is unrestricted.

The following code fragment illustrates the manner in which a custom permission class might be defined. A constructor that accepts a**PermissionState** enumeration and a Boolean value called unrestricted are both created. The **PermissionState** enumeration has a value of either **Unrestricted** or **None**. If the passed enumeration has a value of **Unrestricted**, the constructor sets unrestricted to **true**. Otherwise,unrestricted is set to **false**. In addition to constructors specific to your custom permission, all code access permissions (any permission that inherits from **CodeAccessPermission**) must support a constructor that takes only a **PermissionState** enumeration.

In addition to the code shown in the following example, you must implement **IsUnrestricted** method and override the **Copy**, **Intersect**,**IsSubsetOf**, **ToXML**, and **FromXML**methods. For information about completing these steps, see the sections that follow the example.

VB

Option Strict

Option Explicit

Imports System

Imports System.Security

Imports System.Security.Permissions

<SerializableAttribute()> NotInheritable Public Class CustomPermission

Inherits CodeAccessPermission

Implements IUnrestrictedPermission

Private unrestricted As Boolean

Public Sub New(state As PermissionState)

If state = PermissionState.Unrestricted Then

unrestricted = True

Else

unrestricted = False

End If

End Sub

'Define the rest of your custom permission here. You must

'implement IsUnrestricted and override the Copy, Intersect,

'IsSubsetOf, ToXML, and FromXML methods.

End Class

[C#]

using System;

using System.Security;

using System.Security.Permissions;

[SerializableAttribute()]

public sealed class CustomPermission: CodeAccessPermission, IUnrestrictedPermission

{

private bool unrestricted;

public CustomPermission(PermissionState state)

{

if(state == PermissionState.Unrestricted)

{

unrestricted = true;

}

else

{

unrestricted = false;

}

}

//Define the rest of your custom permission here. You must

//implement IsUnrestricted and override the Copy, Intersect,

//IsSubsetOf, ToXML, and FromXML methods.

}

Notice that the class is marked with [SerializableAttribute](https://msdn.microsoft.com/en-us/library/system.serializableattribute(v=vs.71).aspx). You must mark your class with **SerializableAttribute** in order to support declarative syntax using an attribute. For information about creating a custom attribute that uses a custom security object, see [Adding Declarative Security Support](https://msdn.microsoft.com/en-us/library/84kh7ht8(v=vs.71).aspx).

**Implementing the IsUnrestricted Method**

The **IsUnrestricted** method is required by the **IUnrestrictedPermission** interface and simply returns a Boolean value that indicates whether the current instance of the permission has unrestricted access to the resource protected by the permission. To implement this method, simply return the value of unrestricted.

The following code example implements the **IsUnrestricted** method.

VB

Public Function IsUnrestricted() As Boolean Implements IUnrestrictedPermission.IsUnrestricted

Return unrestricted

End Function

[C#]

public bool IsUnrestricted()

{

return unrestricted;

}

**Overriding the Copy Method**

The copy method is required by the **CodeAccessPermission** class and returns a copy of the current permission class.

The following code illustrates how to override the **Copy** method.

VB

Public Overrides Function Copy() As IPermission

Dim myCopy As New CustomPermission(PermissionState.None)

If Me.IsUnrestricted() Then

myCopy.unrestricted = True

Else

myCopy.unrestricted = False

End If

Return myCopy

End Function

[C#]

public override IPermission Copy()

{

CustomPermission copy = new CustomPermission(PermissionState.None);

if(this.IsUnrestricted())

{

copy.unrestricted = true;

}

else

{

copy.unrestricted = false;

}

return copy;

}

**Overriding the Intersect and IsSubsetOf Methods**

All permissions must implement the **Intersect** and **IsSubsetOf** methods. The behavior of these operations must be implemented as follows:

* X.IsSubsetOf(Y) is **true** if permission Y includes everything allowed by X.
* X.Intersect(Y) results in a permission that allows all operations and only those operations allowed by both the X and Y permissions.

The following example illustrates how to override and implement the **Intersect** method. The method accepts a class that derives from**IPermission** and initializes this class to a new instance of the CustomPermisison object. In this case, the intersection of the current object and the passed object is a final object with the value of unrestricted if both objects have that value. However, if one of the two objects has a**false** value for unrestricted, then the final object will also have a **false** value for unrestricted. This code returns an unrestricted object only if both objects are unrestricted.

VB

Public Overrides Function Intersect(target As IPermission) As IPermission

If Nothing Is target Then

Return Nothing

End If

Try

Dim PassedPermission As CustomPermission = CType(target, CustomPermission)

If Not PassedPermission.IsUnrestricted() Then

Return PassedPermission

End If

Return Me.Copy()

Catch InvalidCastException As Exception

Throw New ArgumentException("Argument\_WrongType", Me.GetType().FullName)

End Try

End Function

[C#]

public override IPermission Intersect(IPermission target)

{

try

{

if(null == target)

{

return null;

}

CustomPermission PassedPermission = (CustomPermission)target;

if(!PassedPermission.IsUnrestricted())

{

return PassedPermission;

}

return this.Copy();

}

catch (InvalidCastException)

{

throw new ArgumentException("Argument\_WrongType", this.GetType().FullName);

}

}

In the following example, the **IsSubsetOf** method is overridden. In order for this method to return **true**, the current instance and a passed instance must allow exactly the same set of operations. In this case, the overridden method initializes a new instance of theCustomPermission object to the passed permission object. If the unrestricted values are the same, then the method returns **true**. If they are not, the method returns **false**.

VB

Public Overrides Function IsSubsetOf(target As IPermission) As Boolean

If Nothing Is target Then

Return Not Me.unrestricted

End If

Try

Dim passedpermission As CustomPermission = CType(target, CustomPermission)

If Me.unrestricted = passedpermission.unrestricted Then

Return True

Else

Return False

End If

Catch InvalidCastException As Exception

Throw New ArgumentException("Argument\_WrongType", Me.GetType().FullName)

End Try

End Function

[C#]

public override bool IsSubsetOf(IPermission target)

{

if(null == target)

{

return !this.unrestricted;

}

try

{

CustomPermission passedpermission = (CustomPermission)target;

if(this.unrestricted == passedpermission.unrestricted)

{

return true;

}

else

{

return false;

}

}

catch (InvalidCastException)

{

throw new ArgumentException("Argument\_WrongType", this.GetType().FullName);

}

}

**Overriding the ToXml and FromXml Methods**

Permissions support XML encoding so that a permission object can be saved as XML and then another permission object can be restored to the value of the original, using the XML file. To support XML encoding, your custom permission must implement the [ISecurityEncodable](https://msdn.microsoft.com/en-us/library/system.security.isecurityencodable(v=vs.71).aspx)interface, which defines a **ToXml** and a **FromXml** method. Because both methods are implemented by [CodeAccessPermission](https://msdn.microsoft.com/en-us/library/system.security.codeaccesspermission(v=vs.71).aspx), if your custom permission class derives from **CodeAccessPermission**, you should override these methods.

The content of the XML element that represents the object state is determined by the object itself. The **FromXML** method can use any XML representation as long as **ToXML** can interpret it and restore the same state. However, the containing **Permission** element must be of a standard form. For example, the form for CustomPermission might look like the following:

<IPermission class="CustomPermissions.CustomPermission, CustomPermissionAssembly " version="1" Unrestricted="True">

The **IPermission** element contains three attributes:

* **class**: Contains the type name disambiguated by the name of the assembly that contains it.
* **version**: Specifies the version of the XML encoding (not the version of the class's assembly).
* **Unrestricted**: Specifies whether the permission has unrestricted rights.

All permissions must be encoded in an XML element called **IPermission** in order to be used by the common language runtime security system.

New versions of a permission object should remain backward compatible with information persisted in XML from previous versions. The version tag provides information to a permission object about which version originally encoded the data.

The [SecurityElement](https://msdn.microsoft.com/en-us/library/system.security.securityelement(v=vs.71).aspx) class encapsulates the main functionality that you need in order to create and interact with XML-encoded permission objects. However, because the XML object model used for .NET Framework security is different from other XML object models, the**SecurityElement** class should not be used to generate other types of XML files. See the description of the **SecurityElement** class for a complete list of its members.

The following code fragment creates an XML **Permission** element:

VB

Public Overrides Function ToXml() As SecurityElement

Dim element As New SecurityElement("IPermission")

Dim type As Type = Me.GetType()

Dim AssemblyName As New StringBuilder(type.Assembly.ToString())

AssemblyName.Replace(ControlChars.Quote, "'"c)

element.AddAttribute("class", type.FullName & ", " & AssemblyName.ToString)

element.AddAttribute("version", "1")

element.AddAttribute("Unrestricted", unrestricted.ToString())

Return element

End Function

[C#]

public override SecurityElement ToXml()

{

SecurityElement element = new SecurityElement("IPermission");

Type type = this.GetType();

StringBuilder AssemblyName = new StringBuilder(type.Assembly.ToString());

AssemblyName.Replace('\"', '\'');

element.AddAttribute("class", type.FullName + ", " + AssemblyName);

element.AddAttribute("version", "1");

element.AddAttribute("Unrestricted", unrestricted.ToString());

return element;

}

Notice that the previous example uses the **StringBuilder.Replace** method. Attributes in the **SecurityElement** class cannot contain double quotes, but some assembly name information is in double quotes. To handle this situation, the **Replace** method converts double quotes (") in the assembly name to single quotes (').

The following method reads a **SecurityElement** object created by the previous method and sets the current value of the **Unrestricted**property to the one specified by the passed object. This method should ensure that any information stored by the **ToXml** method is retrieved.

VB

Public Overrides Sub FromXml(PassedElement As SecurityElement)

Dim element As String = PassedElement.Attribute("Unrestricted")

If Not element Is Nothing Then

Me.unrestricted = Convert.ToBoolean(element)

End If

End Sub

[C#]

public override void FromXml(SecurityElement PassedElement)

{

string element = PassedElement.Attribute("Unrestricted");

if(null != element)

{

this.unrestricted = Convert.ToBoolean(element);

}

}

**Custom Permission Example**

The following code example shows an entire custom permission class:

VB

Option Explicit

Option Strict

Imports System

Imports System.Text

Imports System.Security

Imports System.Security.Permissions

Imports Microsoft.VisualBasic

<Serializable()>NotInheritable Public Class CustomPermission

Inherits CodeAccessPermission

Implements IUnrestrictedPermission

Private unrestricted As Boolean

Public Sub New(state As PermissionState)

If state = PermissionState.Unrestricted Then

unrestricted = True

Else

unrestricted = False

End If

End Sub

Public Function IsUnrestricted() As Boolean Implements IUnrestrictedPermission.IsUnrestricted

Return unrestricted

End Function

Public Overrides Function Copy() As IPermission

'Create a new instance of CustomPermission with the current

'value of unrestricted.

Dim myCopy As New CustomPermission(PermissionState.None)

If Me.IsUnrestricted() Then

myCopy.unrestricted = True

Else

myCopy.unrestricted = False

End If

'Return the copy.

Return copy

End Function

Public Overrides Function Intersect(target As IPermission) As IPermission

'If nothing was passed, return null.

If Nothing Is target Then

Return Nothing

End If

Try

'Create a new instance of CustomPermission from the passed object.

Dim PassedPermission As CustomPermission = CType(target, CustomPermission)

'If one class has an unrestricted value of false, then the

'intersection will have an unrestricted value of false.

'Return the passed class with the unrestricted value of false.

If Not PassedPermission.unrestricted Then

Return target

End If

'Return a copy of the current class if the passed one has

'an unrestricted value of true.

Return Me.Copy()

'Catch an InvalidCastException.

'Throw ArgumentException to notify the user.

Catch InvalidCastException As Exception

Throw New ArgumentException("Argument\_WrongType", Me.GetType().FullName)

End Try

End Function

Public Overrides Function IsSubsetOf(target As IPermission) As Boolean

'If nothing was passed and unrestricted is false,

' return true.

If Nothing Is target Then

Return Not Me.unrestricted

End If

Try

'Create a new instance of CustomPermission from the passed object.

Dim passedpermission As CustomPermission = CType(target, CustomPermission)

'If unrestricted has the same value in both objects, then

'one is the subset of the other.

If Me.unrestricted = passedpermission.unrestricted Then

Return True

Else

Return False

End If

'Catch an InvalidCastException.

'Throw ArgumentException to notify the user.

Catch InvalidCastException As Exception

Throw New ArgumentException("Argument\_WrongType", Me.GetType().FullName)

End Try

End Function

Public Overrides Sub FromXml(PassedElement As SecurityElement)

'Get the unrestricted value from the XML and initialize

'the current instance of unrestricted to that value.

Dim element As String = PassedElement.Attribute("Unrestricted")

If Not element Is Nothing Then

Me.unrestricted = Convert.ToBoolean(element)

End If

End Sub

Public Overrides Function ToXml() As SecurityElement

'Encode the current permission to XML using the

'SecurityElement class.

Dim element As New SecurityElement("IPermission")

Dim type As Type = Me.GetType()

Dim AssemblyName As New StringBuilder(type.Assembly.ToString())

AssemblyName.Replace(ControlChars.Quote, "'"c)

element.AddAttribute("class", type.FullName & ", " & AssemblyName.ToString)

element.AddAttribute("version", "1")

element.AddAttribute("Unrestricted", unrestricted.ToString())

Return element

End Function

End Class

[C#]

using System;

using System.Text;

using System.Security;

using System.Security.Permissions;

[Serializable()]

public sealed class CustomPermission: CodeAccessPermission , IUnrestrictedPermission

{

private bool unrestricted;

public CustomPermission(PermissionState state)

{

if(state == PermissionState.Unrestricted)

{

unrestricted = true;

}

else

{

unrestricted = false;

}

}

public bool IsUnrestricted()

{

return unrestricted;

}

public override IPermission Copy()

{

//Create a new instance of CustomPermission with the current

//value of unrestricted.

CustomPermission copy = new CustomPermission(PermissionState.None);

if(this.IsUnrestricted())

{

copy.unrestricted = true;

}

else

{

copy.unrestricted = false;

}

//Return the copy.

return copy;

}

public override IPermission Intersect(IPermission target)

{

//If nothing was passed, return null.

if(null == target)

{

return null;

}

try

{

//Create a new instance of CustomPermission from the passed object.

CustomPermission PassedPermission = (CustomPermission)target;

//If one class has an unrestricted value of false, then the

//intersection will have an unrestricted value of false.

//Return the passed class with the unrestricted value of false.

if(!PassedPermission.unrestricted)

{

return target;

}

//Return a copy of the current class if the passed one has

//an unrestricted value of true.

return this.Copy();

}

//Catch an InvalidCastException.

//Throw ArgumentException to notify the user.

catch (InvalidCastException)

{

throw new ArgumentException("Argument\_WrongType", this.GetType().FullName);

}

}

public override bool IsSubsetOf(IPermission target)

{

//If nothing was passed and unrestricted is false,

//then return true.

if(null == target)

{

return !this.unrestricted;

}

try

{

//Create a new instance of CustomPermission from the passed object.

CustomPermission passedpermission = (CustomPermission)target;

//If unrestricted has the same value in both objects, then

//one is the subset of the other.

if(this.unrestricted == passedpermission.unrestricted)

{

return true;

}

else

{

return false;

}

}

//Catch an InvalidCastException.

//Throw ArgumentException to notify the user.

catch (InvalidCastException)

{

throw new ArgumentException("Argument\_WrongType", this.GetType().FullName);

}

}

public override void FromXml(SecurityElement PassedElement)

{

//Get the unrestricted value from the XML and initialize

//the current instance of unrestricted to that value.

string element = PassedElement.Attribute("Unrestricted");

if(null != element)

{

this.unrestricted = Convert.ToBoolean(element);

}

}

public override SecurityElement ToXml()

{

//Encode the current permission to XML using the

//SecurityElement class.

SecurityElement element = new SecurityElement("IPermission");

Type type = this.GetType();

StringBuilder AssemblyName = new StringBuilder(type.Assembly.ToString());

AssemblyName.Replace('\"', '\'');

element.AddAttribute("class", type.FullName + ", " + AssemblyName);

element.AddAttribute("version", "1");

element.AddAttribute("Unrestricted", unrestricted.ToString());

return element;

}

}

#### Adding Declarative Security Support

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4.0**

Although not strictly required, a custom permission should support declarative security so that developers can specify the custom permission when using declarative syntax for security actions such as requests, demands, or assertions. In fact, permission requests, link demands, and inheritance demands can only be made declaratively. For this reason, your custom code access permission cannot be requested or used with link demands or inheritance demands unless you provide support for declarative security. This topic describes how to implement an **Attribute**class that enables declarative security support for your custom permission.

|  |
| --- |
| **NoteNote** |
| The attribute version of the custom permission must be defined in an assembly other than the assembly in which it is referenced. The custom permission should also be defined in that assembly. This is mandatory for declarative security, because the attribute is executed when the assembly is loaded, and the attribute may not have been created at the time the reference to it is encountered. Attempting to use a declarative permission in the same assembly in which it is defined results in a [TypeLoadException](https://msdn.microsoft.com/en-us/library/system.typeloadexception(v=vs.100).aspx) being thrown. |

Security attributes for declarations must derive (either directly or indirectly) from the [SecurityAttribute](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityattribute(v=vs.71).aspx) class. If the permission is a code access permission, the attribute class derives from [CodeAccessSecurityAttribute](https://msdn.microsoft.com/en-us/library/system.security.permissions.codeaccesssecurityattribute(v=vs.71).aspx), which derives from **SecurityAttribute**. Security attribute classes must implement the **CreatePermission** method, which creates an instance of the permission object from the associated custom permission. Note that this associated custom permission class must be marked with the [SerializableAttribute](https://msdn.microsoft.com/en-us/library/system.serializableattribute(v=vs.71).aspx) in order to be serialized into metadata by the compiler. For more information, see [Implementing a Custom Permission](https://msdn.microsoft.com/en-us/library/yaah0wb2(v=vs.71).aspx).

The following code implements an attribute class for a Boolean permission named CustomPermission. In this example, the permission class has a single Boolean Unrestricted property that contains its state.

VB

<AttributeUsageAttribute(AttributeTargets.All, AllowMultiple := True)> Public Class

CustomPermissionAttribute

Inherits CodeAccessSecurityAttribute

Private myUnrestricted As Boolean = False

Public Shadows Property Unrestricted() As Boolean

Get

Return myUnrestricted

End Get

Set

myUnrestricted = value

End Set

End Property

Public Sub New(action As SecurityAction)

MyBase.New(action)

End Sub

Public Overrides Function CreatePermission() As IPermission

If Unrestricted Then

Return New CustomPermission(PermissionState.Unrestricted)

Else

Return New CustomPermission(PermissionState.None)

End If

End Function

End Class

[C#]

[AttributeUsageAttribute(AttributeTargets.All, AllowMultiple = true)]

public class CustomPermissionAttribute: CodeAccessSecurityAttribute

{

bool unrestricted = false;

public new bool Unrestricted

{

get{ return unrestricted; }

set{ unrestricted = value; }

}

public CustomPermissionAttribute(SecurityAction action): base (action)

{

}

public override IPermission CreatePermission()

{

if(Unrestricted)

{

return new CustomPermission(PermissionState.Unrestricted);

}

else

{

return new CustomPermission(PermissionState.None);

}

}

}

In this case, **CreatePermission** checks the internal Unrestricted property and creates the appropriate instance of a CustomPermissionobject. While only the Unrestricted property is used in this case, other custom permission attribute classes should support all possible states of the permission objects they support.

The use of CustomPermissionAttribute is illustrated in the following demand declaration:

VB

<CustomPermissionAttribute(SecurityAction.Demand, Unrestricted := true)>

[C#]

[CustomPermissionAttribute(SecurityAction.Demand, Unrestricted = true)]

See Also

#### Demanding a Custom Permission

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4.0**

To cause the runtime to perform security checks on code that attempts to access the resource protected by your custom code access permission, you must place security demands for the custom code access permission in your code. Security demands can be imperative or declarative; both cause the runtime to walk the call stack to see that all callers in the stack have the permission that is being demanded.

If your code simply calls a secure system library that exposes a protected resource, you do not need to demand a permission just because the library exposes the resource protected by that permission. It is the library's responsibility to demand that your code and all its callers have the necessary permission.

However, if your code calls a system library and [asserts](https://msdn.microsoft.com/en-us/library/91wteedy(v=vs.71).aspx) a permission that is demanded by the library (effectively causing a security check for that permission to succeed), then it must either demand that permission of its callers or otherwise ensure that security is not compromised.

For more information about when to make security demands, see [Security Demands](https://msdn.microsoft.com/en-us/library/60zfc754(v=vs.71).aspx).

#### Updating Security Policy

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4.0**

Default security policy does not know about the existence of any custom permission. For example, the **Everything** named permission set contains all the built-in code access permissions that the runtime provides, but it does not include any custom permissions. To update security policy so that it knows about your custom permission, you must do three things:

* Make policy aware of your custom permission.
* Add the assembly to the list of trusted assemblies.
* Tell security policy what code should be granted to your custom permission.

**Making Policy Aware of Your Custom Permission**

To make policy aware of your custom permission, you must:

* Create a new named-permission set that includes your custom permission. (You can modify an existing named-permission set instead of creating a new one.)
* Give the permission set a name.
* Tell security policy that the named-permission set exists.

For more information, see [Code Access Security Policy Tool (Caspol.exe)](https://msdn.microsoft.com/en-us/library/cb6t8dtz(v=vs.71).aspx) or [.NET Framework Configuration Tool (Mscorcfg.msc)](https://msdn.microsoft.com/en-us/library/2bc0cxhc(v=vs.71).aspx). You can add a new permission set in one of several ways. Using the Code Access Security Policy tool (Caspol.exe), you can create an .xml file that contains an XML representation of a custom permission set and then add this file to the security policy on the computer where the code is to run. Using the .NET Framework Configuration tool (Mscorcfg.msc), you can copy an existing permission set and add an XML representation of a permission to the new permission set.

To guarantee that your XML representation is valid and correctly represents your permission, you can generate it using code similar to the example that follows. Notice that this code creates a custom permission called MyCustomPermission, initialized to the unrestricted state. If your custom permission does not implement [IUnrestrictedPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.iunrestrictedpermission(v=vs.71).aspx), or if you do not want to set policy to grant your permission in an unrestricted state, use the constructor to initialize your permission to the state you want it to have.

VB

Imports System.IO

Imports System.Security.Permissions

Class PSetXML

Public Shared Sub Main()

Dim perm As New MyCustomPermission(PermissionState.Unrestricted)

Dim pset As New NamedPermissionSet("MyPermissionSet", PermissionState.None)

pset.Description = "Permission set containing my custom permission"

pset.AddPermission(perm)

Dim file As New StreamWriter("mypermissionset.xml")

file.Write(pset.ToXml())

file.Close()

End Sub

End Class

[C#]

using System.IO;

using System.Security.Permissions;

class PSetXML {

public static void Main()

{

MyCustomPermission perm =

new MyCustomPermission(PermissionState.Unrestricted);

NamedPermissionSet pset =

new NamedPermissionSet("MyPermissionSet", PermissionState.None);

pset.Description = "Permission set containing my custom permission";

pset.AddPermission(perm);

StreamWriter file = new StreamWriter("mypermissionset.xml");

file.Write(pset.ToXml());

file.Close();

}

}

After you have created the .xml file containing your permission set, you can add it to security policy. To use Caspol.exe, type the following on the command line:

caspol –machine –addpset mypermissionset.xml

When Caspol.exe asks whether you want to add the assembly containing your custom permission to the list of trusted assemblies, type **yes**.

To add the .xml file containing your permission set using the [.NET Framework Configuration tool](https://msdn.microsoft.com/en-us/library/2bc0cxhc(v=vs.71).aspx), select the **Runtime Security Policy** node and then select the policy level you want to modify. Right-click **Permission Sets**and select **New**. Use the wizard to add the permission set.

**Adding the Assembly to the List of Trusted Assemblies**

Because your custom permission will participate in the .NET Framework security system, it must be fully trusted (as any code that the security system relies on must be). You obtain full trust for your assembly by adding it to the list of trusted assemblies. After you have added your custom permission's assembly to the list using [Caspol.exe](https://msdn.microsoft.com/en-us/library/cb6t8dtz(v=vs.71).aspx) (as described previously), you must also add any assemblies that your permission class references. To add additional assemblies to the list using Caspol.exe, type the following on the command line:

caspol -addfulltrust mypermissionset.dll

To view the list of fully trusted assemblies, use the following command:

caspol -listfulltrust

Because your custom permission's assembly (and any assemblies it references) will be fully trusted by the security system, it is important that those files be signed with a cryptographically strong name. Caspol.exe will not add an assembly to the full trust list if it does not have a strong name.

To add an assembly to the list of fully trusted assemblies using the [.NET Framework Configuration tool](https://msdn.microsoft.com/en-us/library/2bc0cxhc(v=vs.71).aspx), right-click the **Runtime Security Policy** node and select **Trust Assembly**. Use the wizard to trust the assembly.

**CAUTION**If the assembly implementing the custom security object references other assemblies, you must first add the referenced assemblies to the full trust assembly list. Custom security objects created using Visual Basic .NET, the Managed Extensions for C++, and JScript reference Microsoft.VisualBasic.dll, Microsoft.VisualC.dll, or Microsoft.JScript.dll, respectively. These assemblies are not in the full trust assembly list by default. You must add the appropriate assembly to the full trust list before you add a custom security object. Failure to do so will break the security system, causing all assemblies to fail to load. In this situation, the Caspol.exe **-all -reset** option will not repair security. To repair security, you must manually edit the security files to remove the custom security object.

**Setting Policy to Grant Your Custom Permission**

You must associate your new permission set with the appropriate code groups so that security policy will grant the custom permission to the code that should have it. You do this by modifying an existing code group or by adding a new code group that identifies the set of code that should be granted your custom permission. For more information about code groups, see [Security Policy](https://msdn.microsoft.com/en-us/library/tha13y5z(v=vs.71).aspx). Use the following [Caspol.exe](https://msdn.microsoft.com/en-us/library/cb6t8dtz(v=vs.71).aspx)command to make mypermissionset the permission set granted to code that meets the membership condition of the **LocalIntranet** code group:

caspol -user -chggroup 1.2. mypermissionset

In this example, the label 1.2 represents the code group **LocalIntranet**. To display all the code groups and their associated labels, use the following command:

caspol -list

To view the list of permission sets, use the following command:

caspol -listpset

To make mypermissionset the permission set granted to members of the **LocalIntranet** code group using the [.NET Framework Configuration tool](https://msdn.microsoft.com/en-us/library/2bc0cxhc(v=vs.71).aspx), select the **Runtime Security Policy** node and then select the **Machine** policy. Right-click the **LocalIntranet\_Zone** node and select **Properties**. Change the permission set using the **Permission Set** tab.

#### Creating Other Custom Permissions

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4.0**

If you need to create a permission to protect a resource, but you do not want a demand for your custom permission to cause a stack walk, you should create a permission that implements [IPermission](https://msdn.microsoft.com/en-us/library/system.security.ipermission(v=vs.71).aspx) but does not derive from [CodeAccessPermission](https://msdn.microsoft.com/en-us/library/system.security.codeaccesspermission(v=vs.71).aspx). One example of a scenario in which this makes sense is a bank application that needs to restrict the ability of code to perform transactions, but for which you do not want the permissions of all callers in the stack to be checked to determine whether they are authorized to perform the transaction. Instead, you want authorization to be based on whether the current time of day falls within a specified range. This situation requires a custom permission object that is not a custom code access permission.

The requirements for this kind of permission are very similar to those for a custom code access permission. You must perform the following steps in order to use such a permission:

1. Implement **IPermission**.
2. Implement [ISecurityEncodable](https://msdn.microsoft.com/en-us/library/system.security.isecurityencodable(v=vs.71).aspx) to add support for XML encoding and decoding.
3. [Support declarative security](https://msdn.microsoft.com/en-us/library/84kh7ht8(v=vs.71).aspx) by defining an attribute.
4. Demand your custom permission within a library.

Note that security policy does not have to be updated to include non-code access permissions. You do not have to use the [Code Access Security Policy Tool (Caspol.exe)](https://msdn.microsoft.com/en-us/library/cb6t8dtz(v=vs.71).aspx) or the [.NET Framework Configuration tool (Mscorcfg.msc)](https://msdn.microsoft.com/en-us/library/2bc0cxhc(v=vs.71).aspx) to adjust security policy when using such a hypothetical permission.

### Role-Based Security

**.NET Framework 1.1**

Business applications often provide access to data or resources based on credentials supplied by the user. Typically, such applications check the role of a user and provide access to resources based on that role. The common language runtime provides support for role-based authorization based on a Windows account or a custom identity. Before reading this section, make sure you understand the material presented in [Key Security Concepts](https://msdn.microsoft.com/en-us/library/z164t8hs(v=vs.71).aspx).

In This Section

[Introduction to Role-Based Security](https://msdn.microsoft.com/en-us/library/shz8h065(v=vs.71).aspx)

Provides an overview of .NET Framework role-based security.

[Principal and Identity Objects](https://msdn.microsoft.com/en-us/library/ftx85f8x(v=vs.71).aspx)

Explains how to set up and manage both Windows and generic identities and principals.

[PrincipalPermission Objects](https://msdn.microsoft.com/en-us/library/765xy944(v=vs.71).aspx)

Describes how to implement permission objects declaratively and imperatively.

[Role-Based Security Checks](https://msdn.microsoft.com/en-us/library/k8b3sz1a(v=vs.71).aspx)

Describes how to implement security checks declaratively, imperatively, or by directly accessing a principal object.

[Interoperation with COM+ 1.0 Security](https://msdn.microsoft.com/en-us/library/x740xszs(v=vs.71).aspx)

Provides an overview of how COM+ 1.0 security interoperates with the .NET Framework.

Related Sections

[Key Security Concepts](https://msdn.microsoft.com/en-us/library/z164t8hs(v=vs.71).aspx)

Introduces fundamental concepts you must understand before using .NET Framework security.

[Security Tools](https://msdn.microsoft.com/en-us/library/7w3fd0wb(v=vs.71).aspx)

Lists and briefly describes the security tools included in the .NET Framework SDK.

#### Introduction to Role-Based Security

**.NET Framework 1.1, 2.0, 3.0, 3.5**

Roles are often used in financial or business applications to enforce policy. For example, an application might impose limits on the size of the transaction being processed depending on whether the user making the request is a member of a specified role. Clerks might have authorization to process transactions that are less than a specified threshold, supervisors might have a higher limit, and vice-presidents might have a still higher limit (or no limit at all). Role-based security can also be used when an application requires multiple approvals to complete an action. Such a case might be a purchasing system in which any employee can generate a purchase request, but only a purchasing agent can convert that request into a purchase order that can be sent to a supplier.

.NET Framework role-based security supports authorization by making information about the [principal](https://msdn.microsoft.com/en-us/library/axt6w9h8(v=vs.71).aspx), which is constructed from an associated identity, available to the current thread. The identity (and the principal it helps to define) can be either based on a Windows account or be a custom identity unrelated to a Windows account. .NET Framework applications can make authorization decisions based on the principal's identity or role membership, or both. A role is a named set of principals that have the same privileges with respect to security (such as a teller or a manager). A principal can be a member of one or more roles. Therefore, applications can use role membership to determine whether a principal is authorized to perform a requested action.

To provide ease of use and consistency with code access security, .NET Framework role-based security provides [PrincipalPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.principalpermission(v=vs.71).aspx) objects that enable the common language runtime to perform authorization in a way that is similar to code access security checks. The **PrincipalPermission** class represents the identity or role that the principal must match and is compatible with both declarative and imperative security checks. You can also access a principal's identity information directly and perform role and identity checks in your code when needed.

The .NET Framework provides role-based security support that is flexible and extensible enough to meet the needs of a wide spectrum of applications. You can choose to interoperate with existing authentication infrastructures, such as COM+ 1.0 Services, or to create a custom authentication system. Role-based security is particularly well-suited for use in ASP.NET Web applications, which are processed primarily on the server. However, .NET Framework role-based security can be used on either the client or the server.

#### Role-Based Security

**.NET Framework 4, 4.5, 4.6**

Roles are often used in financial or business applications to enforce policy. For example, an application might impose limits on the size of the transaction being processed depending on whether the user making the request is a member of a specified role. Clerks might have authorization to process transactions that are less than a specified threshold, supervisors might have a higher limit, and vice-presidents might have a still higher limit (or no limit at all). Role-based security can also be used when an application requires multiple approvals to complete an action. Such a case might be a purchasing system in which any employee can generate a purchase request, but only a purchasing agent can convert that request into a purchase order that can be sent to a supplier.

.NET Framework role-based security supports authorization by making information about the [Principal](https://msdn.microsoft.com/en-us/library/axt6w9h8(v=vs.100).aspx), which is constructed from an associated identity, available to the current thread. The identity (and the principal it helps to define) can be either based on a Windows account or be a custom identity unrelated to a Windows account. .NET Framework applications can make authorization decisions based on the principal's identity or role membership, or both. A role is a named set of principals that have the same privileges with respect to security (such as a teller or a manager). A principal can be a member of one or more roles. Therefore, applications can use role membership to determine whether a principal is authorized to perform a requested action.

To provide ease of use and consistency with code access security, .NET Framework role-based security provides[System.Security.Permissions.PrincipalPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.principalpermission(v=vs.100).aspx) objects that enable the common language runtime to perform authorization in a way that is similar to code access security checks. The [PrincipalPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.principalpermission(v=vs.100).aspx) class represents the identity or role that the principal must match and is compatible with both declarative and imperative security checks. You can also access a principal's identity information directly and perform role and identity checks in your code when needed.

The .NET Framework provides role-based security support that is flexible and extensible enough to meet the needs of a wide spectrum of applications. You can choose to interoperate with existing authentication infrastructures, such as COM+ 1.0 Services, or to create a custom authentication system. Role-based security is particularly well-suited for use in ASP.NET Web applications, which are processed primarily on the server. However, .NET Framework role-based security can be used on either the client or the server.

Before reading this section, make sure that you understand the material presented in [Key Security Concepts](https://msdn.microsoft.com/en-us/library/z164t8hs(v=vs.100).aspx).

[**Related Topics**](javascript:void(0)) 4.0

|  |  |
| --- | --- |
| **Title** | **Description** |
| [Principal and Identity Objects](https://msdn.microsoft.com/en-us/library/ftx85f8x(v=vs.100).aspx) | Explains how to set up and manage both Windows and generic identities and principals. |
| [PrincipalPermission Objects](https://msdn.microsoft.com/en-us/library/765xy944(v=vs.100).aspx) | Describes how to implement permission objects declaratively and imperatively. |
| [Role-Based Security Checks](https://msdn.microsoft.com/en-us/library/k8b3sz1a(v=vs.100).aspx) | Describes how to implement security checks declaratively, imperatively, or by directly accessing a principal object. |
| [Interoperation with COM+ 1.0 Security](https://msdn.microsoft.com/en-us/library/x740xszs(v=vs.100).aspx) | Provides an overview of how COM+ 1.0 security interoperates with the .NET Framework. |
| [Key Security Concepts](https://msdn.microsoft.com/en-us/library/z164t8hs(v=vs.100).aspx) | Introduces fundamental concepts you must understand before using .NET Framework security. |
| [Security Tools (.NET Framework)](https://msdn.microsoft.com/en-us/library/dd233106(v=vs.100).aspx) | Lists and briefly describes the security tools included in the .NET Framework. |

[**Reference**](javascript:void(0)) 4.5, 4.6

[System.Security.Permissions.PrincipalPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.principalpermission(v=vs.100).aspx)

[**Related Topics**](javascript:void(0))

|  |  |
| --- | --- |
| **Title** | **Description** |
| [Principal and Identity Objects](https://msdn.microsoft.com/en-us/library/ftx85f8x(v=vs.110).aspx) | Explains how to set up and manage both Windows and generic identities and principals. |
| [Key Security Concepts](https://msdn.microsoft.com/en-us/library/z164t8hs(v=vs.110).aspx) | Introduces fundamental concepts you must understand before using .NET Framework security. |

[**Reference**](javascript:void(0))

[System.Security.Permissions.PrincipalPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.principalpermission(v=vs.110).aspx)

### Principal and Identity Objects

**.NET Framework 1.1, 2.0, 3.0, 4.0, 4.5, 4.6**

Managed code can discover the identity or the role of a principal through a [Principal](https://msdn.microsoft.com/en-us/library/system.security.principal.iprincipal(v=vs.71).aspx) object, which contains a reference to an [Identity](https://msdn.microsoft.com/en-us/library/system.security.principal.iidentity(v=vs.71).aspx) object. It might be helpful to compare identity and principal objects to familiar concepts like user and group accounts. In most network environments, user accounts represent people or programs, while group accounts represent certain categories of users and the rights they possess. Similarly, .NET Framework identity objects represent users, while roles represent memberships and security contexts. In the .NET Framework, the principal object encapsulates both an identity object and a role. .NET Framework applications grant rights to the principal based on its identity or, more commonly, its role membership.

**Identity Objects**

The identity object encapsulates information about the user or entity being validated. At their most basic level, identity objects contain a name and an authentication type. The name can either be a user's name or the name of a Windows account, while the authentication type can be either a supported logon protocol, such as Kerberos V5, or a custom value. The .NET Framework defines a [GenericIdentity](https://msdn.microsoft.com/en-us/library/system.security.principal.genericidentity(v=vs.71).aspx) object that can be used for most custom logon scenarios and a more specialized [WindowsIdentity](https://msdn.microsoft.com/en-us/library/system.security.principal.windowsidentity(v=vs.71).aspx) object that can be used when you want your application to rely on Windows authentication. Additionally, you can define your own identity class that encapsulates custom user information.

The [IIdentity](https://msdn.microsoft.com/en-us/library/system.security.principal.iidentity(v=vs.71).aspx) interface defines properties for accessing a name and an authentication type, such as Kerberos V5 or NTLM. All **Identity** classes implement the **IIdentity** interface. There is no required relationship between an **Identity** object and the Windows NT process token under which a thread is currently executing. However, if the **Identity** object is a **WindowsIdentity** object, the identity is assumed to represent a Windows NT security token.

**Principal Objects**

The principal object represents the security context under which code is running. Applications that implement role-based security grant rights based on the role associated with a principal object. Similar to identity objects, the .NET Framework provides a [GenericPrincipal](https://msdn.microsoft.com/en-us/library/system.security.principal.genericprincipal(v=vs.71).aspx) object and a[WindowsPrincipal](https://msdn.microsoft.com/en-us/library/system.security.principal.windowsprincipal(v=vs.71).aspx) object. You can also define your own custom principal classes.

The [IPrincipal](https://msdn.microsoft.com/en-us/library/system.security.principal.iprincipal(v=vs.71).aspx) interface defines a property for accessing an associated **Identity** object as well as a method for determining whether the user identified by the **Principal** object is a member of a given role. All **Principal** classes implement the **IPrincipal** interface as well as any additional properties and methods that are necessary. For example, the common language runtime provides the **WindowsPrincipal** class, which implements additional functionality for mapping Windows NT or Windows 2000 group membership to roles.

A **Principal** object is bound to a call context ([CallContext](https://msdn.microsoft.com/en-us/library/system.runtime.remoting.messaging.callcontext(v=vs.71).aspx)) object within an application domain ([AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.71).aspx)). A default call context is always created with each new **AppDomain**, so there is always a call context available to accept the **Principal** object. When a new thread is created, a**CallContext** object is also created for the thread. The **Principal** object reference is automatically copied from the creating thread to the new thread's **CallContext**. If the runtime cannot determine which **Principal** object belongs to the creator of the thread, it follows the default policy for **Principal** and **Identity** object creation.

A configurable application domain-specific policy defines the rules for deciding what type of **Principal** object to associate with a new application domain. Where security policy permits, the runtime can create **Principal** and **Identity** objects that reflect the operating system token associated with the current thread of execution. By default, the runtime uses **Principal** and **Identity** objects that represent unauthenticated users. The runtime does not create these default **Principal** and **Identity** objects until the code attempts to access them.

Trusted code that creates an application domain can set the application domain policy that controls construction of the default **Principal** and**Identity** objects. This application domain-specific policy applies to all execution threads in that application domain. An unmanaged, trusted host inherently has the ability to set this policy, but managed code that sets this policy must have the[System.Security.Permissions.SecurityPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermission(v=vs.71).aspx) for controlling domain policy.

When transmitting a **Principal** object across application domains but within the same process (and therefore on the same computer), the remoting infrastructure copies a reference to the **Principal** object associated with the caller's context to the callee's context.

#### Creating WindowsIdentity and WindowsPrincipal Objects

**.NET Framework 1.1**

The [WindowsIdentity](https://msdn.microsoft.com/en-us/library/system.security.principal.windowsidentity(v=vs.71).aspx) object encapsulates information about Windows accounts. You use the **WindowsIdentity** object if you want to make authorization decisions based on a user's Windows account information. For example, using **WindowsIdentity** and [WindowsPrincipal](https://msdn.microsoft.com/en-us/library/system.security.principal.windowsprincipal(v=vs.71).aspx) objects, you can write an application that requires all users to be currently validated by a Windows NT or Windows 2000 domain. You can also allow certain domain accounts to access your application while denying access to others.

There are two ways to create **WindowsPrincipal** objects, depending on whether code must repeatedly perform role-based validation or only needs to perform it once.

**Creating WindowsPrincipal Objects for Repeated Validation**

If code must repeatedly perform role-based validation, the following method produces less overhead.

1. First, call the **SetPrincipalPolicy** method on the [System.AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.71).aspx) object, passing it a [PrincipalPolicy](https://msdn.microsoft.com/en-us/library/system.security.principal.principalpolicy(v=vs.71).aspx) enumeration value that indicates what the new policy should be. Supported values are **NoPrincipal**, **UnauthenticatedPrincipal**, and **WindowsPrincipal**. The following code demonstrates this call.

C#

AppDomain.CurrentDomain.SetPrincipalPolicy(PrincipalPolicy.WindowsPrincipal);

[Visual Basic]

AppDomain.CurrentDomain.SetPrincipalPolicy(PrincipalPolicy.WindowsPrincipal)

1. With the policy set, use the [Thread.CurrentPrincipal](https://msdn.microsoft.com/en-us/library/075dt2hk(v=vs.71).aspx) property to retrieve the principal that encapsulates the current Windows user. The following code initializes a new **WindowsPrincipal** object to the value of the principal associated with the current thread.

C#

WindowsPrincipal MyPrincipal = (WindowsPrincipal) Thread.CurrentPrincipal

[Visual Basic]

Dim MyPrincipal As WindowsPrincipal = CType(Thread.CurrentPrincipal, WindowsPrincipal)

**Creating WindowsPrincipal Objects for One Validation**

When code only needs to make role-based validations once, you can create a **WindowsPrincipal** object by performing the following tasks.

1. Initialize a new instance of the **WindowsIdentity** class by calling the [WindowsIdentity.GetCurrent](https://msdn.microsoft.com/en-us/library/system.security.principal.windowsidentity.getcurrent(v=vs.71).aspx) method, which queries the current Windows account and places information about that account into the newly created identity object. The following code creates a new instance of the class and initializes it to the current authenticated user.

C#

WindowsIdentity MyIdentity = WindowsIdentity.GetCurrent();

[Visual Basic]

Dim MyIdentity As WindowsIdentity = WindowsIdentity.GetCurrent()

1. Create a new instance of the **Principal** class and pass it the value of a **WindowsIdentity**object. The following code demonstrates the creation of a new **WindowsPrincipal**object initialized with the previously created **WindowsIdentity** object.

C#

WindowsPrincipal MyPrincipal = new WindowsPrincipal(MyIdentity);

[Visual Basic]

Dim MyPrincipal As New WindowsPrincipal(MyIdentity)

1. When the principal object has been created, you can use one of several methods to validate it. For more information, see [Role-Based Security Checks](https://msdn.microsoft.com/en-us/library/k8b3sz1a(v=vs.71).aspx).

The following code example creates a **WindowsIdentity** object and a **WindowsPrincipal**object and displays the information to the console. You can use this code to query the values of the **WindowsIdentity** and **WindowsPrincipal**that are produced by your network environment.

C#

using System;

using System.Threading;

using System.Security.Principal;

public class Class1

{

public static int Main(string[] args)

{

//Get the current identity and put it into an identity object.

WindowsIdentity MyIdentity = WindowsIdentity.GetCurrent();

//Put the previous identity into a principal object.

WindowsPrincipal MyPrincipal = new WindowsPrincipal(MyIdentity);

//Principal values.

string Name = MyPrincipal.Identity.Name;

string Type = MyPrincipal.Identity.AuthenticationType;

string Auth = MyPrincipal.Identity.IsAuthenticated.ToString();

//Identity values.

string IdentName = MyIdentity.Name;

string IdentType = MyIdentity.AuthenticationType;

string IdentIsAuth = MyIdentity.IsAuthenticated.ToString();

string ISAnon = MyIdentity.IsAnonymous.ToString();

string IsG = MyIdentity.IsGuest.ToString();

string IsSys = MyIdentity.IsSystem.ToString();

string Token = MyIdentity.Token.ToString();

//Print the values.

Console.WriteLine("Principal Values for current thread:");

Console.WriteLine("\n\nPrincipal Name: {0}", Name);

Console.WriteLine("Principal Type: {0}", Type);

Console.WriteLine("Principal IsAuthenticated: {0}", Auth);

Console.WriteLine("\n\nIdentity Values for current thread:");

Console.WriteLine("Identity Name: {0}", IdentName);

Console.WriteLine("Identity Type: {0}", IdentType);

Console.WriteLine("Identity IsAuthenticated: {0}", IdentIsAuth);

Console.WriteLine("\n\nIdentity IsAnonymous: {0}", ISAnon);

Console.WriteLine("Identity IsGuest: {0}", IsG);

Console.WriteLine("Identity IsSystem: {0}", IsSys);

Console.WriteLine("Identity Token: {0}", Token);

return 0;

}

}

[Visual Basic]

Imports System

Imports System.Threading

Imports System.Security.Principal

Imports Microsoft.VisualBasic

Public Class Class1

Public Shared Sub Main()

'Get the current identity and put it into an identity object.

Dim MyIdentity As WindowsIdentity = WindowsIdentity.GetCurrent()

'Put the previous identity into a principal object.

Dim MyPrincipal As New WindowsPrincipal(MyIdentity)

'Principal values.

Dim PrincipalName As String = MyPrincipal.Identity.Name

Dim PrincipalType As String = MyPrincipal.Identity.AuthenticationType

Dim PrincipalAuth As String = MyPrincipal.Identity.IsAuthenticated.ToString()

'Identity values.

Dim IdentName As String = MyIdentity.Name

Dim IdentType As String = MyIdentity.AuthenticationType

Dim IdentIsAuth As String = MyIdentity.IsAuthenticated.ToString()

Dim ISAnon As String = MyIdentity.IsAnonymous.ToString()

Dim IsG As String = MyIdentity.IsGuest.ToString()

Dim IsSys As String = MyIdentity.IsSystem.ToString()

Dim Token As String = MyIdentity.Token.ToString()

'Print the values.

Console.WriteLine("Principal Values for current thread:")

Console.WriteLine(ControlChars.CrLf + ControlChars.CrLf + "Principal Name: {0}", PrincipalName)

Console.WriteLine("Principal Type: {0}", PrincipalType)

Console.WriteLine("Principal IsAuthenticated: {0}", PrincipalAuth)

Console.WriteLine(ControlChars.CrLf + ControlChars.CrLf + "Identity Values for current thread:")

Console.WriteLine("Identity Name: {0}", IdentName)

Console.WriteLine("Identity Type: {0}", IdentType)

Console.WriteLine("Identity IsAuthenticated: {0}", IdentIsAuth)

Console.WriteLine(ControlChars.CrLf + ControlChars.CrLf + "Identity IsAnonymous: {0}", ISAnon)

Console.WriteLine("Identity IsGuest: {0}", IsG)

Console.WriteLine("Identity IsSystem: {0}", IsSys)

Console.WriteLine("Identity Token: {0}", Token)

End Sub

End Class

When compiled and executed, the previous code displays output similar to the following (the actual values will vary from one network environment to another). Note that the **Name**, **Type**, and **IsAuthenticated** values will always be the same for the **WindowsPrincipal** and**WindowsIdentity** objects.

Principal Values for current thread:

Principal Name: MYDOMAIN\myuseraccount

Principal Type: NTLM

Principal IsAuthenticated: True

Identity Values for current thread:

Identity Name: MYDOMAIN\myuseraccount

Identity Type: NTLM

Identity IsAuthenticated: True

Identity IsAnonymous: False

Identity IsGuest: False

Identity IsSystem: False

Identity Token: 276

#### How to: Create a WindowsPrincipal Object

**.NET Framework 2.0, 3.0, 3.5, 4.5, 4.6**

There are two ways to create a [WindowsPrincipal](https://msdn.microsoft.com/en-us/library/system.security.principal.windowsprincipal(v=vs.80).aspx) object, depending on whether code must repeatedly perform role-based validation or must perform it only once.

If code must repeatedly perform role-based validation, the first of the following procedures produces less overhead. When code needs to make role-based validations only once, you can create a **WindowsPrincipal** object by using the second of the following procedures.

**To create a WindowsPrincipal object for repeated validation**

1. Call the [SetPrincipalPolicy](https://msdn.microsoft.com/en-us/library/system.appdomain.setprincipalpolicy(v=vs.80).aspx) method on the [AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.80).aspx) object that is returned by the static [System.AppDomain.CurrentDomain](https://msdn.microsoft.com/en-us/library/system.appdomain.currentdomain(v=vs.80).aspx)property, passing the method a [PrincipalPolicy](https://msdn.microsoft.com/en-us/library/system.security.principal.principalpolicy(v=vs.80).aspx) enumeration value that indicates what the new policy should be. Supported values are[NoPrincipal](https://msdn.microsoft.com/en-us/library/system.security.principal.principalpolicy(v=vs.80).aspx), [UnauthenticatedPrincipal](https://msdn.microsoft.com/en-us/library/system.security.principal.principalpolicy(v=vs.80).aspx), and [WindowsPrincipal](https://msdn.microsoft.com/en-us/library/system.security.principal.principalpolicy(v=vs.80).aspx). The following code demonstrates this method call.

C#

AppDomain.CurrentDomain.SetPrincipalPolicy(

PrincipalPolicy.WindowsPrincipal);

VB

AppDomain.CurrentDomain.SetPrincipalPolicy( \_

PrincipalPolicy.WindowsPrincipal)

1. With the policy set, use the static [System.Threading.Thread.CurrentPrincipal](https://msdn.microsoft.com/en-us/library/system.threading.thread.currentprincipal(v=vs.80).aspx) property to retrieve the principal that encapsulates the current Windows user. Because the property return type is [IPrincipal](https://msdn.microsoft.com/en-us/library/system.security.principal.iprincipal(v=vs.80).aspx), you must cast the result to a **WindowsPrincipal** type. The following code initializes a new **WindowsPrincipal** object to the value of the principal associated with the current thread.

C#

WindowsPrincipal MyPrincipal =

(WindowsPrincipal) Thread.CurrentPrincipal;

VB

Dim MyPrincipal As WindowsPrincipal = \_

CType(Thread.CurrentPrincipal, WindowsPrincipal)

**To create a WindowsPrincipal object for a single validation**

1. Initialize a new [WindowsIdentity](https://msdn.microsoft.com/en-us/library/system.security.principal.windowsidentity(v=vs.80).aspx) object by calling the static [System.Security.Principal.WindowsIdentity.GetCurrent](https://msdn.microsoft.com/en-us/library/sfs49sw0(v=vs.80).aspx) method, which queries the current Windows account and places information about that account into the newly created identity object. The following code creates a new **WindowsIdentity** object and initializes it to the current authenticated user.

C#

WindowsIdentity MyIdentity = WindowsIdentity.GetCurrent();

VB

Dim MyIdentity As WindowsIdentity = WindowsIdentity.GetCurrent()

1. Create a new **WindowsPrincipal** object and pass it the value of the **WindowsIdentity** object created in the preceding step.

C#

WindowsPrincipal MyPrincipal = new WindowsPrincipal(MyIdentity);

VB

Dim MyPrincipal As New WindowsPrincipal(MyIdentity)

1. When the principal object has been created, you can use one of several methods to validate it. For more information, see [Role-Based Security Checks](https://msdn.microsoft.com/en-us/library/k8b3sz1a(v=vs.80).aspx).

#### Creating GenericPrincipal and GenericIdentity Objects

**.NET Framework 1.1**

**The newer**

#### How to: Create GenericPrincipal and GenericIdentity Objects

**.NET Framework 2.0, 3.0, 3.5, 4, 4.5, 4.6**

You can use the [GenericIdentity](https://msdn.microsoft.com/en-us/library/system.security.principal.genericidentity(v=vs.71).aspx) class in conjunction with the [GenericPrincipal](https://msdn.microsoft.com/en-us/library/system.security.principal.genericprincipal(v=vs.71).aspx) class to create an authorization scheme that exists independent of a Windows NT or Windows 2000 domain.

Perform the following tasks to create an instance of the **GenericPrincipal** class.

1. Create a new instance of the identity class and initialize it with the name you want it to hold. The following code creates a new**GenericIdentity** object and initializes it with the name MyUser.

C#

GenericIdentity MyIdentity = new GenericIdentity("MyUser");

[Visual Basic]

Dim MyIdentity As New GenericIdentity("MyUser")

1. Next, create a new instance of the **GenericPrincipal** class and initialize it with the previously created **GenericIdentity** object and an array of strings that represent the roles that you want associated with this principal. The following code example specifies an array of strings that represent an administrator role and a user role. The **GenericPrincipal** is then initialized with the previous **GenericIdentity**and the string array.

C#

String[] MyStringArray = {"Manager", "Teller"};

GenericPrincipal MyPrincipal = new GenericPrincipal(MyIdentity, MyStringArray);

[Visual Basic]

Dim MyStringArray As String() = {"Manager", "Teller"}

DIm MyPrincipal As New GenericPrincipal(MyIdentity, MyStringArray)

1. Finally, use the following code to attach the principal to the current thread. This is valuable in situations where the principal must be validated several times, it must be validated by other code running in your application, or it must be validated by a [PrincipalPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.principalpermission(v=vs.71).aspx)object. You can still perform role-based validation on the principal object without attaching it to the thread. For more information, see[Replacing a Principal Object](https://msdn.microsoft.com/en-us/library/w2zhxceh(v=vs.71).aspx).

C#

Thread.CurrentPrincipal = MyPrincipal;

[Visual Basic]

Thread.CurrentPrincipal = MyPrincipal

The following code example demonstrates how to create an instance of a **GenericPrincipal** and a **GenericIdentity**. This code displays the values of these classes to the console.

C#

using System;

using System.Security.Principal;

using System.Threading;

public class Class1

{

public static int Main(string[] args)

{

//Create generic identity.

GenericIdentity MyIdentity = new GenericIdentity("MyIdentity");

//Create generic principal.

String[] MyStringArray = {"Manager", "Teller"};

GenericPrincipal MyPrincipal = new GenericPrincipal(MyIdentity, MyStringArray);

//Attach the principal to the current thread.

//This is not required unless repeated validation must occur,

//other code in your application must validate, or the

// PrincipalPermisson object is used.

Thread.CurrentPrincipal = MyPrincipal;

//Print values to the console.

String Name = MyPrincipal.Identity.Name;

bool Auth = MyPrincipal.Identity.IsAuthenticated;

bool IsInRole = MyPrincipal.IsInRole("Manager");

Console.WriteLine("The Name is: {0}", Name);

Console.WriteLine("The IsAuthenticated is: {0}", Auth);

Console.WriteLine("Is this a Manager? {0}", IsInRole);

return 0;

}

}

[Visual Basic]

Imports System

Imports System.Security.Principal

Imports System.Threading

Public Class Class1

Public Shared Sub Main()

'Create generic identity.

Dim MyIdentity As New GenericIdentity("MyIdentity")

'Create generic principal.

Dim MyStringArray As String() = {"Manager", "Teller"}

Dim MyPrincipal As New GenericPrincipal(MyIdentity, MyStringArray)

'Attach the principal to the current thread.

'This is not required unless repeated validation must occur,

'other code in your application must validate, or the

' PrincipalPermisson object is used.

Thread.CurrentPrincipal = MyPrincipal

'Print values to the console.

Dim Name As String = MyPrincipal.Identity.Name

Dim Auth As Boolean = MyPrincipal.Identity.IsAuthenticated

Dim IsInRole As Boolean = MyPrincipal.IsInRole("Manager")

Console.WriteLine("The Name is: {0}", Name)

Console.WriteLine("The IsAuthenticated is: {0}", Auth)

Console.WriteLine("Is this a Manager? {0}", IsInRole)

End Sub

End Class

When executed, the application displays output similar to the following.

The Name is: MyIdentity

The IsAuthenticated is: True

Is this a Manager? True

#### Replacing a Principal Object

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4, 4.5, 4.6**

Applications that provide authentication services must be able to replace the [Principal](https://msdn.microsoft.com/en-us/library/system.security.principal.iprincipal(v=vs.71).aspx) object for a given thread. Further, the security system must help protect the ability to replace **Principal** objects because a maliciously attached, incorrect **Principal** compromises the security of your application by claiming an untrue identity or role. Therefore, applications that require the ability to replace **Principal** objects must be granted the [System.Security.Permissions.SecurityPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermission(v=vs.71).aspx) object for principal control. (Note that this permission is not required for performing role-based security checks or for creating **Principal** objects.)

The current **Principal** object can be replaced by performing the following tasks:

1. Create the replacement **Principal** object and associated **Identity** object, usually after performing authentication.
2. Create a new **System.Security.Permissions.SecurityPermission** object, passing the [SecurityPermissionAttribute.ControlPrincipal](https://msdn.microsoft.com/en-us/library/0htb2e7e(v=vs.71).aspx)enumeration value to the constructor. (Note that this permission is not necessary for trusted code that simply replaces the permission object, but is necessary when a trusted host is providing the principal to semi-trusted code for role-based verification.)
3. Attach the new **Principal** object to the call context, as shown in the following code.

C#

Thread.CurrentPrincipal = principalObject;

[Visual Basic]

Thread.CurrentPrincipal = principalObject

[**Example**](javascript:void(0)) (3.5)

The following example shows how to create a generic principal object and use it to set the principal of a thread.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/w2zhxceh(v=vs.90).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

using System;

using System.Threading;

using System.Security.Permissions;

using System.Security.Principal;

class SecurityPrincipalDemo

{

public static void Main()

{

// Retrieve a GenericPrincipal that is based on the current user's

// WindowsIdentity.

GenericPrincipal genericPrincipal = GetGenericPrincipal();

// Retrieve the generic identity of the GenericPrincipal object.

GenericIdentity principalIdentity =

(GenericIdentity)genericPrincipal.Identity;

// Display the identity name and authentication type.

if (principalIdentity.IsAuthenticated)

{

Console.WriteLine(principalIdentity.Name);

Console.WriteLine("Type:" + principalIdentity.AuthenticationType);

}

// Verify that the generic principal has been assigned the

// NetworkUser role.

if (genericPrincipal.IsInRole("NetworkUser"))

{

Console.WriteLine("User belongs to the NetworkUser role.");

}

Thread.CurrentPrincipal = genericPrincipal;

}

// Create a generic principal based on values from the current

// WindowsIdentity.

private static GenericPrincipal GetGenericPrincipal()

{

// Use values from the current WindowsIdentity to construct

// a set of GenericPrincipal roles.

WindowsIdentity windowsIdentity = WindowsIdentity.GetCurrent();

string[] roles = new string[10];

if (windowsIdentity.IsAuthenticated)

{

// Add custom NetworkUser role.

roles[0] = "NetworkUser";

}

if (windowsIdentity.IsGuest)

{

// Add custom GuestUser role.

roles[1] = "GuestUser";

}

if (windowsIdentity.IsSystem)

{

// Add custom SystemUser role.

roles[2] = "SystemUser";

}

// Construct a GenericIdentity object based on the current Windows

// identity name and authentication type.

string authenticationType = windowsIdentity.AuthenticationType;

string userName = windowsIdentity.Name;

GenericIdentity genericIdentity =

new GenericIdentity(userName, authenticationType);

// Construct a GenericPrincipal object based on the generic identity

// and custom roles for the user.

GenericPrincipal genericPrincipal =

new GenericPrincipal(genericIdentity, roles);

return genericPrincipal;

}

}

#### Impersonating and Reverting

**.NET Framework 1.1, 2.0, 3.0, 4.0, 4.5, 4.6**

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

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

1. Retrieve an account token for a particular user by making a call to the unmanaged **LogonUser** method. This method is not in the .NET Framework base class library, but is located in the unmanaged **advapi32.dll**. Accessing methods in unmanaged code is an advanced operation and is beyond the scope of this discussion. For more information, see [Interoperating with Unmanaged Code](https://msdn.microsoft.com/en-us/library/sd10k43k(v=vs.71).aspx). For more information about the **LogonUser** method and **advapi32.dll**, see the Platform SDK documentation.
2. Create a new instance of the **WindowsIdentity** class, passing the token. The following code demonstrates this call, where hTokenrepresents a Windows token.

C#

WindowsIdentity ImpersonatedIdentity = new WindowsIdentity(hToken);

[Visual Basic]

Dim ImpersonatedIdentity As New WindowsIdentity(hToken)

1. Begin impersonation by creating a new instance of the [WindowsImpersonationContext](https://msdn.microsoft.com/en-us/library/system.security.principal.windowsimpersonationcontext(v=vs.71).aspx) class and initializing it with the[WindowsIdentity.Impersonate](https://msdn.microsoft.com/en-us/library/system.security.principal.windowsidentity.impersonate(v=vs.71).aspx) method of the initialized class, as shown in the following code.

C#

WindowsImpersonationContext MyImpersonation = ImpersonatedIdentity.Impersonate();

[Visual Basic]

WindowsImpersonationContext MyImpersonation = ImpersonatedIdentity.Impersonate()

1. When you no longer need to impersonate, call the [WindowsImpersonationContext.Undo](https://msdn.microsoft.com/en-us/library/a39ytsk3(v=vs.71).aspx) method to revert the impersonation, as shown in the following code.

C#

MyImpersonation.Undo();

[Visual Basic]

MyImpersonation.Undo()

If trusted code has already attached a [WindowsPrincipal](https://msdn.microsoft.com/en-us/library/system.security.principal.windowsprincipal(v=vs.71).aspx) object to the thread, you can call the instance method **Impersonate**, which does not take an account token. Note that this is only useful when the **WindowsPrincipal** object on the thread represents a user other than the one under which the process is currently executing. For example, you might encounter this situation using ASP.NET with Windows authentication turned on and impersonation turned off. In this case, the process is running under an account configured in Internet Information Services (IIS) while the current principal represents the Windows user that is accessing the page.

Note that neither **Impersonate** nor **Undo** changes the [Principal](https://msdn.microsoft.com/en-us/library/system.security.principal.iprincipal(v=vs.71).aspx) object associated with the current call context. Rather, impersonation and reverting change the token associated with the current operating system process..

### PrincipalPermission Objects

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4**

The role-based security model supports a permission object similar to the permission objects found in the code access security model. This object, [PrincipalPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.principalpermission(v=vs.71).aspx), represents the identity and role that a particular principal class must have to run. You can use the**PrincipalPermission** class for both imperative and declarative security checks.

To implement the **PrincipalPermission** class imperatively, create a new instance of the class and initialize it with the name and role that you want users to have to access your code. For example, the following code initializes a new instance of this object with an identity of "Joan"and a role of "Teller".

C#

String id = "Joan";

String role = "Teller";

PrincipalPermission principalPerm = new PrincipalPermission(id, role);

[Visual Basic]

Dim id As String = "Joan"

Dim role As String = "Teller"

Dim principalPerm As New PrincipalPermission(id, role)

You can create a similar permission declaratively using the [PrincipalPermissionAttribute](https://msdn.microsoft.com/en-us/library/system.security.permissions.principalpermissionattribute(v=vs.71).aspx) class. The following code declaratively initializes the identity to "Joan" and the role to "Teller".

C#

[PrincipalPermissionAttribute(SecurityAction.Demand, Name = "Joan", Role = "Teller")]

[Visual Basic]

<PrincipalPermissionAttribute(SecurityAction.Demand, Name := "Joan", Role := "Teller")>

When the security check is performed, both the specified identity and role must match for the check to succeed. However, when you create the **PrincipalPermission** object, you can pass a null identity string to indicate that the identity of the principal can be anything. Similarly, passing a null role string indicates that the principal can be a member of any role (or no roles at all). For declarative security, the same result can be achieved by omitting either property. For example, the following code uses the **PrincipalPermissionAttribute** to declaratively indicate that a principal can have any name, but must have the role of teller.

C#

[PrincipalPermissionAttribute(SecurityAction.Demand, Role = "Teller")]

[Visual Basic]

<PrincipalPermissionAttribute(SecurityAction.Demand, Role := "Teller")>

#### Combining PrincipalPermission Objects

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4.0**

In most cases, you use [PrincipalPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.principalpermission(v=vs.71).aspx) objects in the same way that you use code access security permission objects. However, some aspects of permission functionality are more useful than others. For example, you can call the **Intersect** method to intersect two **PrincipalPermission** objects, but this might not be very useful because identity and role are intersected independently to return a**PrincipalPermission** object that represents the identity and role common to the intersected objects. In many cases, the returned object contains an empty string for the identity, the role, or both, and the object matches only an unauthenticated user (identity = "") or a user that does not belong to any roles (role = "").

Subset operations determine whether one **PrincipalPermission** object is a strict subset of another **PrincipalPermission** object. Effectively, this means that **IsSubsetOf** returns true only if the identities and roles match exactly, or if a role or identity is null. Therefore, **IsSubsetOf** has limited usefulness for this permission.

On the other hand, performing a union operation on two **PrincipalPermission** objects can be useful when you want to compactly represent a set of conditions that you want to test. For example, a union operation can be used when you want to check for the presence of either one identity or another identity. The following code shows a security check that succeeds if the [Principal](https://msdn.microsoft.com/en-us/library/system.security.principal.iprincipal(v=vs.71).aspx) object represents fred or sally in the role of Administrator.

C#

String id1 = "fred";

String role1 = "Administrator";

PrincipalPermission myPrincipalPerm1 = new PrincipalPermission(id1, role1);

String id2 = "sally";

String role2 = "Administrator";

PrincipalPermission myPrincipalPerm2 = new PrincipalPermission(id2, role2);

(myPrincipalPerm1.Union(myPrincipalPerm2)).Demand();

[Visual Basic]

Dim id1 As String = "fred"

Dim role1 As String = "Administrator"

Dim myPrincipalPerm1 As New PrincipalPermission(id1, role1)

Dim id2 As String = "sally"

Dim role2 As String = "Administrator"

Dim myPrincipalPerm2 As New PrincipalPermission(id2, role2)

myPrincipalPerm1.Union(myPrincipalPerm2).Demand()

The following code shows how to specify the union of two permissions that accept any principal with the role1 and role2 values from the previous code.

C#

((new PrincipalPermission(null, role1)).Union(new PrincipalPermission(null, role2))).Demand();

[Visual Basic]

Dim pp1 As New PrincipalPermission(Nothing, role1)

Dim pp2 As New PrincipalPermission(Nothing, role2)

pp1.Union(pp2).Demand()

### Role-Based Security Checks

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4**

Once you have defined identity and principal objects, you can perform security checks against them in one of the following ways:

* Using imperative security checks.
* Using declarative security checks.
* Directly accessing the [Principal](https://msdn.microsoft.com/en-us/library/system.security.principal.iprincipal(v=vs.71).aspx) object.

Managed code can use imperative or declarative security checks to determine whether a particular principal object is a member of a known role, has a known identity, or represents a known identity acting in a role. To cause the security check to occur using imperative or declarative security, a security demandfor an appropriately constructed [PrincipalPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.principalpermission(v=vs.71).aspx) object must be made. During the security check, the common language runtime examines the caller's principal object to determine whether its identity and role match those represented by the**PrincipalPermission** being demanded. If the principal object does not match, a [SecurityException](https://msdn.microsoft.com/en-us/library/system.security.securityexception(v=vs.71).aspx) is thrown. (Only the principal object of the current thread is examined; the **PrincipalPermission**class does not cause a stack walk as with code access permission.)

Additionally, you can access the values of the principal object directly and perform checks without a **PrincipalPermission** object. In this case, you simply read the values of the current thread's principal or use the **IsInRole** method perform authorization.

#### Performing Imperative Security Checks

**.NET Framework 1.1, 2.0**

**Newer title**

#### How to: Perform Imperative Security Checks

**.NET Framework 3.0, 3.5, 4**

For an imperative demand, you can call the **Demand** method of the [PrincipalPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.principalpermission(v=vs.71).aspx) object to determine whether the current principal object represents the specified identity, role, or both. Assuming a properly constructed **PrincipalPermission** object calledMyPrincipalPermission, an imperative demand can be called with the following code.

C#

MyPrincipalPermission.Demand();

[Visual Basic]

MyPrincipalPermission.Demand()

The following code example uses an imperative check to ensure that a [GenericPrincipal](https://msdn.microsoft.com/en-us/library/system.security.principal.genericprincipal(v=vs.71).aspx) matches the **PrincipalPermission** object. An imperative check is useful when many methods or other assemblies in the application domain must make role-based determinations. While this example is extremely simple, it illustrates the behavior associated with a role-based demand.

C#

using System;

using System.Security.Permissions;

using System.Security.Principal;

using System.Security;

using System.Threading;

using System.Security.Cryptography;

public class MainClass

{

public static int Main(string[] args)

{

Console.WriteLine("Enter '1' to use the proper identity or any other character to use the improper identity.");

if(Console.ReadLine() == "1")

{

//Create a generic identity.

GenericIdentity MyIdentity = new GenericIdentity( "MyUser");

//Create a generic principal.

String[] MyString = {"Administrator", "User"};

GenericPrincipal MyPrincipal = new GenericPrincipal(MyIdentity, MyString);

Thread.CurrentPrincipal = MyPrincipal;

}

PrivateInfo();

return 0;

}

public static void PrivateInfo()

{

try

{

//Create a PrincipalPermission object.

PrincipalPermission MyPermission = new PrincipalPermission("MyUser", "Administrator");

//Demand this permission.

MyPermission.Demand();

//Print secret data.

Console.WriteLine("\n\nYou have access to the private data!");

}

catch(SecurityException e)

{

Console.WriteLine(e.Message);

}

}

}

[Visual Basic]

Imports System

Imports System.Security.Permissions

Imports System.Security.Principal

Imports System.Security

Imports System.Threading

Imports System.Security.Cryptography

\_

Public Class MainClass

Public Overloads Shared Function Main() As Integer

Console.WriteLine("Enter '1' to use the proper identity or any other character to use the improper identity.")

If Console.ReadLine() = "1" Then

'Create a generic identity.

Dim MyIdentity As New GenericIdentity("MyUser")

'Create a generic principal.

Dim MyString As [String]() = {"Administrator", "User"}

Dim MyPrincipal As New GenericPrincipal(MyIdentity, MyString)

Thread.CurrentPrincipal = MyPrincipal

End If

PrivateInfo()

Return 0

End Function

Public Shared Sub PrivateInfo()

Try

'Create a PrincipalPermission object.

Dim MyPermission As New PrincipalPermission("MyUser", "Administrator")

'Demand this permission.

MyPermission.Demand()

'Print secret data.

Console.WriteLine(ControlChars.Cr + ControlChars.Cr + "You have access to the private data!")

Catch e As SecurityException

Console.WriteLine(e.Message)

End Try

End Sub

End Class

If the user types 1, the principal and identity objects needed to access the PrivateInfo method are created. If the user types any other character, no principal and identity objects are created and a security exception is thrown when the PrivateInfo method is called. If the current thread is associated with a principal that has the name MyUser and the Administrator role, the following message appears.

You have access to the private data!

#### Performing Declarative Security Checks

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4**

Declarative demands for [PrincipalPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.principalpermission(v=vs.71).aspx) work the same way as declarative demands for code access permissions. Demands can be placed at the class level as well as on individual methods, properties, or events. If a declarative demand is placed at both the class and member level, the declarative demand on the member overrides (or replaces) the demand at the class level.

The following code example shows a modified version of the PrivateInfo method from the previous section's example. This version uses declarative security. The [PrincipalPermissionAttribute](https://msdn.microsoft.com/en-us/library/system.security.permissions.principalpermissionattribute(v=vs.71).aspx) defines the principal that the current thread must have to invoke the method. Simply pass [SecurityAction.Demand](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.71).aspx) with the name and role that you require.

C#

[PrincipalPermissionAttribute(SecurityAction.Demand, Name = "MyUser", Role = "User")]

public static void PrivateInfo()

{

//Print secret data.

Console.WriteLine("\n\nYou have access to the private data!");

}

[Visual Basic]

Public Shared Sub \_

<PrincipalPermissionAttribute(SecurityAction.Demand, Name := "MyUser", Role := "User")> \_

PrivateInfo()

'Print secret data.

Console.WriteLine(ControlChars.CrLf + "You have access to the private data!")

End Sub

This method throws a security exception if the current thread does not contain the proper principal. If the the user enters 1, the PrivateInfomethod is invoked and the following message displays to the console.

You have access to the private data!

#### Directly Accessing a Principal Object

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4**

Although using imperative and declarative demands to invoke role-based security checks is the primary mechanism for checking and enforcing identity and role membership, there might be cases where you want to access the [Principal](https://msdn.microsoft.com/en-us/library/system.security.principal.iprincipal(v=vs.71).aspx) object and its associated [Identity](https://msdn.microsoft.com/en-us/library/system.security.principal.iidentity(v=vs.71).aspx) object directly to do authorization tasks without creating permission objects. For example, you might not want to use declarative or imperative demands if you do not want a thrown exception to be the default behavior for validation failure. In this case, you can use the static**CurrentPrincipal** property on the [System.Threading.Thread](https://msdn.microsoft.com/en-us/library/system.threading.thread(v=vs.71).aspx) class to access the **Principal** object and call its methods.

After obtaining the principal object, you can use conditional statements to control access to your code based on the principal name as shown in the following code example.

C#

WindowsPrincipal MyPrincipal = (WindowsPrincipal) Thread.CurrentPrincipal;

if (MyPrincipal.Identity.Name == "fred")

// Permit access to some code.

[Visual Basic]

Dim MyPrincipal As WindowsPrincipal = \_

CType(Thread.CurrentPrincipal, WindowsPrincipal)

If (MyPrincipal.Identity.Name = "fred") Then

' Permit access to some code.

End If

You can also programmatically check role membership by calling the **IsInRole** method on the current **Principal** object as shown in the following code example.

C#

WindowsPrincipal MyPrincipal = (Thread.CurrentPrincipal as WindowsPrincipal);

if (MyPrincipal.IsInRole("Administrator")) {

// Permit access to some code.

}

[Visual Basic]

WindowsPrincipal MyPrincipal = (Thread.CurrentPrincipal as WindowsPrincipal);

If (MyPrincipal.IsInRole("Administrator")) Then

// Permit access to some code.

End If

You might use this technique when you want to access behaviors that are specific to an application-defined **Principal** object. However, in most cases, you use the [PrincipalPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.principalpermission(v=vs.71).aspx) class to control access to your code based on identity or role membership.

The following code example creates a [WindowsPrincipal](https://msdn.microsoft.com/en-us/library/system.security.principal.windowsprincipal(v=vs.71).aspx) object and a [WindowsIdentity](https://msdn.microsoft.com/en-us/library/system.security.principal.windowsidentity(v=vs.71).aspx)object, sets them to the current user, and makes a security decision based on the value of the **Principal**. It does not use a **PrincipalPermission** object imperatively or declaratively, but makes an access decision based on the values of the principal object instead.

C#

using System;

using System.Security.Permissions;

using System.Security.Policy;

using System.Security.Principal;

using System.Threading;

public class Class1

{

public static int Main(string[] args)

{

//Set principal policy so that you have permission to get current user information.

AppDomain.CurrentDomain.SetPrincipalPolicy(PrincipalPolicy.WindowsPrincipal);

//Get the current principal and put it into a principal object.

WindowsPrincipal MyPrincipal = (Thread.CurrentPrincipal as WindowsPrincipal);

//Check the name and see if the user is authenticated.

if(MyPrincipal.Identity.Name.Equals(@"MYDOMAIN\myuser") && MyPrincipal.Identity.IsAuthenticated.Equals(true))

{

Console.WriteLine("Hello {0}, you are authenticated!", MyPrincipal.Identity.Name.ToString());

}

else

{

Console.WriteLine("Go away! You are not authorized!");

}

return 0;

}

}

C# 3.5 updated code example

using System;

using System.Security.Permissions;

using System.Security.Policy;

using System.Security.Principal;

using System.Threading;

public class Class1

{

public static int Main(string[] args)

{

// Set principal policy to get a WindowsPrincipal

// as the current principal so you have permission to get

// current user information.

AppDomain.CurrentDomain.SetPrincipalPolicy(

PrincipalPolicy.WindowsPrincipal);

// Get the current principal and put it into a principal object.

WindowsPrincipal myPrincipal = (Thread.CurrentPrincipal

as WindowsPrincipal);

// Check the name and see if the user is authenticated.

if (myPrincipal.Identity.Name.Equals(@"MYDOMAIN\myuser")

&& myPrincipal.Identity.IsAuthenticated.Equals(true))

{

Console.WriteLine("Hello {0}, you are authenticated!",

myPrincipal.Identity.Name.ToString());

}

else

{

Console.WriteLine("Go away! You are not authorized!");

}

// Use IsInRole to determine the role of the current user.

Array wbirFields = Enum.GetValues(typeof(WindowsBuiltInRole));

foreach (object roleName in wbirFields)

{

try

{

Console.WriteLine("{0}? {1}.", roleName,

myPrincipal.IsInRole((WindowsBuiltInRole)roleName));

}

catch (Exception)

{

Console.WriteLine("{0}: Could not obtain role for this RID.",

roleName);

}

}

return 0;

}

}

[Visual Basic]

Imports System

Imports System.Security.Permissions

Imports System.Security.Policy

Imports System.Security.Principal

Imports System.Threading

Public Class Class1

Public Shared Sub Main()

'Set principal policy so that you have permission to get current user information.

AppDomain.CurrentDomain.SetPrincipalPolicy(PrincipalPolicy.WindowsPrincipal)

'Get the current principal and put it into a principal object.

Dim MyPrincipal As WindowsPrincipal = CType(Thread.CurrentPrincipal, WindowsPrincipal)

'Check the name and see if the user is authenticated.

If (MyPrincipal.Identity.Name.Equals("MYDOMAIN\myuser") And MyPrincipal.Identity.IsAuthenticated) Then

Console.WriteLine("Hello {0}, you are authenticated!", MyPrincipal.Identity.Name.ToString())

Else

Console.WriteLine("Go away! You are not authorized!")

End If

End Sub

End Class

If the current user is MYDOMAIN\myuser, this program displays the following message to the console.

Hello MYDOMAIN\myuser, you are authenticated!

However, if any other user is the current user, the program displays the following message.

Go away! You are not authorized!

The value in MyPrincipal.Identity.Name shows the domain and user name that represents the authorized account. Notice that in C# the string "MYDOMAIN\myuser" is prefixed with the at sign (@) so that the backslash is not interpreted as an escape character. Although the previous example uses a **WindowsIdentity** object, you can easily produce similar code using a generic object. Simply create an instance of the generic object, pass it the values you want, and later check the object for those values.

### Interoperation with COM+ 1.0 Security

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4**

You can extend an existing COM+ 1.0 application with new .NET Framework managed components. The COM+ 1.0 security context is still managed by COM+ 1.0, and the COM+ 1.0 administrative user interface is used to configure the application. Basically, from a COM+ 1.0 application, .NET Framework objects look like COM+ 1.0 objects.

To make .NET Framework objects visible to COM+ 1.0 security services, you must run tools (such as Tlbexp.exe) supplied by the .NET Framework SDK to generate type libraries for the public interfaces and register the objects so that COM+ 1.0 can locate them. COM+ 1.0 administrative facilities must be used to configure roles and other role-based security behavior.

There are some limitations to COM+ 1.0 security interoperability. COM+ 1.0 security properties are not propagated across process or machine boundaries or to newly created execution threads within managed code. COM+ 1.0 security services can only be used by managed code on Windows 2000 systems.

The .NET Framework provides several managed wrappers in the [System.EnterpriseServices](https://msdn.microsoft.com/en-us/library/system.enterpriseservices(v=vs.71).aspx) namespace that allow access to COM+ 1.0 security features.

## Cryptographic Services

**.NET Framework 1.1, 2.0**

Public networks such as the Internet do not provide a means of secure communication between entities. Communication over such networks is susceptible to being read or even modified by unauthorized third parties. In addition to file encryption and encryption on a local disk, cryptography helps you create encrypted channels of communication over otherwise insecure channels, providing data integrity and authentication.

The classes in the .NET Framework cryptography namespace manage many details of cryptography for you. Some are wrappers for the unmanaged Microsoft CryptoAPI, while others are purely managed implementations. You do not need to be an expert in cryptography to use these classes. When you create a new instance of one of the encryption algorithm classes, keys are autogenerated for ease of use, and default properties are as safe and secure as possible.

**In This Section**

[Cryptography Overview](https://msdn.microsoft.com/en-us/library/92f9ye3s(v=vs.71).aspx)

Provides an introduction to key concepts in cryptography, such as asymmetric cryptography, symmetric cryptography, digital signatures, and cryptographic hashes.

[.NET Framework Cryptography Model](https://msdn.microsoft.com/en-us/library/0ss79b2x(v=vs.71).aspx)

Describes how cryptography is implemented in the base class library.

[Cryptographic Tasks](https://msdn.microsoft.com/en-us/library/7yx4d854(v=vs.71).aspx)

Describes how to perform specific cryptographic tasks using the base class library.

**Related Sections**

[Securing Applications](https://msdn.microsoft.com/en-us/library/fkytk30f(v=vs.71).aspx)

Describes the entire .NET Framework security system.

[Configuring Cryptography Classes](https://msdn.microsoft.com/en-us/library/bke5we9a(v=vs.71).aspx)

Describes how to map algorithm names to cryptographic classes and map object identifiers to a cryptographic algorithm.

## Cryptographic Services

**.NET Framework 1.1, 2.0, 3.0, 3.5**

Public networks such as the Internet do not provide a means of secure communication between entities. Communication over such networks is susceptible to being read or even modified by unauthorized third parties. In addition to file encryption and encryption on a local disk, cryptography helps you create a secure means of communication over otherwise insecure channels, providing data integrity and authentication.

The classes in the .NET Framework cryptography namespace manage many details of cryptography for you. Some are wrappers for the unmanaged Microsoft CryptoAPI, while others are purely managed implementations. You do not need to be an expert in cryptography to use these classes. When you create a new instance of one of the encryption algorithm classes, keys are autogenerated for ease of use, and default properties are as safe and secure as possible.

**In This Section**

[Cryptography Overview](https://msdn.microsoft.com/en-us/library/92f9ye3s(v=vs.80).aspx)

Provides an introduction to key concepts in cryptography, such as asymmetric cryptography, symmetric cryptography, digital signatures, and cryptographic hashes.

[.NET Framework Cryptography Model](https://msdn.microsoft.com/en-us/library/0ss79b2x(v=vs.80).aspx)

Describes how cryptography is implemented in the base class library.

[Cryptographic Tasks](https://msdn.microsoft.com/en-us/library/7yx4d854(v=vs.80).aspx)

Describes how to perform specific cryptographic tasks using the base class library.

[Walkthrough: Creating a Cryptographic Application](https://msdn.microsoft.com/en-us/library/aa964697(v=vs.80).aspx)

Demonstrates basic encryption and decryption tasks.

Related Sections

[Configuring Cryptography Classes](https://msdn.microsoft.com/en-us/library/bke5we9a(v=vs.80).aspx)

Describes how to map algorithm names to cryptographic classes and map object identifiers to a cryptographic algorithm.

[About System.Security.Cryptography.Pkcs](https://msdn.microsoft.com/en-us/library/ms180945(v=vs.80).aspx)

Describes the namespace that contains the managed code implementation of the Cryptographic Message Syntax (CMS) and Public-Key Cryptography Standards #7 (PKCS #7) standards. This section is of interest to developers.

[Using System.Security.Cryptography.Pkcs](https://msdn.microsoft.com/en-us/library/ms180955(v=vs.80).aspx)

Explains how to use the [System.Security.Cryptography.Pkcs](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs(v=vs.80).aspx) namespace to program the Cryptographic Message Syntax (CMS) and Public-Key Cryptography Standards #7 (PKCS #7) standards into your application. This section is of interest to developers.

[Security in the .NET Framework](https://msdn.microsoft.com/en-us/library/fkytk30f(v=vs.80).aspx)

Describes the entire .NET Framework security system.

### Cryptography Overview

**.NET Framework 1.1, 2.0, 3.0**

Cryptography helps protect data from being viewed or modified and helps provide a secure means of communication over otherwise insecure channels. For example, data can be encrypted using a cryptographic algorithm, transmitted in an encrypted state, and later decrypted by the intended party. If a third party intercepts the encrypted data, it will be difficult to decipher.

In a typical situation where cryptography is used, two parties (Alice and Bob) communicate over an insecure channel. Alice and Bob want to ensure that their communication remains incomprehensible by anyone who might be listening. Furthermore, because Alice and Bob are in remote locations, Alice must be sure that the information she receives from Bob has not been modified by anyone during transmission. In addition, she must be sure that the information really does originate from Bob and not from someone impersonating Bob.

Cryptography is used to achieve the following goals:

* Confidentiality: To help protect a user's identity or data from being read.
* Data integrity: To help protect data from being altered.
* Authentication: To assure that data originates from a particular party.

To achieve these goals, you can use a combination of algorithms and practices known as cryptographic primitives to create a cryptographic scheme. The following table lists the cryptographic primitives and their uses.

|  |  |
| --- | --- |
| **Cryptographic primitive** | **Use** |
| Secret-key encryption (symmetric cryptography) | Performs a transformation on data, keeping the data from being read by third parties. This type of encryption uses a single shared, secret key to encrypt and decrypt data. |
| Public-key encryption (asymmetric cryptography) | Performs a transformation on data, keeping the data from being read by third parties. This type of encryption uses a public/private key pair to encrypt and decrypt data. |
| Cryptographic signing | Helps verify that data originates from a specific party by creating a digital signature that is unique to that party. This process also uses hash functions. |
| Cryptographic hashes | Maps data from any length to a fixed-length byte sequence. Hashes are statistically unique; a different two-byte sequence will not hash to the same value. |

**Secret-Key Encryption**

Secret-key encryption algorithms use a single secret key to encrypt and decrypt data. You must secure the key from access by unauthorized agents because any party that has the key can use it to decrypt data. Secret-key encryption is also referred to as symmetric encryption because the same key is used for encryption and decryption. Secret-key encryption algorithms are extremely fast (compared to public-key algorithms) and are well suited for performing cryptographic transformations on large streams of data.

Typically, secret-key algorithms, called block ciphers, are used to encrypt one block of data at a time. Block ciphers (like RC2, DES, TripleDES, and Rijndael) cryptographically transform an input block of n bytes into an output block of encrypted bytes. If you want to encrypt or decrypt a sequence of bytes, you have to do it block by block. Because n is small (n = 8 bytes for RC2, DES, and TripleDES; n = 16 [the default], n = 24, or n = 32 bytes for Rijndael), values larger than n have to be encrypted one block at a time.

The block cipher classes provided in the base class library use a chaining mode called cipher block chaining (CBC), which uses a key and an initialization vector (IV) to perform cryptographic transformations on data. For a given secret key k, a simple block cipher that does not use an initialization vector will encrypt the same input block of plain text into the same output block of cipher text. If you have duplicate blocks within your plain text stream, you will have duplicate blocks within your cipher text stream. If unauthorized users know anything about the structure of a block of your plain text, they can use that information to decipher the known cipher text block and possibly recover your key. To combat this problem, information from the previous block is mixed into the process of encrypting the next block. Thus, the output of two identical plain text blocks is different. Because this technique uses the previous block to encrypt the next block, an IV is used to encrypt the first block of data. Using this system, common message headers that might be known to an unauthorized user cannot be used to reverse engineer a key.

One way to compromise data encrypted with this type of cipher is to perform an exhaustive search of every possible key. Depending on the size of the key used to perform encryption, this type of search is extremely time consuming using even the fastest computers and is therefore unfeasible. Larger key sizes are more difficult to decipher. Although encryption does not make it theoretically impossible for an adversary to retrieve the encrypted data, it does raise the cost of doing so prohibitively. If it takes three months to perform an exhaustive search to retrieve data that is only meaningful for a few days, then the exhaustive search method is impractical.

The disadvantage of secret-key encryption is that it presumes two parties have agreed on a key and IV and communicated their values. Also, the key must be kept secret from unauthorized users. Because of these problems, secret-key encryption is often used in conjunction with public-key encryption to privately communicate the values of the key and IV.

Assuming that Alice and Bob are two parties who want to communicate over an insecure channel, they might use secret-key encryption as follows. Both Alice and Bob agree to use one particular algorithm (Rijndael, for example) with a particular key and IV. Alice composes a message and creates a network stream on which to send the message. Next she encrypts the text using the key and IV, and sends it across the Internet. She does not send the key and IV to Bob. Bob receives the encrypted text and decrypts it using the previously agreed upon key and IV. If the transmission is intercepted, the interceptor cannot recover the original message because he does not know the key or IV. In this scenario, the key must remain secret, but the IV does not need to remain secret. In a real world scenario, either Alice or Bob generates a secret key and uses public-key (asymmetric) encryption to transfer the secret (symmetric) key to the other party. For more information, see the section on Public-Key Encryption, later in this topic.

The .NET Framework provides the following classes that implement secret key encryption algorithms:

* [DESCryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.descryptoserviceprovider(v=vs.71).aspx)
* [RC2CryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rc2cryptoserviceprovider(v=vs.71).aspx)
* [RijndaelManaged](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rijndaelmanaged(v=vs.71).aspx)
* [TripleDESCryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.tripledescryptoserviceprovider(v=vs.71).aspx)

**Public-Key Encryption**

Public-key encryption uses a private key that must be kept secret from unauthorized users and a public key that can be made public to anyone. The public key and the private key are mathematically linked; data encrypted with the public key can be decrypted only with the private key, and data signed with the private key can be verified only with the public key. The public key can be made available to anyone; it is used for encrypting data to be sent to the keeper of the private key. Both keys are unique to the communication session. Public-key cryptographic algorithms are also known as asymmetric algorithms because one key is required to encrypt data while another is required to decrypt data.

Public-key cryptographic algorithms use a fixed buffer size whereas secret-key cryptographic algorithms use a variable-length buffer. Public-key algorithms cannot be used to chain data together into streams the way secret-key algorithms can because only small amounts of data can be encrypted. Therefore, asymmetric operations do not use the same streaming model as symmetric operations.

Two parties (Alice and Bob) might use public-key encryption as follows. First, Alice generates a public/private key pair. If Bob wants to send Alice an encrypted message, he asks her for her public key. Alice sends Bob her public key over an insecure network and Bob uses this key to encrypt a message. (If Bob received Alice's key over an insecure channel, such as a public network, Bob must verify with Alice that he has a correct copy of her public key.) Bob sends the encrypted message to Alice and she decrypts it using her private key.

During the transmission of Alice's public key, however, an unauthorized agent might intercept the key. Furthermore, the same agent might intercept the encrypted message from Bob. However, the agent cannot decrypt the message with the public key. The message can only be decrypted with Alice's private key, which has not been transmitted. Alice does not use her private key to encrypt a reply message to Bob, because anyone with the public key could decrypt the message. If Alice wants to send a message back to Bob, she asks Bob for his public key and encrypts her message using that public key. Bob then decrypts the message using his associated private key.

In a real world scenario, Alice and Bob use public key (asymmetric) encryption to transfer a secret (symmetric) key and use secret key encryption for the remainder of their session.

Public-key encryption has a much larger keyspace, or range of possible values for the key, and is therefore less susceptible to exhaustive attacks that try every possible key. A public key is easy to distribute because it does not have to be secured. Public-key algorithms can be used to create digital signatures to verify the identity of the sender of data. However, public-key algorithms are extremely slow (compared to secret-key algorithms) and are not designed to encrypt large amounts of data. Public-key algorithms are useful only for transferring very small amounts of data. Typically, public-key encryption is used to encrypt a key and IV to be used by a secret-key algorithm. After the key and IV are transferred, then secret-key encryption is used for the remainder of the session.

The .NET Framework provides the following classes that implement public-key encryption algorithms:

* [DSACryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.dsacryptoserviceprovider(v=vs.71).aspx)
* [RSACryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsacryptoserviceprovider(v=vs.71).aspx)

**Digital Signatures**

Public-key algorithms can also be used to form digital signatures. Digital signatures authenticate the identity of a sender (if you trust the sender's public key) and help protect the integrity of data. Using a public key generated by Alice, the recipient of Alice's data can verify that Alice sent it by comparing the digital signature to Alice's data and Alice's public key.

To use public-key cryptography to digitally sign a message, Alice first applies a hash algorithm to the message to create a message digest. The message digest is a compact and unique representation of data. Alice then encrypts the message digest with her private key to create her personal signature. Upon receiving the message and signature, Bob decrypts the signature using Alice's public key to recover the message digest and hashes the message using the same hash algorithm that Alice used. If the message digest that Bob computes exactly matches the message digest received from Alice, Bob is assured that the message came from the possessor of the private key and that the data has not been modified. If Bob trusts that Alice is the possessor of the private key, then he knows that the message came from Alice.

Note that a signature can be verified by anyone because the sender's public key is common knowledge and is typically included in the digital signature format. This method does not retain the secrecy of the message; for the message to be secret, it must also be encrypted.

The .NET Framework provides the following classes that implement digital signature algorithms:

* [DSACryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.dsacryptoserviceprovider(v=vs.71).aspx)
* [RSACryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsacryptoserviceprovider(v=vs.71).aspx)

**Hash Values**

Hash algorithms map binary values of an arbitrary length to small binary values of a fixed length, known as hash values. A hash value is a unique and extremely compact numerical representation of a piece of data. If you hash a paragraph of plain text and change even one letter of the paragraph, a subsequent hash will produce a different value. It is computationally improbable to find two distinct inputs that hash to the same value.

Message authentication code (MAC) hash functions are commonly used with digital signatures to sign data, while message detection code (MDC) hash functions are used for data integrity.

Two parties (Alice and Bob) might use a hash function in the following way to ensure data integrity. If Alice writes a message for Bob and creates a hash of that message, Bob can then hash the message at a later time and compare his hash to the original hash. If the hash values are identical, then the message was not altered; however, if the values are not identical, the message was altered after Alice wrote it. For this system to work, Alice must hide the original hash value from all parties except Bob.

The .NET Framework provides the following classes that implement digital signature algorithms:

* [HMACSHA1](https://msdn.microsoft.com/en-us/library/system.security.cryptography.hmacsha1(v=vs.71).aspx)
* [MACTripleDES](https://msdn.microsoft.com/en-us/library/system.security.cryptography.mactripledes(v=vs.71).aspx)
* [MD5CryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.md5cryptoserviceprovider(v=vs.71).aspx)
* [SHA1Managed](https://msdn.microsoft.com/en-us/library/system.security.cryptography.sha1managed(v=vs.71).aspx)
* [SHA256Managed](https://msdn.microsoft.com/en-us/library/system.security.cryptography.sha256managed(v=vs.71).aspx)
* [SHA384Managed](https://msdn.microsoft.com/en-us/library/system.security.cryptography.sha384managed(v=vs.71).aspx)
* [SHA512Managed](https://msdn.microsoft.com/en-us/library/system.security.cryptography.sha512managed(v=vs.71).aspx)

**Random Number Generation**

Random number generation is integral to many cryptographic operations. For example, cryptographic keys need to be as random as possible so that it is infeasible to reproduce them. Cryptographic random number generators must generate output that is computationally infeasible to predict with better than a probability of p < .05; that is, any method of predicting the next output bit must not perform better than random guessing. The classes in the .NET Framework use random number generators to generate cryptographic keys.

The [RNGCryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rngcryptoserviceprovider(v=vs.71).aspx) class is an implementation of a random number generator algorithm.

### Cryptographic Services

**.NET Framework 3.5, 4, 4.5, 4.6**

Public networks such as the Internet do not provide a means of secure communication between entities. Communication over such networks is susceptible to being read or even modified by unauthorized third parties. Cryptography helps protect data from being viewed, provides ways to detect whether data has been modified, and helps provide a secure means of communication over otherwise nonsecure channels. For example, data can be encrypted by using a cryptographic algorithm, transmitted in an encrypted state, and later decrypted by the intended party. If a third party intercepts the encrypted data, it will be difficult to decipher.

In the .NET Framework, the classes in the [System.Security.Cryptography](https://msdn.microsoft.com/en-us/library/system.security.cryptography(v=vs.100).aspx) namespace manage many details of cryptography for you. Some are wrappers for the unmanaged Microsoft Cryptography API (CryptoAPI), while others are purely managed implementations. You do not need to be an expert in cryptography to use these classes. When you create a new instance of one of the encryption algorithm classes, keys are autogenerated for ease of use, and default properties are as safe and secure as possible.

This overview provides a synopsis of the encryption methods and practices supported by the .NET Framework, including the ClickOnce manifests, Suite B, and Cryptography Next Generation (CNG) support introduced in the .NET Framework version 3.5.

This overview contains the following sections:

* [Cryptographic Primitives](https://msdn.microsoft.com/en-us/library/92f9ye3s(v=vs.100).aspx#primitives)
* [Secret-Key Encryption](https://msdn.microsoft.com/en-us/library/92f9ye3s(v=vs.100).aspx#secret_key)
* [Public-Key Encryption](https://msdn.microsoft.com/en-us/library/92f9ye3s(v=vs.100).aspx#public_key)
* [Digital Signatures](https://msdn.microsoft.com/en-us/library/92f9ye3s(v=vs.100).aspx#digital_signatures)
* [Hash Values](https://msdn.microsoft.com/en-us/library/92f9ye3s(v=vs.100).aspx#hash_values)
* [Random Number Generation](https://msdn.microsoft.com/en-us/library/92f9ye3s(v=vs.100).aspx#random_numbers)
* [ClickOnce Manifests](https://msdn.microsoft.com/en-us/library/92f9ye3s(v=vs.100).aspx#clickonce)
* [Suite B Support](https://msdn.microsoft.com/en-us/library/92f9ye3s(v=vs.100).aspx#suite_b)
* [Cryptography Next Generation (CNG) Classes](https://msdn.microsoft.com/en-us/library/92f9ye3s(v=vs.100).aspx#cng)
* [Related Topics](https://msdn.microsoft.com/en-us/library/92f9ye3s(v=vs.100).aspx#related_topics)

For additional information about cryptography and about Microsoft services, components, and tools that enable you to add cryptographic security to your applications, see the Win32 and COM Development, Security section of this documentation.

[**Cryptographic Primitives**](javascript:void(0))

In a typical situation where cryptography is used, two parties (Alice and Bob) communicate over a nonsecure channel. Alice and Bob want to ensure that their communication remains incomprehensible by anyone who might be listening. Furthermore, because Alice and Bob are in remote locations, Alice must make sure that the information she receives from Bob has not been modified by anyone during transmission. In addition, she must make sure that the information really does originate from Bob and not from someone who is impersonating Bob.

Cryptography is used to achieve the following goals:

* Confidentiality: To help protect a user's identity or data from being read.
* Data integrity: To help protect data from being changed.
* Authentication: To ensure that data originates from a particular party.
* Non-repudiation: To prevent a particular party from denying that they sent a message.

To achieve these goals, you can use a combination of algorithms and practices known as cryptographic primitives to create a cryptographic scheme. The following table lists the cryptographic primitives and their uses.

|  |  |
| --- | --- |
| **Cryptographic primitive** | **Use** |
| Secret-key encryption (symmetric cryptography) | Performs a transformation on data to keep it from being read by third parties. This type of encryption uses a single shared, secret key to encrypt and decrypt data. |
| Public-key encryption (asymmetric cryptography) | Performs a transformation on data to keep it from being read by third parties. This type of encryption uses a public/private key pair to encrypt and decrypt data. |
| Cryptographic signing | Helps verify that data originates from a specific party by creating a digital signature that is unique to that party. This process also uses hash functions. |
| Cryptographic hashes | Maps data from any length to a fixed-length byte sequence. Hashes are statistically unique; a different two-byte sequence will not hash to the same value. |

[Back to top](https://msdn.microsoft.com/en-us/library/92f9ye3s(v=vs.100).aspx#top)

[**Secret-Key Encryption**](javascript:void(0))

Secret-key encryption algorithms use a single secret key to encrypt and decrypt data. You must secure the key from access by unauthorized agents, because any party that has the key can use it to decrypt your data or encrypt their own data, claiming it originated from you.

Secret-key encryption is also referred to as symmetric encryption because the same key is used for encryption and decryption. Secret-key encryption algorithms are very fast (compared with public-key algorithms) and are well suited for performing cryptographic transformations on large streams of data. Asymmetric encryption algorithms such as RSA are limited mathematically in how much data they can encrypt. Symmetric encryption algorithms do not generally have those problems.

A type of secret-key algorithm called a block cipher is used to encrypt one block of data at a time. Block ciphers such as Data Encryption Standard (DES), TripleDES, and Advanced Encryption Standard (AES) cryptographically transform an input block of *n* bytes into an output block of encrypted bytes. If you want to encrypt or decrypt a sequence of bytes, you have to do it block by block. Because *n* is small (8 bytes for DES and TripleDES; 16 bytes [the default], 24 bytes, or 32 bytes for AES), data values that are larger than *n* have to be encrypted one block at a time. Data values that are smaller than *n* have to be expanded to *n* in order to be processed.

One simple form of block cipher is called the electronic codebook (ECB) mode. ECB mode is not considered secure, because it does not use an initialization vector to initialize the first plaintext block. For a given secret key *k*, a simple block cipher that does not use an initialization vector will encrypt the same input block of plaintext into the same output block of ciphertext. Therefore, if you have duplicate blocks in your input plaintext stream, you will have duplicate blocks in your output ciphertext stream. These duplicate output blocks alert unauthorized users to the weak encryption used the algorithms that might have been employed, and the possible modes of attack. The ECB cipher mode is therefore quite vulnerable to analysis, and ultimately, key discovery.

The block cipher classes that are provided in the base class library use a default chaining mode called cipher-block chaining (CBC), although you can change this default if you want.

CBC ciphers overcome the problems associated with ECB ciphers by using an initialization vector (IV) to encrypt the first block of plaintext. Each subsequent block of plaintext undergoes a bitwise exclusive OR (**XOR**) operation with the previous ciphertext block before it is encrypted. Each ciphertext block is therefore dependent on all previous blocks. When this system is used, common message headers that might be known to an unauthorized user cannot be used to reverse-engineer a key.

One way to compromise data that is encrypted with a CBC cipher is to perform an exhaustive search of every possible key. Depending on the size of the key that is used to perform encryption, this kind of search is very time-consuming using even the fastest computers and is therefore infeasible. Larger key sizes are more difficult to decipher. Although encryption does not make it theoretically impossible for an adversary to retrieve the encrypted data, it does raise the cost of doing this. If it takes three months to perform an exhaustive search to retrieve data that is meaningful only for a few days, the exhaustive search method is impractical.

The disadvantage of secret-key encryption is that it presumes two parties have agreed on a key and IV, and communicated their values. The IV is not considered a secret and can be transmitted in plaintext with the message. However, the key must be kept secret from unauthorized users. Because of these problems, secret-key encryption is often used together with public-key encryption to privately communicate the values of the key and IV.

Assuming that Alice and Bob are two parties who want to communicate over a nonsecure channel, they might use secret-key encryption as follows: Alice and Bob agree to use one particular algorithm (AES, for example) with a particular key and IV. Alice composes a message and creates a network stream (perhaps a named pipe or network e-mail) on which to send the message. Next, she encrypts the text using the key and IV, and sends the encrypted message and IV to Bob over the intranet. Bob receives the encrypted text and decrypts it by using the IV and previously agreed upon key. If the transmission is intercepted, the interceptor cannot recover the original message, because he does not know the key. In this scenario, only the key must remain secret. In a real world scenario, either Alice or Bob generates a secret key and uses public-key (asymmetric) encryption to transfer the secret (symmetric) key to the other party. For more information about public-key encryption, see the next section.

The .NET Framework provides the following classes that implement secret-key encryption algorithms:

* [AesManaged](https://msdn.microsoft.com/en-us/library/system.security.cryptography.aesmanaged(v=vs.100).aspx) (introduced in the .NET Framework version 3.5).
* [DESCryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.descryptoserviceprovider(v=vs.100).aspx).
* [HMACSHA1](https://msdn.microsoft.com/en-us/library/system.security.cryptography.hmacsha1(v=vs.100).aspx) (This is technically a secret-key algorithm because it represents message authentication code that is calculated by using a cryptographic hash function combined with a secret key. See [Hash Values](https://msdn.microsoft.com/en-us/library/92f9ye3s(v=vs.100).aspx#hash_values), later in this topic.)
* [RC2CryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rc2cryptoserviceprovider(v=vs.100).aspx).
* [RijndaelManaged](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rijndaelmanaged(v=vs.100).aspx).
* [TripleDESCryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.tripledescryptoserviceprovider(v=vs.100).aspx).

[Back to top](https://msdn.microsoft.com/en-us/library/92f9ye3s(v=vs.100).aspx#top)

[**Public-Key Encryption**](javascript:void(0))

Public-key encryption uses a private key that must be kept secret from unauthorized users and a public key that can be made public to anyone. The public key and the private key are mathematically linked; data that is encrypted with the public key can be decrypted only with the private key, and data that is signed with the private key can be verified only with the public key. The public key can be made available to anyone; it is used for encrypting data to be sent to the keeper of the private key. Public-key cryptographic algorithms are also known as asymmetric algorithms because one key is required to encrypt data, and another key is required to decrypt data. A basic cryptographic rule prohibits key reuse, and both keys should be unique for each communication session. However, in practice, asymmetric keys are generally long-lived.

Two parties (Alice and Bob) might use public-key encryption as follows: First, Alice generates a public/private key pair. If Bob wants to send Alice an encrypted message, he asks her for her public key. Alice sends Bob her public key over a nonsecure network, and Bob uses this key to encrypt a message. Bob sends the encrypted message to Alice, and she decrypts it by using her private key. If Bob received Alice's key over a nonsecure channel, such as a public network, Bob is open to a man-in-the-middle attack. Therefore, Bob must verify with Alice that he has a correct copy of her public key.

During the transmission of Alice's public key, an unauthorized agent might intercept the key. Furthermore, the same agent might intercept the encrypted message from Bob. However, the agent cannot decrypt the message with the public key. The message can be decrypted only with Alice's private key, which has not been transmitted. Alice does not use her private key to encrypt a reply message to Bob, because anyone with the public key could decrypt the message. If Alice wants to send a message back to Bob, she asks Bob for his public key and encrypts her message using that public key. Bob then decrypts the message using his associated private key.

In this scenario, Alice and Bob use public-key (asymmetric) encryption to transfer a secret (symmetric) key and use secret-key encryption for the remainder of their session.

The following list offers comparisons between public-key and secret-key cryptographic algorithms:

* Public-key cryptographic algorithms use a fixed buffer size, whereas secret-key cryptographic algorithms use a variable-length buffer.
* Public-key algorithms cannot be used to chain data together into streams the way secret-key algorithms can, because only small amounts of data can be encrypted. Therefore, asymmetric operations do not use the same streaming model as symmetric operations.
* Public-key encryption has a much larger keyspace (range of possible values for the key) than secret-key encryption. Therefore, public-key encryption is less susceptible to exhaustive attacks that try every possible key.
* Public keys are easy to distribute because they do not have to be secured, provided that some way exists to verify the identity of the sender.
* Some public-key algorithms (such as RSA and DSA, but not Diffie-Hellman) can be used to create digital signatures to verify the identity of the sender of data.
* Public-key algorithms are very slow compared with secret-key algorithms, and are not designed to encrypt large amounts of data. Public-key algorithms are useful only for transferring very small amounts of data. Typically, public-key encryption is used to encrypt a key and IV to be used by a secret-key algorithm. After the key and IV are transferred, secret-key encryption is used for the remainder of the session.

The .NET Framework provides the following classes that implement public-key encryption algorithms:

* [DSACryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.dsacryptoserviceprovider(v=vs.100).aspx)
* [RSACryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsacryptoserviceprovider(v=vs.100).aspx)
* [ECDiffieHellman](https://msdn.microsoft.com/en-us/library/system.security.cryptography.ecdiffiehellman(v=vs.100).aspx) (base class)
* [ECDiffieHellmanCng](https://msdn.microsoft.com/en-us/library/system.security.cryptography.ecdiffiehellmancng(v=vs.100).aspx)
* [ECDiffieHellmanCngPublicKey](https://msdn.microsoft.com/en-us/library/system.security.cryptography.ecdiffiehellmancngpublickey(v=vs.100).aspx) (base class)
* [ECDiffieHellmanKeyDerivationFunction](https://msdn.microsoft.com/en-us/library/system.security.cryptography.ecdiffiehellmankeyderivationfunction(v=vs.100).aspx) (base class)
* [ECDsaCng](https://msdn.microsoft.com/en-us/library/system.security.cryptography.ecdsacng(v=vs.100).aspx)

RSA allows both encryption and signing, but DSA can be used only for signing, and Diffie-Hellman can be used only for key generation. In general, public-key algorithms are more limited in their uses than private-key algorithms.

[Back to top](https://msdn.microsoft.com/en-us/library/92f9ye3s(v=vs.100).aspx#top)

[**Digital Signatures**](javascript:void(0))

Public-key algorithms can also be used to form digital signatures. Digital signatures authenticate the identity of a sender (if you trust the sender's public key) and help protect the integrity of data. Using a public key generated by Alice, the recipient of Alice's data can verify that Alice sent it by comparing the digital signature to Alice's data and Alice's public key.

To use public-key cryptography to digitally sign a message, Alice first applies a hash algorithm to the message to create a message digest. The message digest is a compact and unique representation of data. Alice then encrypts the message digest with her private key to create her personal signature. Upon receiving the message and signature, Bob decrypts the signature using Alice's public key to recover the message digest and hashes the message using the same hash algorithm that Alice used. If the message digest that Bob computes exactly matches the message digest received from Alice, Bob is assured that the message came from the possessor of the private key and that the data has not been modified. If Bob trusts that Alice is the possessor of the private key, he knows that the message came from Alice.

|  |
| --- |
| **NoteNote** |
| A signature can be verified by anyone because the sender's public key is common knowledge and is typically included in the digital signature format. This method does not retain the secrecy of the message; for the message to be secret, it must also be encrypted. |

The .NET Framework provides the following classes that implement digital signature algorithms:

* [DSACryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.dsacryptoserviceprovider(v=vs.100).aspx)
* [RSACryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsacryptoserviceprovider(v=vs.100).aspx)
* [ECDsa](https://msdn.microsoft.com/en-us/library/system.security.cryptography.ecdsa(v=vs.100).aspx) (base class)
* [ECDsaCng](https://msdn.microsoft.com/en-us/library/system.security.cryptography.ecdsacng(v=vs.100).aspx)

[Back to top](https://msdn.microsoft.com/en-us/library/92f9ye3s(v=vs.100).aspx#top)

[**Hash Values**](javascript:void(0))

Hash algorithms map binary values of an arbitrary length to smaller binary values of a fixed length, known as hash values. A hash value is a numerical representation of a piece of data. If you hash a paragraph of plaintext and change even one letter of the paragraph, a subsequent hash will produce a different value. If the hash is cryptographically strong, its value will change significantly. For example, if a single bit of a message is changed, a strong hash function may produce an output that differs by 50 percent. Many input values may hash to the same output value. However, it is computationally infeasible to find two distinct inputs that hash to the same value.

Two parties (Alice and Bob) could use a hash function to ensure message integrity. They would select a hash algorithm to sign their messages. Alice would write a message, and then create a hash of that message by using the selected algorithm. They would then follow one of the following methods:

* Alice sends the plaintext message and the hashed message (digital signature) to Bob. Bob receives and hashes the message and compares his hash value to the hash value that he received from Alice. If the hash values are identical, the message was not altered. If the values are not identical, the message was altered after Alice wrote it.

Unfortunately, this method does not establish the authenticity of the sender. Anyone can impersonate Alice and send a message to Bob. They can use the same hash algorithm to sign their message, and all Bob can determine is that the message matches its signature. This is one form of a man-in-the-middle attack. See [Cryptography Next Generation (CNG) Secure Communication Example](https://msdn.microsoft.com/en-us/library/cc488018(v=vs.100).aspx) for more information.

* Alice sends the plaintext message to Bob over a nonsecure public channel. She sends the hashed message to Bob over a secure private channel. Bob receives the plaintext message, hashes it, and compares the hash to the privately exchanged hash. If the hashes match, Bob knows two things:
  + The message was not altered.
  + The sender of the message (Alice) is authentic.

For this system to work, Alice must hide her original hash value from all parties except Bob.

* Alice sends the plaintext message to Bob over a nonsecure public channel and places the hashed message on her publicly viewable Web site.

This method prevents message tampering by preventing anyone from modifying the hash value. Although the message and its hash can be read by anyone, the hash value can be changed only by Alice. An attacker who wants to impersonate Alice would require access to Alice's Web site.

None of the previous methods will prevent someone from reading Alice's messages, because they are transmitted in plaintext. Full security typically requires digital signatures (message signing) and encryption.

The .NET Framework provides the following classes that implement hashing algorithms:

* [HMACSHA1](https://msdn.microsoft.com/en-us/library/system.security.cryptography.hmacsha1(v=vs.100).aspx).
* [MACTripleDES](https://msdn.microsoft.com/en-us/library/system.security.cryptography.mactripledes(v=vs.100).aspx).
* [MD5CryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.md5cryptoserviceprovider(v=vs.100).aspx).
* [RIPEMD160](https://msdn.microsoft.com/en-us/library/system.security.cryptography.ripemd160(v=vs.100).aspx).
* [SHA1Managed](https://msdn.microsoft.com/en-us/library/system.security.cryptography.sha1managed(v=vs.100).aspx).
* [SHA256Managed](https://msdn.microsoft.com/en-us/library/system.security.cryptography.sha256managed(v=vs.100).aspx).
* [SHA384Managed](https://msdn.microsoft.com/en-us/library/system.security.cryptography.sha384managed(v=vs.100).aspx).
* [SHA512Managed](https://msdn.microsoft.com/en-us/library/system.security.cryptography.sha512managed(v=vs.100).aspx).
* HMAC variants of all of the Secure Hash Algorithm (SHA), Message Digest 5 (MD5), and RIPEMD-160 algorithms.
* CryptoServiceProvider implementations (managed code wrappers) of all the SHA algorithms.
* Cryptography Next Generation (CNG) implementations of all the MD5 and SHA algorithms.

|  |
| --- |
| **NoteNote** |
| MD5 design flaws were discovered in 1996, and SHA-1 was recommended instead. In 2004, additional flaws were discovered, and the MD5 algorithm is no longer considered secure. The SHA-1 algorithm has also been found to be insecure, and SHA-2 is now recommended instead. |

[Back to top](https://msdn.microsoft.com/en-us/library/92f9ye3s(v=vs.100).aspx#top)

[**Random Number Generation**](javascript:void(0))

Random number generation is integral to many cryptographic operations. For example, cryptographic keys need to be as random as possible so that it is infeasible to reproduce them. Cryptographic random number generators must generate output that is computationally infeasible to predict with a probability that is better than one half. Therefore, any method of predicting the next output bit must not perform better than random guessing. The classes in the .NET Framework use random number generators to generate cryptographic keys.

The [RNGCryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rngcryptoserviceprovider(v=vs.100).aspx) class is an implementation of a random number generator algorithm.

[Back to top](https://msdn.microsoft.com/en-us/library/92f9ye3s(v=vs.100).aspx#top)

[**ClickOnce Manifests**](javascript:void(0))

In the .NET Framework 3.5, the following cryptography classes let you obtain and verify information about manifest signatures for applications that are deployed using [ClickOnce technology](https://msdn.microsoft.com/en-us/library/t71a733d(v=vs.100).aspx):

* The [ManifestSignatureInformation](https://msdn.microsoft.com/en-us/library/system.security.cryptography.manifestsignatureinformation(v=vs.100).aspx) class obtains information about a manifest signature when you use its [VerifySignature](https://msdn.microsoft.com/en-us/library/system.security.cryptography.manifestsignatureinformation.verifysignature(v=vs.100).aspx) method overloads.
* You can use the [ManifestKinds](https://msdn.microsoft.com/en-us/library/bb340409(v=vs.100).aspx) enumeration to specify which manifests to verify. The result of the verification is one of the[SignatureVerificationResult](https://msdn.microsoft.com/en-us/library/bb461969(v=vs.100).aspx) enumeration values.
* The [ManifestSignatureInformationCollection](https://msdn.microsoft.com/en-us/library/system.security.cryptography.manifestsignatureinformationcollection(v=vs.100).aspx) class provides a read-only collection of [ManifestSignatureInformation](https://msdn.microsoft.com/en-us/library/system.security.cryptography.manifestsignatureinformation(v=vs.100).aspx) objects of the verified signatures.

In addition, the following classes provide specific signature information:

* [StrongNameSignatureInformation](https://msdn.microsoft.com/en-us/library/system.security.cryptography.strongnamesignatureinformation(v=vs.100).aspx) holds the strong name signature information for a manifest.
* [AuthenticodeSignatureInformation](https://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.authenticodesignatureinformation(v=vs.100).aspx) represents the Authenticode signature information for a manifest.
* [TimestampInformation](https://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.timestampinformation(v=vs.100).aspx) contains information about the time stamp on an Authenticode signature.
* [TrustStatus](https://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.truststatus(v=vs.100).aspx) provides a simple way to check whether an Authenticode signature is trusted.

[Back to top](https://msdn.microsoft.com/en-us/library/92f9ye3s(v=vs.100).aspx#top)

[**Suite B Support**](javascript:void(0))

The .NET Framework 3.5 supports the Suite B set of cryptographic algorithms published by the National Security Agency (NSA). For more information about Suite B, see the [NSA Suite B Cryptography Fact Sheet](http://go.microsoft.com/fwlink/?LinkId=100111).

The following algorithms are included:

* Advanced Encryption Standard (AES) algorithm with key sizes of 128, 192, , and 256 bits for encryption.
* Secure Hash Algorithms SHA-1, SHA-256, SHA-384, and SHA-512 for hashing. (The last three are generally grouped together and referred to as SHA-2.)
* Elliptic Curve Digital Signature Algorithm (ECDSA), using curves of 256-bit, 384-bit, and 521-bit prime moduli for signing. The NSA documentation specifically defines these curves, and calls them P-256, P-384, and P-521. This algorithm is provided by the[ECDsaCng](https://msdn.microsoft.com/en-us/library/system.security.cryptography.ecdsacng(v=vs.100).aspx) class. It enables you to sign with a private key and verify the signature with a public key.
* Elliptic Curve Diffie-Hellman (ECDH) algorithm, using curves of 256-bit, 384-bit, and 521-bit prime moduli for the key exchange and secret agreement. This algorithm is provided by the [ECDiffieHellmanCng](https://msdn.microsoft.com/en-us/library/system.security.cryptography.ecdiffiehellmancng(v=vs.100).aspx) class.

Managed code wrappers for the Federal Information Processing Standard (FIPS) certified implementations of the AES, SHA-256, SHA-384, and SHA-512 implementations are available in the new [AesCryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.aescryptoserviceprovider(v=vs.100).aspx), [SHA256CryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.sha256cryptoserviceprovider(v=vs.100).aspx),[SHA384CryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.sha384cryptoserviceprovider(v=vs.100).aspx), and [SHA512CryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.sha512cryptoserviceprovider(v=vs.100).aspx) classes.

[Back to top](https://msdn.microsoft.com/en-us/library/92f9ye3s(v=vs.100).aspx#top)

[**Cryptography Next Generation (CNG) Classes**](javascript:void(0))

The Cryptography Next Generation (CNG) classes provide a managed wrapper around the native CNG functions. (CNG is the replacement for CryptoAPI.) These classes have "Cng" as part of their names. Central to the CNG wrapper classes is the [CngKey](https://msdn.microsoft.com/en-us/library/system.security.cryptography.cngkey(v=vs.100).aspx) key container class, which abstracts the storage and use of CNG keys. This class lets you store a key pair or a public key securely and refer to it by using a simple string name. The elliptic curve-based [ECDsaCng](https://msdn.microsoft.com/en-us/library/system.security.cryptography.ecdsacng(v=vs.100).aspx) signature class and the [ECDiffieHellmanCng](https://msdn.microsoft.com/en-us/library/system.security.cryptography.ecdiffiehellmancng(v=vs.100).aspx) encryption class can use [CngKey](https://msdn.microsoft.com/en-us/library/system.security.cryptography.cngkey(v=vs.100).aspx)objects.

The [CngKey](https://msdn.microsoft.com/en-us/library/system.security.cryptography.cngkey(v=vs.100).aspx) class is used for a variety of additional operations, including opening, creating, deleting, and exporting keys. It also provides access to the underlying key handle to use when calling native functions directly.

The .NET Framework 3.5 also includes a variety of supporting CNG classes, such as the following:

* [CngProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.cngprovider(v=vs.100).aspx) maintains a key storage provider.
* [CngAlgorithm](https://msdn.microsoft.com/en-us/library/system.security.cryptography.cngalgorithm(v=vs.100).aspx) maintains a CNG algorithm.
* [CngProperty](https://msdn.microsoft.com/en-us/library/system.security.cryptography.cngproperty(v=vs.100).aspx) maintains frequently used key properties.

[Back to top](https://msdn.microsoft.com/en-us/library/92f9ye3s(v=vs.100).aspx#top)

[**Related Topics**](javascript:void(0))

|  |  |
| --- | --- |
| **Title** | **Description** |
| [.NET Framework Cryptography Model](https://msdn.microsoft.com/en-us/library/0ss79b2x(v=vs.100).aspx) | Describes how cryptography is implemented in the base class library. |
| [Cryptographic Tasks](https://msdn.microsoft.com/en-us/library/7yx4d854(v=vs.100).aspx) | Describes how to perform specific cryptographic tasks using the base class library. |
| [Walkthrough: Creating a Cryptographic Application](https://msdn.microsoft.com/en-us/library/bb397867(v=vs.100).aspx) | Demonstrates basic encryption and decryption tasks. |
| [Configuring Cryptography Classes](https://msdn.microsoft.com/en-us/library/bke5we9a(v=vs.100).aspx) | Describes how to map algorithm names to cryptographic classes and map object identifiers to a cryptographic algorithm. |

[Back to top](https://msdn.microsoft.com/en-us/library/92f9ye3s(v=vs.100).aspx#top)

### .NET Framework Cryptography Model

**.NET Framework 1.1, 2.0, 3.0, 3.5**

The .NET Framework provides implementations of many standard cryptographic algorithms. These algorithms are easy to use and have the safest possible default properties. In addition, the .NET Framework cryptography model of object inheritance, stream design, and configuration are extremely extensible.

**Object Inheritance**

The .NET Framework security system implements an extensible pattern of derived class inheritance. The hierarchy is as follows:

* Algorithm type class, such as [SymmetricAlgorithm](https://msdn.microsoft.com/en-us/library/system.security.cryptography.symmetricalgorithm(v=vs.71).aspx) or [HashAlgorithm](https://msdn.microsoft.com/en-us/library/system.security.cryptography.hashalgorithm(v=vs.71).aspx). This level is abstract.
* Algorithm class that inherits from an algorithm type class; for example, [RC2](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rc2(v=vs.71).aspx) or [SHA1](https://msdn.microsoft.com/en-us/library/system.security.cryptography.sha1(v=vs.71).aspx). This level is abstract.
* Implementation of an algorithm class that inherits from an algorithm class; for example, [RC2CryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rc2cryptoserviceprovider(v=vs.71).aspx) or [SHA1Managed](https://msdn.microsoft.com/en-us/library/system.security.cryptography.sha1managed(v=vs.71).aspx). This level is fully implemented.

Using this pattern of derived classes, it is easy to add a new algorithm or a new implementation of an existing algorithm. For example, to create a new public-key algorithm, you would inherit from the [AsymmetricAlgorithm](https://msdn.microsoft.com/en-us/library/system.security.cryptography.asymmetricalgorithm(v=vs.71).aspx) class. To create a new implementation of a specific algorithm, you would create a nonabstract derived class of that algorithm.

**Stream Design**

The common language runtime uses a stream-oriented design for implementing symmetric algorithms and hash algorithms. The core of this design is the [CryptoStream](https://msdn.microsoft.com/en-us/library/system.security.cryptography.cryptostream(v=vs.71).aspx) class, which derives from the [Stream](https://msdn.microsoft.com/en-us/library/system.io.stream(v=vs.71).aspx) class. Stream-based cryptographic objects all support a single standard interface (**CryptoStream**) for handling the data transfer portion of the object. Because all the objects are built on a standard interface, you can chain together multiple objects (such as a hash object followed by an encryption object), and you can perform multiple operations on the data without needing any intermediate storage for it. The streaming model also allows you to build objects from smaller objects. For example, a combined encryption and hash algorithm can be viewed as a single stream object even though this object might be built from a set of stream objects.

**Cryptographic Configuration**

Cryptographic configuration allows you to resolve a specific implementation of an algorithm to an algorithm name, allowing extensibility of the .NET Framework cryptography classes. You can add your own hardware or software implementation of an algorithm and map the implementation to the algorithm name of your choice. If an algorithm is not specified in the configuration file, the default settings are used. For more information on cryptographic configuration, see [Configuring Cryptography Classes](https://msdn.microsoft.com/en-us/library/bke5we9a(v=vs.71).aspx).

### .NET Framework Cryptography Model

**.NET Framework 4, 4.5, 4.6**

The .NET Framework provides implementations of many standard cryptographic algorithms. These algorithms are easy to use and have the safest possible default properties. In addition, the .NET Framework cryptography model of object inheritance, stream design, and configuration is extremely extensible.

[**Object Inheritance**](javascript:void(0))

The .NET Framework security system implements an extensible pattern of derived class inheritance. The hierarchy is as follows:

* Algorithm type class, such as [SymmetricAlgorithm](https://msdn.microsoft.com/en-us/library/system.security.cryptography.symmetricalgorithm(v=vs.100).aspx), [AsymmetricAlgorithm](https://msdn.microsoft.com/en-us/library/system.security.cryptography.asymmetricalgorithm(v=vs.100).aspx) or [HashAlgorithm](https://msdn.microsoft.com/en-us/library/system.security.cryptography.hashalgorithm(v=vs.100).aspx). This level is abstract.
* Algorithm class that inherits from an algorithm type class; for example, [Aes](https://msdn.microsoft.com/en-us/library/system.security.cryptography.aes(v=vs.100).aspx), [RC2](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rc2(v=vs.100).aspx), or [ECDiffieHellman](https://msdn.microsoft.com/en-us/library/system.security.cryptography.ecdiffiehellman(v=vs.100).aspx). This level is abstract.
* Implementation of an algorithm class that inherits from an algorithm class; for example, [AesManaged](https://msdn.microsoft.com/en-us/library/system.security.cryptography.aesmanaged(v=vs.100).aspx), [RC2CryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rc2cryptoserviceprovider(v=vs.100).aspx), or[ECDiffieHellmanCng](https://msdn.microsoft.com/en-us/library/system.security.cryptography.ecdiffiehellmancng(v=vs.100).aspx). This level is fully implemented.

Using this pattern of derived classes, it is easy to add a new algorithm or a new implementation of an existing algorithm. For example, to create a new public-key algorithm, you would inherit from the [AsymmetricAlgorithm](https://msdn.microsoft.com/en-us/library/system.security.cryptography.asymmetricalgorithm(v=vs.100).aspx) class. To create a new implementation of a specific algorithm, you would create a non-abstract derived class of that algorithm.

[**How Algorithms Are Implemented in the .NET Framework**](javascript:void(0))

As an example of the different implementations available for an algorithm, consider symmetric algorithms. The base for all symmetric algorithms is [SymmetricAlgorithm](https://msdn.microsoft.com/en-us/library/system.security.cryptography.symmetricalgorithm(v=vs.100).aspx), which is inherited by the following algorithms:

1. [Aes](https://msdn.microsoft.com/en-us/library/system.security.cryptography.aes(v=vs.100).aspx)
2. [DES](https://msdn.microsoft.com/en-us/library/system.security.cryptography.des(v=vs.100).aspx)
3. [RC2](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rc2(v=vs.100).aspx)
4. [Rijndael](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rijndael(v=vs.100).aspx)
5. [TripleDES](https://msdn.microsoft.com/en-us/library/system.security.cryptography.tripledes(v=vs.100).aspx)

[Aes](https://msdn.microsoft.com/en-us/library/system.security.cryptography.aes(v=vs.100).aspx) is inherited by two classes: [AesCryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.aescryptoserviceprovider(v=vs.100).aspx) and [AesManaged](https://msdn.microsoft.com/en-us/library/system.security.cryptography.aesmanaged(v=vs.100).aspx). The [AesCryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.aescryptoserviceprovider(v=vs.100).aspx) class is a wrapper around the Windows Cryptography API (CAPI) implementation of Aes, whereas the [AesManaged](https://msdn.microsoft.com/en-us/library/system.security.cryptography.aesmanaged(v=vs.100).aspx) class is written entirely in managed code. There is also a third type of implementation, Cryptography Next Generation (CNG), in addition to the managed and CAPI implementations. An example of a CNG algorithm is [ECDiffieHellmanCng](https://msdn.microsoft.com/en-us/library/system.security.cryptography.ecdiffiehellmancng(v=vs.100).aspx). CNG algorithms are available on Windows Vista and later.

You can choose which implementation is best for you. The managed implementations are available on all platforms that support the .NET Framework. The CAPI implementations are available on older operating systems, and are no longer being developed. CNG is the very latest implementation where new development will take place. However, the managed implementations are not certified by the Federal Information Processing Standards (FIPS), and may be slower than the wrapper classes.

[**Stream Design**](javascript:void(0))

The common language runtime uses a stream-oriented design for implementing symmetric algorithms and hash algorithms. The core of this design is the [CryptoStream](https://msdn.microsoft.com/en-us/library/system.security.cryptography.cryptostream(v=vs.100).aspx) class, which derives from the [Stream](https://msdn.microsoft.com/en-us/library/system.io.stream(v=vs.100).aspx) class. Stream-based cryptographic objects support a single standard interface (**CryptoStream**) for handling the data transfer portion of the object. Because all the objects are built on a standard interface, you can chain together multiple objects (such as a hash object followed by an encryption object), and you can perform multiple operations on the data without needing any intermediate storage for it. The streaming model also enables you to build objects from smaller objects. For example, a combined encryption and hash algorithm can be viewed as a single stream object, although this object might be built from a set of stream objects.

[**Cryptographic Configuration**](javascript:void(0))

Cryptographic configuration lets you resolve a specific implementation of an algorithm to an algorithm name, allowing extensibility of the .NET Framework cryptography classes. You can add your own hardware or software implementation of an algorithm and map the implementation to the algorithm name of your choice. If an algorithm is not specified in the configuration file, the default settings are used. For more information about cryptographic configuration, see [Configuring Cryptography Classes](https://msdn.microsoft.com/en-us/library/bke5we9a(v=vs.100).aspx).

[**Choosing an Algorithm**](javascript:void(0))

You can select an algorithm for different reasons: for example, for data integrity, for data privacy, or to generate a key. Symmetric and hash algorithms are intended for protecting data for either integrity reasons (protect from change) or privacy reasons (protect from viewing). Hash algorithms are used primarily for data integrity.

Here is a list of recommended algorithms by application:

* Data privacy:
  + [Aes](https://msdn.microsoft.com/en-us/library/system.security.cryptography.aes(v=vs.100).aspx)
* Data integrity:
  + [HMACSHA256](https://msdn.microsoft.com/en-us/library/system.security.cryptography.hmacsha256(v=vs.100).aspx)
  + [HMACSHA512](https://msdn.microsoft.com/en-us/library/system.security.cryptography.hmacsha512(v=vs.100).aspx)
* Digital signature:
  + [ECDsa](https://msdn.microsoft.com/en-us/library/system.security.cryptography.ecdsa(v=vs.100).aspx)
  + [RSA](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsa(v=vs.100).aspx)
* Key exchange:
  + [ECDiffieHellman](https://msdn.microsoft.com/en-us/library/system.security.cryptography.ecdiffiehellman(v=vs.100).aspx)
  + [RSA](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsa(v=vs.100).aspx)
* Random number generation:
  + [RNGCryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rngcryptoserviceprovider(v=vs.100).aspx)
* Generating a key from a password:
  + [Rfc2898DeriveBytes](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rfc2898derivebytes(v=vs.100).aspx)

### Cryptographic Tasks

**.NET Framework 1.1**

The [System.Security.Cryptography](https://msdn.microsoft.com/en-us/library/system.security.cryptography(v=vs.71).aspx) namespace contains classes that allow you to perform both symmetric and asymmetric cryptography, create hashes, and provide random number generation. Successful cryptography is the result of combining these tasks. This section describes the key cryptographic tasks that you can perform to create a cryptographic scheme.

**In This Section**

[Encrypting and Decrypting Data](https://msdn.microsoft.com/en-us/library/e970bs09(v=vs.71).aspx)

Describes how to generate and manage keys and how to encrypt and decrypt data.

[Cryptographic Signatures](https://msdn.microsoft.com/en-us/library/hk8wx38z(v=vs.71).aspx)

Describes how to generate and verify cryptographic signatures.

[Ensuring Data Integrity with Hash Codes](https://msdn.microsoft.com/en-us/library/f9ax34y5(v=vs.71).aspx)

Describes how to generate and verify hash codes.

[Creating a Cryptographic Scheme](https://msdn.microsoft.com/en-us/library/0cwc0x23(v=vs.71).aspx)

Describes how to help create a cryptographic scheme from the various cryptographic primitives.

[Extending the KeyedHashAlgorithm Class](https://msdn.microsoft.com/en-us/library/33kc9tdw(v=vs.71).aspx)

Describes how to extend the .NET Framework cryptographic classes by creating a keyed hash algorithm class that implements the [MD5](https://msdn.microsoft.com/en-us/library/system.security.cryptography.md5(v=vs.71).aspx)hash algorithm.

**Related Sections**

[Cryptographic Services](https://msdn.microsoft.com/en-us/library/93bskf9z(v=vs.71).aspx)

Provides an overview of cryptography and explains how to perform cryptography with the .NET Framework.

[Securing Applications](https://msdn.microsoft.com/en-us/library/fkytk30f(v=vs.71).aspx)

Describes the entire .NET Framework security system.

[Configuring Cryptography Classes](https://msdn.microsoft.com/en-us/library/bke5we9a(v=vs.71).aspx)

Describes how to map algorithm names to cryptographic classes and how to map object identifiers to a cryptographic algorithm.

### Cryptographic Tasks

**.NET Framework 2.0, 3.0, 3.5, 4**

The [System.Security.Cryptography](https://msdn.microsoft.com/en-us/library/system.security.cryptography(v=vs.80).aspx) namespace contains classes that allow you to perform both symmetric and asymmetric cryptography, create hashes, and provide random number generation. Successful cryptography is the result of combining these tasks. This section describes the key cryptographic tasks that you can perform to create a cryptographic scheme.

**In This Section**

[Encrypting and Decrypting Data](https://msdn.microsoft.com/en-us/library/e970bs09(v=vs.80).aspx)

Describes how to generate and manage keys and how to encrypt and decrypt data.

[Cryptographic Signatures](https://msdn.microsoft.com/en-us/library/hk8wx38z(v=vs.80).aspx)

Describes how to generate and verify cryptographic signatures.

[Ensuring Data Integrity with Hash Codes](https://msdn.microsoft.com/en-us/library/f9ax34y5(v=vs.80).aspx)

Describes how to generate and verify hash codes.

[Creating a Cryptographic Scheme](https://msdn.microsoft.com/en-us/library/0cwc0x23(v=vs.80).aspx)

Describes how to help create a cryptographic scheme from the various cryptographic primitives.

[Extending the KeyedHashAlgorithm Class](https://msdn.microsoft.com/en-us/library/33kc9tdw(v=vs.80).aspx)

Describes how to extend the .NET Framework cryptographic classes by creating a keyed hash algorithm class that implements the [MD5](https://msdn.microsoft.com/en-us/library/system.security.cryptography.md5(v=vs.80).aspx)hash algorithm.

[XML Encryption and Digital Signatures](https://msdn.microsoft.com/en-us/library/ms229749(v=vs.80).aspx)

Provides links to reference and task-based documentation for XML encryption and digital signatures.

[How to: Use Data Protection](https://msdn.microsoft.com/en-us/library/ms229741(v=vs.80).aspx)

Describes how to use the managed data protection API (DPAPI) to encrypt and decrypt data.

[How to: Access Hardware Encryption Devices](https://msdn.microsoft.com/en-us/library/ms229931(v=vs.80).aspx)

Describes how to use hardware encryption devices with the .NET Framework.

**Related Sections**

[Cryptographic Services](https://msdn.microsoft.com/en-us/library/93bskf9z(v=vs.80).aspx)

Provides an overview of cryptography and explains how to perform cryptography with the .NET Framework.

[Securing .NET Framework Applications](https://msdn.microsoft.com/en-us/library/fkytk30f(v=vs.80).aspx)

Describes the entire .NET Framework security system.

[Configuring Cryptography Classes](https://msdn.microsoft.com/en-us/library/bke5we9a(v=vs.80).aspx)

Describes how to map algorithm names to cryptographic classes and how to map object identifiers to a cryptographic algorithm.

#### Encrypting and Decrypting Data

**.NET Framework 1.1, 2.0, 3.0**

To encrypt and decrypt data, you must use a key with an encryption algorithm that performs a transformation on the data. The .NET Framework provides several classes that enable you to perform cryptographic transformations on data using several standard algorithms. This section describes how to create and manage keys and how to encrypt and decrypt data using public-key and secret-key algorithms.

#### Encrypting and Decrypting Data

**.NET Framework 3.5, 4**

Updated: January 2010

To encrypt and decrypt data, you must use a key with an encryption algorithm that performs a transformation on the data. The .NET Framework provides several classes that enable you to perform cryptographic transformations on data using several standard algorithms. This section describes how to create and manage keys and how to encrypt and decrypt data using public-key and secret-key algorithms.

[**In This Section**](javascript:void(0))

[Generating Keys for Encryption and Decryption](https://msdn.microsoft.com/en-us/library/5e9ft273(v=vs.90).aspx)

Describes symmetric and asymmetric algorithms used for encryption and decryption.

[How to: Store Asymmetric Keys in a Key Container](https://msdn.microsoft.com/en-us/library/tswxhw92(v=vs.90).aspx)

Describes how to store private keys in a key container.

[Encrypting Data](https://msdn.microsoft.com/en-us/library/as0w18af(v=vs.90).aspx)

Explains how to do symmetric and asymmetric encryption.

[Decrypting Data](https://msdn.microsoft.com/en-us/library/te15te69(v=vs.90).aspx)

Explains how to do symmetric and asymmetric decryption.

##### Generating Keys for Encryption and Decryption

**.NET Framework 1.1**

Creating and managing keys is an important part of the cryptographic process. Symmetric algorithms require the creation of a key and an initialization vector (IV) that must be kept secret from anyone who should not decrypt your data. Asymmetric algorithms require the creation of a public key and a private key. The public key can be made public to anyone, while the private key must known only by the party who will decrypt the data encrypted with the public key. This section describes how to generate and manage keys for both symmetric and asymmetric algorithms.

**Symmetric Keys**

The symmetric encryption classes supplied by the .NET Framework require a key and a new IV to encrypt and decrypt data. Whenever you create a new instance of one of the managed symmetric cryptographic classes using the default constructor, a new key and IV are automatically created. Anyone whom you want to be able to decrypt your data must possess the same key and IV and use the same encryption algorithm. Generally, a new key and IV should be created for every session, and neither the key nor IV should be stored for use in a later session.

To communicate a symmetric key and IV to a remote party, you would usually encrypt the symmetric key and IV using asymmetric encryption. Sending these values across an insecure network without encrypting them is extremely unsafe, as anyone that intercepts these values has the ability to decrypt your data. For more information on this process of encrypting and transferring the key and IV, see [Creating a Cryptographic Scheme](https://msdn.microsoft.com/en-us/library/0cwc0x23(v=vs.71).aspx).

The following example shows the creation of a new instance of the [TripleDESCryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.tripledescryptoserviceprovider(v=vs.71).aspx) class that implements the TripleDES algorithm.

VB

Dim TDES As TripleDESCryptoServiceProvider = new TripleDESCryptoServiceProvider()

[C#]

TripleDESCryptoServiceProvider TDES = new TripleDESCryptoServiceProvider();

When the previous code is executed, a new key and IV are generated and placed in the **Key** and **IV** properties, respectively.

Sometimes you might need to generate multiple keys. In this situation, you can create a new instance of a class that implements a symmetric algorithm and then create a new key and IV by calling the **GenerateKey** and **GenerateIV** methods. The following code example illustrates how to create new keys and IVs after a new instance of the asymmetric cryptographic class has been made.

VB

Dim TDES As TripleDESCryptoServiceProvider = new TripleDESCryptoServiceProvider()

TDES.GenerateIV()

TDES.GenerateKey()

[C#]

TripleDESCryptoServiceProvider TDES = new TripleDESCryptoServiceProvider();

TDES.GenerateIV();

TDES.GenerateKey();

When the previous code is executed, a key and IV are generated when the new instance of **TripleDESCryptoServiceProvider** is made. Another key and IV are created when the **GenerateKey** and **GenerateIV** methods are called.

**Asymmetric Keys**

The .NET Framework provides the [RSACryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsacryptoserviceprovider(v=vs.71).aspx) and [DSACryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.dsacryptoserviceprovider(v=vs.71).aspx) classes for asymmetric encryption. These classes create a public/private key pair when you use the default constructor to create a new instance. Asymmetric keys can be either stored for use in multiple sessions or generated for one session only. While the public key can be made generally available, the private key should be closely guarded.

A public/private key pair is generated whenever a new instance of an asymmetric algorithm class is created. Once a new instance of the class is created, the key information can be extracted using one of two methods:

* The [ToXMLString](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsa.toxmlstring(v=vs.71).aspx) method, which returns an XML representation of the key information.
* The [ExportParameters](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsacryptoserviceprovider.exportparameters(v=vs.71).aspx) method, which returns an [RSAParameters](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsaparameters(v=vs.71).aspx) enumeration to hold the key information.

Both methods accept a Boolean value that indicates whether to return only the public key information or to return both the public-key and the private-key information. An **RSACryptoServiceProvider** class can be initialized to the value of an **RSAParameters** structure by using the[ImportParameters](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsacryptoserviceprovider.importparameters(v=vs.71).aspx) method.

The following code example creates a new instance of the **RSACryptoServiceProvider**class, creating a public/private key pair, and saves the public key information to an **RSAParameters**structure.

VB

'Generate a public/private key pair.

Dim RSA as RSACryptoServiceProvider = new RSACryptoServiceProvider()

'Save the public key information to an RSAParameters structure.

Dim RSAKeyInfo As RSAParameters = RSA.ExportParameters(false)

[C#]

//Generate a public/private key pair.

RSACryptoServiceProvider RSA = new RSACryptoServiceProvider();

//Save the public key information to an RSAParameters structure.

RSAParameters RSAKeyInfo = RSA.ExportParameters(false);

Storing Asymmetric Keys in a Key Container

Asymmetric private keys should never be stored verbatim or in plain text on the local computer. If you need to store a private key, you should use a key container. For more information on key containers, see the CryptoAPI section in the Platform SDK documentation at http://msdn.microsoft.com.

**To create an asymmetric key and save it in a key container**

1. Create a new instance of a [CspParameters](https://msdn.microsoft.com/en-us/library/system.security.cryptography.cspparameters(v=vs.71).aspx) class and pass the name that you want to call the key container to the[CspParameters.KeyContainerName](https://msdn.microsoft.com/en-us/library/system.security.cryptography.cspparameters.keycontainername(v=vs.71).aspx) field.
2. Create a new instance of a class that derives from the [AsymmetricAlgorithm](https://msdn.microsoft.com/en-us/library/system.security.cryptography.asymmetricalgorithm(v=vs.71).aspx) class (usually **RSACryptoServiceProvider** or**DSACryptoServiceProvider**) and pass the previously created **CspParameters** object to its constructor.

**To delete a key from a key container**

1. Create a new instance of a **CspParameters** class and pass the name that you want to call the key container to the**CspParameters.KeyContainerName** field.
2. Create a new instance of a class that derives from the **AsymmetricAlgorithm** class (usually **RSACryptoServiceProvider** or**DSACryptoServiceProvider**) and pass the previously created **CspParameters** object to its constructor.
3. Set the **PersistKeyInCSP** property of the class that derives from **AsymmetricAlgorithm** to **false** (**False** in Visual Basic).
4. Call the **Clear** method of the class that derives from **AsymmetricAlgorithm**. This method releases all resources of the class and clears the key container.

The following example demonstrates how to create an asymmetric key, save it in a key container, retrieve the key at a later time, and delete the key from the container.

VB

Imports System

Imports System.IO

Imports System.Security.Cryptography

\_

Public Class StoreKey

Public Shared Sub Main()

Try

' Create a key and save it in a container.

GenKey\_SaveInContainer("MyKeyContainer")

' Retrieve the key from the container.

GetKeyFromContainer("MyKeyContainer")

' Delete the key from the container.

DeleteKeyFromContainer("MyKeyContainer")

' Create a key and save it in a container.

GenKey\_SaveInContainer("MyKeyContainer")

' Delete the key from the container.

DeleteKeyFromContainer("MyKeyContainer")

Catch e As CryptographicException

Console.WriteLine(e.Message)

End Try

End Sub

Public Shared Sub GenKey\_SaveInContainer(ByVal ContainerName As String)

' Create the CspParameters object and set the key container

' name used to store the RSA key pair.

Dim cp As New CspParameters()

cp.KeyContainerName = ContainerName

' Create a new instance of RSACryptoServiceProvider that accesses

' the key container MyKeyContainerName.

Dim rsa As New RSACryptoServiceProvider(cp)

' Display the key information to the console.

Console.WriteLine("Key added to container: {0}", rsa.ToXmlString(True))

End Sub

Public Shared Sub GetKeyFromContainer(ByVal ContainerName As String)

' Create the CspParameters object and set the key container

' name used to store the RSA key pair.

Dim cp As New CspParameters()

cp.KeyContainerName = ContainerName

' Create a new instance of RSACryptoServiceProvider that accesses

' the key container MyKeyContainerName.

Dim rsa As New RSACryptoServiceProvider(cp)

' Display the key information to the console.

Console.WriteLine("Key retrieved from container : {0}", rsa.ToXmlString(True))

End Sub

Public Shared Sub DeleteKeyFromContainer(ByVal ContainerName As String)

' Create the CspParameters object and set the key container

' name used to store the RSA key pair.

Dim cp As New CspParameters()

cp.KeyContainerName = ContainerName

' Create a new instance of RSACryptoServiceProvider that accesses

' the key container.

Dim rsa As New RSACryptoServiceProvider(cp)

' Delete the key entry in the container.

rsa.PersistKeyInCsp = False

' Call Clear to release resources and delete the key from the container.

rsa.Clear()

Console.WriteLine("Key deleted.")

End Sub

End Class

[C#]

using System;

using System.IO;

using System.Security.Cryptography;

public class StoreKey

{

public static void Main()

{

try

{

// Create a key and save it in a container.

GenKey\_SaveInContainer("MyKeyContainer");

// Retrieve the key from the container.

GetKeyFromContainer("MyKeyContainer");

// Delete the key from the container.

DeleteKeyFromContainer("MyKeyContainer");

// Create a key and save it in a container.

GenKey\_SaveInContainer("MyKeyContainer");

// Delete the key from the container.

DeleteKeyFromContainer("MyKeyContainer");

}

catch(CryptographicException e)

{

Console.WriteLine(e.Message);

}

}

public static void GenKey\_SaveInContainer(string ContainerName)

{

// Create the CspParameters object and set the key container

// name used to store the RSA key pair.

CspParameters cp = new CspParameters();

cp.KeyContainerName = ContainerName;

// Create a new instance of RSACryptoServiceProvider that accesses

// the key container MyKeyContainerName.

RSACryptoServiceProvider rsa = new RSACryptoServiceProvider(cp);

// Display the key information to the console.

Console.WriteLine("Key added to container: \n {0}", rsa.ToXmlString(true));

}

public static void GetKeyFromContainer(string ContainerName)

{

// Create the CspParameters object and set the key container

// name used to store the RSA key pair.

CspParameters cp = new CspParameters();

cp.KeyContainerName = ContainerName;

// Create a new instance of RSACryptoServiceProvider that accesses

// the key container MyKeyContainerName.

RSACryptoServiceProvider rsa = new RSACryptoServiceProvider(cp);

// Display the key information to the console.

Console.WriteLine("Key retrieved from container : \n {0}", rsa.ToXmlString(true));

}

public static void DeleteKeyFromContainer(string ContainerName)

{

// Create the CspParameters object and set the key container

// name used to store the RSA key pair.

CspParameters cp = new CspParameters();

cp.KeyContainerName = ContainerName;

// Create a new instance of RSACryptoServiceProvider that accesses

// the key container.

RSACryptoServiceProvider rsa = new RSACryptoServiceProvider(cp);

// Delete the key entry in the container.

rsa.PersistKeyInCsp = false;

// Call Clear to release resources and delete the key from the container.

rsa.Clear();

Console.WriteLine("Key deleted.");

}

}

When the previous example is run, the following is displayed to the console.

Key added to container:

<RSAKeyValue> ...Key Information A...</RSAKeyValue>

Key retrieved from container :

<RSAKeyValue> ...Key Information A...</RSAKeyValue>

Key deleted.

Key added to container:

<RSAKeyValue> ...Key Information B...</RSAKeyValue>

Key deleted.

##### Generating Keys for Encryption and Decryption

**.NET Framework 2.0, 3.0, 3.5, 4, 4.5, 4.6**

Creating and managing keys is an important part of the cryptographic process. Symmetric algorithms require the creation of a key and an initialization vector (IV) that must be kept secret from anyone who should not decrypt your data. Asymmetric algorithms require the creation of a public key and a private key. The public key can be made public to anyone, while the private key must known only by the party who will decrypt the data encrypted with the public key. This section describes how to generate and manage keys for both symmetric and asymmetric algorithms.

**Symmetric Keys**

The symmetric encryption classes supplied by the .NET Framework require a key and a new initialization vector (IV) to encrypt and decrypt data. Whenever you create a new instance of one of the managed symmetric cryptographic classes using the default constructor, a new key and IV are automatically created. Anyone that you allow to decrypt your data must possess the same key and IV and use the same algorithm. Generally, a new key and IV should be created for every session, and neither the key nor IV should be stored for use in a later session.

To communicate a symmetric key and IV to a remote party, you would usually encrypt the symmetric key and IV using asymmetric encryption. Sending these values across an insecure network without encrypting them is extremely unsafe, as anyone that intercepts these values can then decrypt your data. For more information on this process of encrypting and transferring the key and IV, see [Creating a Cryptographic Scheme](https://msdn.microsoft.com/en-us/library/0cwc0x23(v=vs.80).aspx).

The following example shows the creation of a new instance of the [TripleDESCryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.tripledescryptoserviceprovider(v=vs.80).aspx) class that implements the TripleDES algorithm.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/5e9ft273(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

TripleDESCryptoServiceProvider TDES = new TripleDESCryptoServiceProvider();

When the previous code is executed, a new key and IV are generated and placed in the **Key** and **IV** properties, respectively.

Sometimes you might need to generate multiple keys. In this situation, you can create a new instance of a class that implements a symmetric algorithm and then create a new key and IV by calling the **GenerateKey** and **GenerateIV** methods. The following code example illustrates how to create new keys and IVs after a new instance of the asymmetric cryptographic class has been made.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/5e9ft273(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

TripleDESCryptoServiceProvider TDES = new TripleDESCryptoServiceProvider();

TDES.GenerateIV();

TDES.GenerateKey();

When the previous code is executed, a key and IV are generated when the new instance of **TripleDESCryptoServiceProvider** is made. Another key and IV are created when the **GenerateKey** and **GenerateIV** methods are called.

**Asymmetric Keys**

The .NET Framework provides the [RSACryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsacryptoserviceprovider(v=vs.80).aspx) and [DSACryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.dsacryptoserviceprovider(v=vs.80).aspx) classes for asymmetric encryption. These classes create a public/private key pair when you use the default constructor to create a new instance. Asymmetric keys can be either stored for use in multiple sessions or generated for one session only. While the public key can be made generally available, the private key should be closely guarded.

A public/private key pair is generated whenever a new instance of an asymmetric algorithm class is created. After a new instance of the class is created, the key information can be extracted using one of two methods:

* The [ToXMLString](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsa.toxmlstring(v=vs.80).aspx) method, which returns an XML representation of the key information.
* The [ExportParameters](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsacryptoserviceprovider.exportparameters(v=vs.80).aspx) method, which returns an [RSAParameters](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsaparameters(v=vs.80).aspx) structure that holds the key information.

Both methods accept a Boolean value that indicates whether to return only the public key information or to return both the public-key and the private-key information. An **RSACryptoServiceProvider** class can be initialized to the value of an **RSAParameters** structure by using the[ImportParameters](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsacryptoserviceprovider.importparameters(v=vs.80).aspx) method.

Asymmetric private keys should never be stored verbatim or in plain text on the local computer. If you need to store a private key, you should use a key container. For more on how to store a private key in a key container, see [How to: Store Asymmetric Keys in a Key Container](https://msdn.microsoft.com/en-us/library/tswxhw92(v=vs.80).aspx).

The following code example creates a new instance of the **RSACryptoServiceProvider**class, creating a public/private key pair, and saves the public key information to an **RSAParameters**structure.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/5e9ft273(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-3)

//Generate a public/private key pair.

RSACryptoServiceProvider RSA = new RSACryptoServiceProvider();

//Save the public key information to an RSAParameters structure.

RSAParameters RSAKeyInfo = RSA.ExportParameters(false);

##### Encrypting Data

**.NET Framework 1.1**

Symmetric encryption and asymmetric encryption are performed using different processes. Symmetric encryption is performed on streams and is therefore useful to encrypt large amounts of data. Asymmetric encryption is performed on a small number of bytes and is therefore useful only for small amounts of data.

**Symmetric Encryption**

The managed symmetric cryptography classes are used with a special stream class called a [CryptoStream](https://msdn.microsoft.com/en-us/library/system.security.cryptography.cryptostream(v=vs.71).aspx) that encrypts data read into the stream. The **CryptoStream** class is initialized with a managed stream class, a class implements the [ICryptoTransform](https://msdn.microsoft.com/en-us/library/system.security.cryptography.icryptotransform(v=vs.71).aspx) interface (created from a class that implements a cryptographic algorithm), and a [CryptoStreamMode](https://msdn.microsoft.com/en-us/library/system.security.cryptography.cryptostreammode(v=vs.71).aspx) enumeration that describes the type of access permitted to the**CryptoStream**. The **CryptoStream** class can be initialized using any class that derives from the [Stream](https://msdn.microsoft.com/en-us/library/system.io.stream(v=vs.71).aspx) class, including [FileStream](https://msdn.microsoft.com/en-us/library/system.io.filestream(v=vs.71).aspx),[MemoryStream](https://msdn.microsoft.com/en-us/library/system.io.memorystream(v=vs.71).aspx), and [NetworkStream](https://msdn.microsoft.com/en-us/library/system.net.sockets.networkstream(v=vs.71).aspx). Using these classes, you can perform symmetric encryption on a variety of stream objects.

The following example illustrates how to create a new instance of the [RijndaelManaged](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rijndaelmanaged(v=vs.71).aspx) class, which implements the Rijndael encryption algorithm, and use it to perform encryption on a **CryptoStream** class. In this example, the **CryptoStream** is initialized with a stream object called MyStream that can be any type of managed stream. The **CreateEncryptor** method from the **RijndaelManaged**classis passed the key and IV that are used for encryption. In this case, the default key and IV generated from RMCrypto are used. Finally, the**CryptoStreamMode.Write** is passed, specifying write access to the stream.

VB

Dim RMCrypto As New RijndaelManaged()

Dim CryptStream As New CryptoStream(MyStream, RMCrypto.CreateEncryptor(RMCrypto.Key, RMCrypto.IV), CryptoStreamMode.Write)

[C#]

RijndaelManaged RMCrypto = new RijndaelManaged();

CryptoStream CryptStream = new CryptoStream(MyStream, RMCrypto.CreateEncryptor(), CryptoStreamMode.Write);

After this code is executed, any data written to the **CryptoStream** object is encrypted using the Rijndael algorithm.

The following example shows the entire process of creating a stream, encrypting the stream, writing to the stream, and closing the stream. This example creates a network stream that is encrypted using the **CryptoStream** class and the **RijndaelManaged** class. A message is written to the encrypted stream with the [StreamWriter](https://msdn.microsoft.com/en-us/library/system.io.streamwriter(v=vs.71).aspx) class.

VB

Imports System

Imports System.IO

Imports System.Security.Cryptography

Imports System.Net.Sockets

Module Module1

Sub Main()

Try

'Create a TCP connection to a listening TCP process.

'Use "localhost" to specify the current computer or

'replace "localhost" with the IP address of the

'listening process.

Dim TCP As New TcpClient("localhost", 11000)

'Create a network stream from the TCP connection.

Dim NetStream As NetworkStream = TCP.GetStream()

'Create a new instance of the RijndaelManaged class

'and encrypt the stream.

Dim RMCrypto As New RijndaelManaged()

Dim Key As Byte() = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16}

Dim IV As Byte() = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16}

'Create a CryptoStream, pass it the NetworkStream, and encrypt

'it with the Rijndael class.

Dim CryptStream As New CryptoStream(NetStream, RMCrypto.CreateEncryptor(Key, IV), CryptoStreamMode.Write)

'Create a StreamWriter for easy writing to the

'network stream.

Dim SWriter As New StreamWriter(CryptStream)

'Write to the stream.

SWriter.WriteLine("Hello World!")

'Inform the user that the message was written

'to the stream.

Console.WriteLine("The message was sent.")

'Close all the connections.

SWriter.Close()

CryptStream.Close()

NetStream.Close()

TCP.Close()

Catch

'Inform the user that an exception was raised.

Console.WriteLine("The connection failed.")

End Try

End Sub

End Module

[C#]

using System;

using System.IO;

using System.Security.Cryptography;

using System.Net.Sockets;

public class main

{

public static void Main(string[] args)

{

try

{

//Create a TCP connection to a listening TCP process.

//Use "localhost" to specify the current computer or

//replace "localhost" with the IP address of the

//listening process.

TcpClient TCP = new TcpClient("localhost",11000);

//Create a network stream from the TCP connection.

NetworkStream NetStream = TCP.GetStream();

//Create a new instance of the RijndaelManaged class

// and encrypt the stream.

RijndaelManaged RMCrypto = new RijndaelManaged();

byte[] Key = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16};

byte[] IV = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16};

//Create a CryptoStream, pass it the NetworkStream, and encrypt

//it with the Rijndael class.

CryptoStream CryptStream = new CryptoStream(NetStream,

RMCrypto.CreateEncryptor(Key, IV),

CryptoStreamMode.Write);

//Create a StreamWriter for easy writing to the

//network stream.

StreamWriter SWriter = new StreamWriter(CryptStream);

//Write to the stream.

SWriter.WriteLine("Hello World!");

//Inform the user that the message was written

//to the stream.

Console.WriteLine("The message was sent.");

//Close all the connections.

SWriter.Close();

CryptStream.Close();

NetStream.Close();

TCP.Close();

}

catch

{

//Inform the user that an exception was raised.

Console.WriteLine("The connection failed.");

}

}

}

For the previous example to execute successfully, there must be a process listening on the IP address and port number specified in the[TCPCLient](https://msdn.microsoft.com/en-us/library/system.net.sockets.tcpclient(v=vs.71).aspx) class. If a listening process exists, the code will connect to the listening process, encrypt the stream using the Rijndael symmetric algorithm, and write "Hello World!" to the stream. If the code is successful, it displays the following text to the console:

The message was sent.

However, if no listening process is found or an exception is raised, the code displays the following text to the console:

The connection failed.

**Asymmetric Encryption**

Asymmetric algorithms are usually used to encrypt small amounts of data such as the encryption of a symmetric key and IV. Typically, an individual performing asymmetric encryption uses the public key generated by another party. The [RSACryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsacryptoserviceprovider(v=vs.71).aspx) class is provided by the .NET Framework for this purpose.

The following example uses public key information to encrypt a symmetric key and IV. Two byte arrays are initialized that represent the public key of a third party. An [RSAParameters](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsaparameters(v=vs.71).aspx) object is initialized to these values. Next, the **RSAParameters** object (along with the public key it represents) is imported into an **RSACryptoServiceProvider** using the [RSACryptoServiceProvider.ImportParameters](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsacryptoserviceprovider.importparameters(v=vs.71).aspx) method. Finally, the private key and IV created by a [RijndaelManaged](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rijndaelmanaged(v=vs.71).aspx) class are encrypted. This example requires systems to have 128-bit encryption installed.

VB

Imports System

Imports System.Security.Cryptography

Module Module1

Sub Main()

'Initialize the byte arrays to the public key information.

Dim PublicKey As Byte() = {214, 46, 220, 83, 160, 73, 40, 39, 201, 155, 19,202, 3, 11, 191, 178, 56, 74, 90, 36, 248, 103, 18, 144, 170, 163, 145, 87, 54, 61, 34, 220, 222, 207, 137, 149, 173, 14, 92, 120, 206, 222, 158, 28, 40, 24, 30, 16, 175, 108, 128, 35, 230, 118, 40, 121, 113, 125, 216, 130, 11, 24, 90, 48, 194, 240, 105, 44, 76, 34, 57, 249, 228, 125, 80, 38, 9, 136, 29, 117, 207, 139, 168, 181, 85, 137, 126, 10, 126, 242, 120, 247, 121, 8, 100, 12, 201, 171, 38, 226, 193, 180, 190, 117, 177, 87, 143, 242, 213, 11, 44, 180, 113, 93, 106, 99, 179, 68, 175, 211, 164, 116, 64, 148, 226, 254, 172, 147}

Dim Exponent As Byte() = {1, 0, 1}

'Create values to store encrypted symmetric keys.

Dim EncryptedSymmetricKey() As Byte

Dim EncryptedSymmetricIV() As Byte

'Create a new instance of the RSACryptoServiceProvider class.

Dim RSA As New RSACryptoServiceProvider()

'Create a new instance of the RSAParameters structure.

Dim RSAKeyInfo As New RSAParameters()

'Set RSAKeyInfo to the public key values.

RSAKeyInfo.Modulus = PublicKey

RSAKeyInfo.Exponent = Exponent

'Import key parameters into RSA.

RSA.ImportParameters(RSAKeyInfo)

'Create a new instance of the RijndaelManaged class.

Dim RM As New RijndaelManaged()

'Encrypt the symmetric key and IV.

EncryptedSymmetricKey = RSA.Encrypt(RM.Key, False)

EncryptedSymmetricIV = RSA.Encrypt(RM.IV, False)

End Sub

End Module

[C#]

using System;

using System.Security.Cryptography;

class Class1

{

static void Main()

{

//Initialize the byte arrays to the public key information.

byte[] PublicKey = {214,46,220,83,160,73,40,39,201,155,19,202,3,11,191,178,56,

74,90,36,248,103,18,144,170,163,145,87,54,61,34,220,222,

207,137,149,173,14,92,120,206,222,158,28,40,24,30,16,175,

108,128,35,230,118,40,121,113,125,216,130,11,24,90,48,194,

240,105,44,76,34,57,249,228,125,80,38,9,136,29,117,207,139,

168,181,85,137,126,10,126,242,120,247,121,8,100,12,201,171,

38,226,193,180,190,117,177,87,143,242,213,11,44,180,113,93,

106,99,179,68,175,211,164,116,64,148,226,254,172,147};

byte[] Exponent = {1,0,1};

//Create values to store encrypted symmetric keys.

byte[] EncryptedSymmetricKey;

byte[] EncryptedSymmetricIV;

//Create a new instance of the RSACryptoServiceProvider class.

RSACryptoServiceProvider RSA = new RSACryptoServiceProvider();

//Create a new instance of the RSAParameters structure.

RSAParameters RSAKeyInfo = new RSAParameters();

//Set RSAKeyInfo to the public key values.

RSAKeyInfo.Modulus = PublicKey;

RSAKeyInfo.Exponent = Exponent;

//Import key parameters into RSA.

RSA.ImportParameters(RSAKeyInfo);

//Create a new instance of the RijndaelManaged class.

RijndaelManaged RM = new RijndaelManaged();

//Encrypt the symmetric key and IV.

EncryptedSymmetricKey = RSA.Encrypt(RM.Key, false);

EncryptedSymmetricIV = RSA.Encrypt(RM.IV, false);

}

}

##### Encrypting Data

**.NET Framework 2.0, 3.0**

Symmetric encryption and asymmetric encryption are performed using different processes. Symmetric encryption is performed on streams and is therefore useful to encrypt large amounts of data. Asymmetric encryption is performed on a small number of bytes and is therefore useful only for small amounts of data.

**Symmetric Encryption**

The managed symmetric cryptography classes are used with a special stream class called a [CryptoStream](https://msdn.microsoft.com/en-us/library/system.security.cryptography.cryptostream(v=vs.80).aspx) that encrypts data read into the stream. The **CryptoStream** class is initialized with a managed stream class, a class implements the [ICryptoTransform](https://msdn.microsoft.com/en-us/library/system.security.cryptography.icryptotransform(v=vs.80).aspx) interface (created from a class that implements a cryptographic algorithm), and a [CryptoStreamMode](https://msdn.microsoft.com/en-us/library/system.security.cryptography.cryptostreammode(v=vs.80).aspx) enumeration that describes the type of access permitted to the**CryptoStream**. The **CryptoStream** class can be initialized using any class that derives from the [Stream](https://msdn.microsoft.com/en-us/library/system.io.stream(v=vs.80).aspx) class, including [FileStream](https://msdn.microsoft.com/en-us/library/system.io.filestream(v=vs.80).aspx),[MemoryStream](https://msdn.microsoft.com/en-us/library/system.io.memorystream(v=vs.80).aspx), and [NetworkStream](https://msdn.microsoft.com/en-us/library/system.net.sockets.networkstream(v=vs.80).aspx). Using these classes, you can perform symmetric encryption on a variety of stream objects.

The following example illustrates how to create a new instance of the [RijndaelManaged](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rijndaelmanaged(v=vs.80).aspx) class, which implements the Rijndael encryption algorithm, and use it to perform encryption on a **CryptoStream** class. In this example, the **CryptoStream** is initialized with a stream object called MyStream that can be any type of managed stream. The **CreateEncryptor** method from the **RijndaelManaged**class is passed the key and IV that are used for encryption. In this case, the default key and IV generated from RMCrypto are used. Finally, the**CryptoStreamMode.Write** is passed, specifying write access to the stream.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/as0w18af(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

RijndaelManaged RMCrypto = new RijndaelManaged();

CryptoStream CryptStream = new CryptoStream(MyStream, RMCrypto.CreateEncryptor(), CryptoStreamMode.Write);

After this code is executed, any data written to the **CryptoStream** object is encrypted using the Rijndael algorithm.

The following example shows the entire process of creating a stream, encrypting the stream, writing to the stream, and closing the stream. This example creates a network stream that is encrypted using the **CryptoStream** class and the **RijndaelManaged** class. A message is written to the encrypted stream with the [StreamWriter](https://msdn.microsoft.com/en-us/library/system.io.streamwriter(v=vs.80).aspx) class.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/as0w18af(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

using System;

using System.IO;

using System.Security.Cryptography;

using System.Net.Sockets;

public class main

{

public static void Main(string[] args)

{

try

{

//Create a TCP connection to a listening TCP process.

//Use "localhost" to specify the current computer or

//replace "localhost" with the IP address of the

//listening process.

TcpClient TCP = new TcpClient("localhost",11000);

//Create a network stream from the TCP connection.

NetworkStream NetStream = TCP.GetStream();

//Create a new instance of the RijndaelManaged class

// and encrypt the stream.

RijndaelManaged RMCrypto = new RijndaelManaged();

byte[] Key = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16};

byte[] IV = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16};

//Create a CryptoStream, pass it the NetworkStream, and encrypt

//it with the Rijndael class.

CryptoStream CryptStream = new CryptoStream(NetStream,

RMCrypto.CreateEncryptor(Key, IV),

CryptoStreamMode.Write);

//Create a StreamWriter for easy writing to the

//network stream.

StreamWriter SWriter = new StreamWriter(CryptStream);

//Write to the stream.

SWriter.WriteLine("Hello World!");

//Inform the user that the message was written

//to the stream.

Console.WriteLine("The message was sent.");

//Close all the connections.

SWriter.Close();

CryptStream.Close();

NetStream.Close();

TCP.Close();

}

catch

{

//Inform the user that an exception was raised.

Console.WriteLine("The connection failed.");

}

}

}

For the previous example to execute successfully, there must be a process listening on the IP address and port number specified in the[TCPCLient](https://msdn.microsoft.com/en-us/library/system.net.sockets.tcpclient(v=vs.80).aspx) class. If a listening process exists, the code will connect to the listening process, encrypt the stream using the Rijndael symmetric algorithm, and write "Hello World!" to the stream. If the code is successful, it displays the following text to the console:

The message was sent.

However, if no listening process is found or an exception is raised, the code displays the following text to the console:

The connection failed.

**Asymmetric Encryption**

Asymmetric algorithms are usually used to encrypt small amounts of data such as the encryption of a symmetric key and IV. Typically, an individual performing asymmetric encryption uses the public key generated by another party. The [RSACryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsacryptoserviceprovider(v=vs.80).aspx) class is provided by the .NET Framework for this purpose.

The following example uses public key information to encrypt a symmetric key and IV. Two byte arrays are initialized that represent the public key of a third party. An [RSAParameters](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsaparameters(v=vs.80).aspx) object is initialized to these values. Next, the **RSAParameters** object (along with the public key it represents) is imported into an **RSACryptoServiceProvider** using the [RSACryptoServiceProvider.ImportParameters](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsacryptoserviceprovider.importparameters(v=vs.80).aspx) method. Finally, the private key and IV created by a [RijndaelManaged](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rijndaelmanaged(v=vs.80).aspx) class are encrypted. This example requires systems to have 128-bit encryption installed.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/as0w18af(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-5)

using System;

using System.Security.Cryptography;

class Class1

{

static void Main()

{

//Initialize the byte arrays to the public key information.

byte[] PublicKey = {214,46,220,83,160,73,40,39,201,155,19,202,3,11,191,178,56,

74,90,36,248,103,18,144,170,163,145,87,54,61,34,220,222,

207,137,149,173,14,92,120,206,222,158,28,40,24,30,16,175,

108,128,35,230,118,40,121,113,125,216,130,11,24,90,48,194,

240,105,44,76,34,57,249,228,125,80,38,9,136,29,117,207,139,

168,181,85,137,126,10,126,242,120,247,121,8,100,12,201,171,

38,226,193,180,190,117,177,87,143,242,213,11,44,180,113,93,

106,99,179,68,175,211,164,116,64,148,226,254,172,147};

byte[] Exponent = {1,0,1};

//Create values to store encrypted symmetric keys.

byte[] EncryptedSymmetricKey;

byte[] EncryptedSymmetricIV;

//Create a new instance of the RSACryptoServiceProvider class.

RSACryptoServiceProvider RSA = new RSACryptoServiceProvider();

//Create a new instance of the RSAParameters structure.

RSAParameters RSAKeyInfo = new RSAParameters();

//Set RSAKeyInfo to the public key values.

RSAKeyInfo.Modulus = PublicKey;

RSAKeyInfo.Exponent = Exponent;

//Import key parameters into RSA.

RSA.ImportParameters(RSAKeyInfo);

//Create a new instance of the RijndaelManaged class.

RijndaelManaged RM = new RijndaelManaged();

//Encrypt the symmetric key and IV.

EncryptedSymmetricKey = RSA.Encrypt(RM.Key, false);

EncryptedSymmetricIV = RSA.Encrypt(RM.IV, false);

}

}

##### Encrypting Data

**.NET Framework 3.5, 4, 4.5, 4.6**

Updated: January 2010

Symmetric encryption and asymmetric encryption are performed using different processes. Symmetric encryption is performed on streams and is therefore useful to encrypt large amounts of data. Asymmetric encryption is performed on a small number of bytes and is therefore useful only for small amounts of data.

[**Symmetric Encryption**](javascript:void(0))

The managed symmetric cryptography classes are used with a special stream class called a [CryptoStream](https://msdn.microsoft.com/en-us/library/system.security.cryptography.cryptostream(v=vs.90).aspx) that encrypts data read into the stream. The **CryptoStream** class is initialized with a managed stream class, a class implements the [ICryptoTransform](https://msdn.microsoft.com/en-us/library/system.security.cryptography.icryptotransform(v=vs.90).aspx) interface (created from a class that implements a cryptographic algorithm), and a [CryptoStreamMode](https://msdn.microsoft.com/en-us/library/system.security.cryptography.cryptostreammode(v=vs.90).aspx) enumeration that describes the type of access permitted to the **CryptoStream**. The **CryptoStream** class can be initialized using any class that derives from the [Stream](https://msdn.microsoft.com/en-us/library/system.io.stream(v=vs.90).aspx) class, including [FileStream](https://msdn.microsoft.com/en-us/library/system.io.filestream(v=vs.90).aspx),[MemoryStream](https://msdn.microsoft.com/en-us/library/system.io.memorystream(v=vs.90).aspx), and [NetworkStream](https://msdn.microsoft.com/en-us/library/system.net.sockets.networkstream(v=vs.90).aspx). Using these classes, you can perform symmetric encryption on a variety of stream objects.

The following example illustrates how to create a new instance of the [RijndaelManaged](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rijndaelmanaged(v=vs.90).aspx) class, which implements the Rijndael encryption algorithm, and use it to perform encryption on a **CryptoStream** class. In this example, the **CryptoStream** is initialized with a stream object called MyStream that can be any type of managed stream. The **CreateEncryptor** method from the **RijndaelManaged**class is passed the key and IV that are used for encryption. In this case, the default key and IV generated from RMCrypto are used. Finally, the**CryptoStreamMode.Write** is passed, specifying write access to the stream.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/as0w18af(v=vs.90).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

RijndaelManaged RMCrypto = new RijndaelManaged();

CryptoStream CryptStream = new CryptoStream(MyStream, RMCrypto.CreateEncryptor(), CryptoStreamMode.Write);

After this code is executed, any data written to the **CryptoStream** object is encrypted using the Rijndael algorithm.

The following example shows the entire process of creating a stream, encrypting the stream, writing to the stream, and closing the stream. This example creates a network stream that is encrypted using the **CryptoStream** class and the **RijndaelManaged** class. A message is written to the encrypted stream with the [StreamWriter](https://msdn.microsoft.com/en-us/library/system.io.streamwriter(v=vs.90).aspx) class.

|  |
| --- |
| **NoteNote:** |
| You can also use this example to write to a file. To do that, delete the [TcpClient](https://msdn.microsoft.com/en-us/library/system.net.sockets.tcpclient(v=vs.90).aspx) reference and replace the [NetworkStream](https://msdn.microsoft.com/en-us/library/system.net.sockets.networkstream(v=vs.90).aspx) with a[FileStream](https://msdn.microsoft.com/en-us/library/system.io.filestream(v=vs.90).aspx). |

C#

[**VB**](https://msdn.microsoft.com/en-us/library/as0w18af(v=vs.90).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

using System;

using System.IO;

using System.Security.Cryptography;

using System.Net.Sockets;

public class main

{

public static void Main(string[] args)

{

try

{

//Create a TCP connection to a listening TCP process.

//Use "localhost" to specify the current computer or

//replace "localhost" with the IP address of the

//listening process.

TcpClient TCP = new TcpClient("localhost",11000);

//Create a network stream from the TCP connection.

NetworkStream NetStream = TCP.GetStream();

//Create a new instance of the RijndaelManaged class

// and encrypt the stream.

RijndaelManaged RMCrypto = new RijndaelManaged();

byte[] Key = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16};

byte[] IV = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16};

//Create a CryptoStream, pass it the NetworkStream, and encrypt

//it with the Rijndael class.

CryptoStream CryptStream = new CryptoStream(NetStream,

RMCrypto.CreateEncryptor(Key, IV),

CryptoStreamMode.Write);

//Create a StreamWriter for easy writing to the

//network stream.

StreamWriter SWriter = new StreamWriter(CryptStream);

//Write to the stream.

SWriter.WriteLine("Hello World!");

//Inform the user that the message was written

//to the stream.

Console.WriteLine("The message was sent.");

//Close all the connections.

SWriter.Close();

CryptStream.Close();

NetStream.Close();

TCP.Close();

}

catch

{

//Inform the user that an exception was raised.

Console.WriteLine("The connection failed.");

}

}

}

For the previous example to execute successfully, there must be a process listening on the IP address and port number specified in the[TCPCLient](https://msdn.microsoft.com/en-us/library/system.net.sockets.tcpclient(v=vs.90).aspx) class. If a listening process exists, the code will connect to the listening process, encrypt the stream using the Rijndael symmetric algorithm, and write "Hello World!" to the stream. If the code is successful, it displays the following text to the console:

The message was sent.

However, if no listening process is found or an exception is raised, the code displays the following text to the console:

The connection failed.

[**Asymmetric Encryption**](javascript:void(0))

Asymmetric algorithms are usually used to encrypt small amounts of data such as the encryption of a symmetric key and IV. Typically, an individual performing asymmetric encryption uses the public key generated by another party. The [RSACryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsacryptoserviceprovider(v=vs.90).aspx) class is provided by the .NET Framework for this purpose.

The following example uses public key information to encrypt a symmetric key and IV. Two byte arrays are initialized that represent the public key of a third party. An [RSAParameters](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsaparameters(v=vs.90).aspx) object is initialized to these values. Next, the **RSAParameters** object (along with the public key it represents) is imported into an **RSACryptoServiceProvider** using the [RSACryptoServiceProvider.ImportParameters](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsacryptoserviceprovider.importparameters(v=vs.90).aspx) method. Finally, the private key and IV created by a [RijndaelManaged](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rijndaelmanaged(v=vs.90).aspx) class are encrypted. This example requires systems to have 128-bit encryption installed.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/as0w18af(v=vs.90).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-5)

using System;

using System.Security.Cryptography;

class Class1

{

static void Main()

{

//Initialize the byte arrays to the public key information.

byte[] PublicKey = {214,46,220,83,160,73,40,39,201,155,19,202,3,11,191,178,56,

74,90,36,248,103,18,144,170,163,145,87,54,61,34,220,222,

207,137,149,173,14,92,120,206,222,158,28,40,24,30,16,175,

108,128,35,230,118,40,121,113,125,216,130,11,24,90,48,194,

240,105,44,76,34,57,249,228,125,80,38,9,136,29,117,207,139,

168,181,85,137,126,10,126,242,120,247,121,8,100,12,201,171,

38,226,193,180,190,117,177,87,143,242,213,11,44,180,113,93,

106,99,179,68,175,211,164,116,64,148,226,254,172,147};

byte[] Exponent = {1,0,1};

//Create values to store encrypted symmetric keys.

byte[] EncryptedSymmetricKey;

byte[] EncryptedSymmetricIV;

//Create a new instance of the RSACryptoServiceProvider class.

RSACryptoServiceProvider RSA = new RSACryptoServiceProvider();

//Create a new instance of the RSAParameters structure.

RSAParameters RSAKeyInfo = new RSAParameters();

//Set RSAKeyInfo to the public key values.

RSAKeyInfo.Modulus = PublicKey;

RSAKeyInfo.Exponent = Exponent;

//Import key parameters into RSA.

RSA.ImportParameters(RSAKeyInfo);

//Create a new instance of the RijndaelManaged class.

RijndaelManaged RM = new RijndaelManaged();

//Encrypt the symmetric key and IV.

EncryptedSymmetricKey = RSA.Encrypt(RM.Key, false);

EncryptedSymmetricIV = RSA.Encrypt(RM.IV, false);

}

}

[**See Also**](javascript:void(0))

Concepts

[Generating Keys for Encryption and Decryption](https://msdn.microsoft.com/en-us/library/5e9ft273(v=vs.90).aspx)

[Decrypting Data](https://msdn.microsoft.com/en-us/library/te15te69(v=vs.90).aspx)

Other Resources

[Cryptographic Tasks](https://msdn.microsoft.com/en-us/library/7yx4d854(v=vs.90).aspx)

[Cryptographic Services](https://msdn.microsoft.com/en-us/library/93bskf9z(v=vs.90).aspx)

[**Change History**](javascript:void(0))

|  |  |  |
| --- | --- | --- |
| **Date** | **History** | **Reason** |
| January 2010 | Added information about using the example to write to a file. | Customer feedback. |

##### Decrypting Data

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4, 4.5, 4.6**

Decryption is the reverse operation of encryption. For secret-key encryption, you must know both the key and IV that were used to encrypt the data. For public-key encryption, you must know either the public key (if the data was encrypted using the private key) or the private key (if the data was encrypted using the public key).

**Symmetric Decryption**

The decryption of data encrypted with symmetric algorithms is similar to the process used to encrypt data with symmetric algorithms. The[CryptoStream](https://msdn.microsoft.com/en-us/library/system.security.cryptography.cryptostream(v=vs.71).aspx)class is used with symmetric cryptography classes provided by the .NET Framework to decrypt data read from any managed stream object.

The following example illustrates how to create a new instance of the [RijndaelManaged](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rijndaelmanaged(v=vs.71).aspx) class and use it to perform decryption on a[CryptoStream](https://msdn.microsoft.com/en-us/library/system.security.cryptography.cryptostream(v=vs.71).aspx) object. This example first creates a new instance of the **RijndaelManaged** class. Next it creates a **CryptoStream** object and initializes it to the value of a managed stream called MyStream. Next, the **CreateDecryptor** method from the **RijndaelManaged**classis passed the same key and IV that was used for encryption and is then passed to the **CryptoStream** constructor. Finally, the**CryptoStreamMode.Read** enumeration is passed to the **CryptoStream** constructor to specify read access to the stream.

VB

Dim RMCrypto As New RijndaelManaged()

Dim CryptStream As New CryptoStream(MyStream, RMCrypto.CreateDecryptor(RMCrypto.Key, RMCrypto.IV), CryptoStreamMode.Read)

[C#]

RijndaelManaged RMCrypto = new RijndaelManaged();

CryptoStream CryptStream = new CryptoStream(MyStream, RMCrypto.CreateDecryptor(Key, IV), CryptoStreamMode.Read);

The following example shows the entire process of creating a stream, decrypting the stream, reading from the stream, and closing the streams. A [TCPListener](https://msdn.microsoft.com/en-us/library/system.net.sockets.tcplistener(v=vs.71).aspx) object is created that initializes a network stream when a connection to the listening object is made. The network stream is then decrypted using the **CryptoStream** class and the **RijndaelManaged** class. This example assumes that the key and IV values have been either safely transferred or previously agreed upon. It does not show the code needed to encrypt and transfer these values.

VB

Imports System

Imports System.Net.Sockets

Imports System.Threading

Imports System.IO

Imports System.Security.Cryptography

Module Module1

Sub Main()

'The key and IV must be the same values that were used

'to encrypt the stream.

Dim Key As Byte() = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16}

Dim IV As Byte() = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16}

Try

'Initialize a TCPListener on port 11000

'using the current IP address.

Dim TCPListen As New TcpListener(11000)

'Start the listener.

TCPListen.Start()

'Check for a connection every five seconds.

While Not TCPListen.Pending()

Console.WriteLine("Still listening. Will try in 5 seconds.")

Thread.Sleep(5000)

End While

'Accept the client if one is found.

Dim TCP As TcpClient = TCPListen.AcceptTcpClient()

'Create a network stream from the connection.

Dim NetStream As NetworkStream = TCP.GetStream()

'Create a new instance of the RijndaelManaged class

'and decrypt the stream.

Dim RMCrypto As New RijndaelManaged()

'Create an instance of the CryptoStream class, pass it the NetworkStream, and decrypt

'it with the Rijndael class using the key and IV.

Dim CryptStream As New CryptoStream(NetStream, RMCrypto.CreateDecryptor(Key, IV), CryptoStreamMode.Read)

'Read the stream.

Dim SReader As New StreamReader(CryptStream)

'Display the message.

Console.WriteLine("The decrypted original message: {0}", SReader.ReadToEnd())

'Close the streams.

SReader.Close()

NetStream.Close()

TCP.Close()

'Catch any exceptions.

Catch

Console.WriteLine("The Listener Failed.")

End Try

End Sub

End Module

[C#]

using System;

using System.Net.Sockets;

using System.Threading;

using System.IO;

using System.Security.Cryptography;

class Class1

{

static void Main(string[] args)

{

//The key and IV must be the same values that were used

//to encrypt the stream.

byte[] Key = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16};

byte[] IV = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16};

try

{

//Initialize a TCPListener on port 11000

//using the current IP address.

TcpListener TCPListen = new TcpListener(11000);

//Start the listener.

TCPListen.Start();

//Check for a connection every five seconds.

while(!TCPListen.Pending())

{

Console.WriteLine("Still listening. Will try in 5 seconds.");

Thread.Sleep(5000);

}

//Accept the client if one is found.

TcpClient TCP = TCPListen.AcceptTcpClient();

//Create a network stream from the connection.

NetworkStream NetStream = TCP.GetStream();

//Create a new instance of the RijndaelManaged class

// and decrypt the stream.

RijndaelManaged RMCrypto = new RijndaelManaged();

//Create a CryptoStream, pass it the NetworkStream, and decrypt

//it with the Rijndael class using the key and IV.

CryptoStream CryptStream = new CryptoStream(NetStream,

RMCrypto.CreateDecryptor(Key, IV),

CryptoStreamMode.Read);

//Read the stream.

StreamReader SReader = new StreamReader(CryptStream);

//Display the message.

Console.WriteLine("The decrypted original message: {0}", SReader.ReadToEnd());

//Close the streams.

SReader.Close();

NetStream.Close();

TCP.Close();

}

//Catch any exceptions.

catch

{

Console.WriteLine("The Listener Failed.");

}

}

}

For the previous sample to work, an encrypted connection must be made to the listener. The connection must use the same key, IV, and algorithm used in the listener. If such a connection is made, the message is decrypted and displayed to the console.

**Asymmetric Decryption**

Someone who generates a public/private key pair typically performs asymmetric decryption. You can decrypt data using the[RSACryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsacryptoserviceprovider(v=vs.71).aspx) that you used to create the public/private key pair or initialize a new **RSACryptoServiceProvider** with the key information. Decryption will be successful only if you use the private key that corresponds to the public key used to encrypt the data.

The following example illustrates the decryption of two arrays of bytes that represent a symmetric key and IV.

VB

'Create a new instance of the RSACryptoServiceProvider class.

Dim RSA As New RSACryptoServiceProvider()

'Decrypt the symmetric key and IV.

SymmetricKey = RSA.Decrypt(EncryptedSymmetricKey, False)

SymmetricIV = RSA.Decrypt(EncryptedSymmetricIV, False)

[C#]

//Create a new instance of the RSACryptoServiceProvider class.

RSACryptoServiceProvider RSA = new RSACryptoServiceProvider();

//Decrypt the symmetric key and IV.

SymmetricKey = RSA.Decrypt( EncryptedSymmetricKey, false);

SymmetricIV = RSA.Decrypt( EncryptedSymmetricIV , false);

#### Cryptographic Signatures

**.NET Framework 1.1**

Cryptographic signatures allow you to verify that data originates from a certain individual. This section explains how to generate and verify digital signatures.

#### Cryptographic Signatures

**.NET Framework 2.0, 3.0, 3.5**

Cryptographic digital signatures use public key algorithms to provide data integrity. When you sign data with a digital signature, someone else can verify the signature, and can prove that the data originated from you and was not altered after you signed it. For more information about digital signatures, see [Cryptography Overview](https://msdn.microsoft.com/en-us/library/92f9ye3s(v=vs.80).aspx).

This section explains how to generate and verify digital signatures using classes in the [System.Security.Cryptography](https://msdn.microsoft.com/en-us/library/system.security.cryptography(v=vs.80).aspx) namespace.

See Also

**Concepts**

[Generating Signatures](https://msdn.microsoft.com/en-us/library/6yxzeb7e(v=vs.80).aspx)  
[Verifying Signatures](https://msdn.microsoft.com/en-us/library/201yh4c4(v=vs.80).aspx)

**Other Resources**

[Cryptographic Tasks](https://msdn.microsoft.com/en-us/library/7yx4d854(v=vs.80).aspx)  
[Cryptographic Services](https://msdn.microsoft.com/en-us/library/93bskf9z(v=vs.80).aspx)

#### Cryptographic Signatures

**.NET Framework 4, 4.5, 4.6**

Cryptographic digital signatures use public key algorithms to provide data integrity. When you sign data with a digital signature, someone else can verify the signature, and can prove that the data originated from you and was not altered after you signed it. For more information about digital signatures, see [Cryptographic Services](https://msdn.microsoft.com/en-us/library/92f9ye3s(v=vs.100).aspx).

This topic explains how to generate and verify digital signatures using classes in the [System.Security.Cryptography](https://msdn.microsoft.com/en-us/library/system.security.cryptography(v=vs.100).aspx) namespace.

* [Generating Signatures](https://msdn.microsoft.com/en-us/library/hk8wx38z(v=vs.100).aspx#generate)
* [Verifying Signatures](https://msdn.microsoft.com/en-us/library/hk8wx38z(v=vs.100).aspx#verify)

[**Generating Signatures**](javascript:void(0))

Digital signatures are usually applied to hash values that represent larger data. The following example applies a digital signature to a hash value. First, a new instance of the [RSACryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsacryptoserviceprovider(v=vs.100).aspx) class is created to generate a public/private key pair. Next, the[RSACryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsacryptoserviceprovider(v=vs.100).aspx) is passed to a new instance of the [RSAPKCS1SignatureFormatter](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsapkcs1signatureformatter(v=vs.100).aspx) class. This transfers the private key to the[RSAPKCS1SignatureFormatter](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsapkcs1signatureformatter(v=vs.100).aspx), which actually performs the digital signing. Before you can sign the hash code, you must specify a hash algorithm to use. This example uses the SHA1 algorithm. Finally, the [RSAPKCS1SignatureFormatter.CreateSignature](https://msdn.microsoft.com/en-us/library/system.security.cryptography.asymmetricsignatureformatter.createsignature(v=vs.100).aspx) method is called to perform the signing.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/hk8wx38z(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

class Class1

{

static void Main()

{

//The hash value to sign.

byte[] HashValue = {59,4,248,102,77,97,142,201,210,12,224,93,25,41,100,197,213,134,130,135};

//The value to hold the signed value.

byte[] SignedHashValue;

//Generate a public/private key pair.

RSACryptoServiceProvider RSA = new RSACryptoServiceProvider();

//Create an RSAPKCS1SignatureFormatter object and pass it the

//RSACryptoServiceProvider to transfer the private key.

RSAPKCS1SignatureFormatter RSAFormatter = new RSAPKCS1SignatureFormatter(RSA);

//Set the hash algorithm to SHA1.

RSAFormatter.SetHashAlgorithm("SHA1");

//Create a signature for HashValue and assign it to

//SignedHashValue.

SignedHashValue = RSAFormatter.CreateSignature(HashValue);

}

}

Signing XML Files

The .NET Framework provides the [System.Security.Cryptography.XML](https://msdn.microsoft.com/en-us/library/system.security.cryptography.xml.encrypteddata.encrypteddata(v=vs.100).aspx) namespace, which enables you sign XML. Signing XML is important when you want to verify that the XML originates from a certain source. For example, if you are using a stock quote service that uses XML, you can verify the source of the XML if it is signed.

The classes in this namespace follow the [XML-Signature Syntax and Processing recommendation](http://go.microsoft.com/fwlink/?LinkId=136777) from the World Wide Web Consortium.

[Back to top](https://msdn.microsoft.com/en-us/library/hk8wx38z(v=vs.100).aspx#top)

[**Verifying Signatures**](javascript:void(0))

To verify that data was signed by a particular party, you must have the following information:

* The public key of the party that signed the data.
* The digital signature.
* The data that was signed.
* The hash algorithm used by the signer.

To verify a signature signed by the [RSAPKCS1SignatureFormatter](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsapkcs1signatureformatter(v=vs.100).aspx) class, use the [RSAPKCS1SignatureDeformatter](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsapkcs1signaturedeformatter(v=vs.100).aspx) class. The[RSAPKCS1SignatureDeformatter](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsapkcs1signaturedeformatter(v=vs.100).aspx) class must be supplied the public key of the signer. You will need the values of the modulus and the exponent to specify the public key. (The party that generated the public/private key pair should provide these values.) First create an[RSACryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsacryptoserviceprovider(v=vs.100).aspx) object to hold the public key that will verify the signature, and then initialize an [RSAParameters](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsaparameters_fields(v=vs.100).aspx) structure to the modulus and exponent values that specify the public key.

The following code shows the creation of an [RSAParameters](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsaparameters_fields(v=vs.100).aspx) structure. The **Modulus** property is set to the value of a byte array calledModulusData and the **Exponent** property is set to the value of a byte array called ExponentData.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/hk8wx38z(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

RSAParameters RSAKeyInfo;

RSAKeyInfo.Modulus = ModulusData;

RSAKeyInfo.Exponent = ExponentData;

After you have created the [RSAParameters](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsaparameters_fields(v=vs.100).aspx) object, you can initialize a new instance of the [RSACryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsacryptoserviceprovider(v=vs.100).aspx) class to the values specified in [RSAParameters](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsaparameters_fields(v=vs.100).aspx). The [RSACryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsacryptoserviceprovider(v=vs.100).aspx) is, in turn, passed to the constructor of an [RSAPKCS1SignatureDeformatter](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsapkcs1signaturedeformatter(v=vs.100).aspx) to transfer the key.

The following example illustrates this process. In this example, HashValue and SignedHashValue are arrays of bytes provided by a remote party. The remote party has signed the HashValue using the SHA1 algorithm, producing the digital signature SignedHashValue. The

[RSAPKCS1SignatureDeformatter.VerifySignature](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsapkcs1signaturedeformatter.verifysignature(v=vs.100).aspx) method verifies that the digital signature is valid and was used to sign the HashValue.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/hk8wx38z(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-3)

RSACryptoServiceProvider RSA = new RSACryptoServiceProvider();

RSA.ImportParameters(RSAKeyInfo);

RSAPKCS1SignatureDeformatter RSADeformatter = new RSAPKCS1SignatureDeformatter(RSA);

RSADeformatter.SetHashAlgorithm("SHA1");

if(RSADeformatter.VerifySignature(HashValue, SignedHashValue))

{

Console.WriteLine("The signature is valid.");

}

else

{

Console.WriteLine("The signature is not valid.");

}

This code fragment will display "The signature is valid" if the signature is valid and "The signature is not valid" if it is not.

#### Cryptographic Signatures in Silverlight

**Silverlight**

In Silverlight, cryptographic digital signatures use a Hash-based Message Authentication Code (HMAC) such as the [HMACSHA256](https://msdn.microsoft.com/en-us/library/system.security.cryptography.hmacsha256(v=vs.95).aspx) algorithm to provide data integrity. When you sign data with a digital signature, someone else can verify the signature, and can prove that the data originated from you and was not altered after you signed it. For more information about digital signatures, see [Cryptographic Services](http://go.microsoft.com/fwlink/?LinkId=113746) in the .NET Framework documentation.

##### Generating Signatures

**.NET Framework 1.1, 2.0, 3.0, 3.5**

Digital signatures are usually applied to hash values that represent larger data. The following example applies a digital signature to a hash value. First, a new instance of the [RSACryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsacryptoserviceprovider(v=vs.71).aspx) class is created to generate a public/private key pair. Next, the**RSACryptoServiceProvider** is passed to a new instance of the [RSAPKCS1SignatureFormatter](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsapkcs1signatureformatter(v=vs.71).aspx) class. This transfers the private key to the**RSAPKCS1SignatureFormatter**, which actually performs the digital signing. Before you can sign the hash code, you must specify a hash algorithm to use. This example uses the SHA1 algorithm. Finally, the [RSAPKCS1SignatureFormatter.CreateSignature](https://msdn.microsoft.com/en-us/library/system.security.cryptography.asymmetricsignatureformatter.createsignature(v=vs.71).aspx) method is called to perform the signing.

VB

Imports System

Imports System.Security.Cryptography

Module Module1

Sub Main()

'The hash value to sign.

Dim HashValue As Byte() = {59, 4, 248, 102, 77, 97, 142, 201, 210, 12, 224, 93, 25, 41, 100, 197, 213, 134, 130, 135}

'The value to hold the signed value.

Dim SignedHashValue() As Byte

'Generate a public/private key pair.

Dim RSA As New RSACryptoServiceProvider()

'Create an RSAPKCS1SignatureFormatter object and pass it

'the RSACryptoServiceProvider to transfer the private key.

Dim RSAFormatter As New RSAPKCS1SignatureFormatter(RSA)

'Set the hash algorithm to SHA1.

RSAFormatter.SetHashAlgorithm("SHA1")

'Create a signature for HashValue and assign it to

'SignedHashValue.

SignedHashValue = RSAFormatter.CreateSignature(HashValue)

End Sub

End Module

using System;

using System.Security.Cryptography;

[C#]

class Class1

{

static void Main()

{

//The hash value to sign.

byte[] HashValue = {59,4,248,102,77,97,142,201,210,12,224,93,25,41,100,197,213,134,130,135};

//The value to hold the signed value.

byte[] SignedHashValue;

//Generate a public/private key pair.

RSACryptoServiceProvider RSA = new RSACryptoServiceProvider();

//Create an RSAPKCS1SignatureFormatter object and pass it the

//RSACryptoServiceProvider to transfer the private key.

RSAPKCS1SignatureFormatter RSAFormatter = new RSAPKCS1SignatureFormatter(RSA);

//Set the hash algorithm to SHA1.

RSAFormatter.SetHashAlgorithm("SHA1");

//Create a signature for HashValue and assign it to

//SignedHashValue.

SignedHashValue = RSAFormatter.CreateSignature(HashValue);

}

}

**Signing XML Files**

The .NET Framework provides the [System.Security.Cryptography.XML](https://msdn.microsoft.com/en-us/library/system.security.cryptography.xml.encrypteddata.encrypteddata(v=vs.71).aspx) namespace, which allows you sign XML. Signing XML is important when you want to verify that the XML originates from a certain source. For example, if you are using a stock quote service that uses XML, you can verify the source of the XML if it is signed.

The classes in this namespace follow the World Wide Web Consortium recommendation "XML-Signature Syntax and Processing," described at www.w3.org.

##### Verifying Signatures

**.NET Framework 1.1, 2.0, 3.0, 3.5**

In order to verify that data was signed by a particular party, you must have the following information:

* The public key of the party that signed the data.
* The digital signature.
* The data that was signed.
* The hash algorithm used by the signer.

To verify a signature signed by the [RSAPKCS1SignatureFormatter](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsapkcs1signatureformatter(v=vs.71).aspx)class, use the [RSAPKCS1SignatureDeformatter](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsapkcs1signaturedeformatter(v=vs.71).aspx)class. The**RSAPKCS1SignatureDeformatter** class must be supplied the public key of the signer. You will need the values of the modulus and the exponent to specify the public key. (The party that generated the public/private key pair should provide these values.) First create an[RSACryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsacryptoserviceprovider(v=vs.71).aspx) object to hold the public key that will verify the signature, and then initialize an [RSAParameters](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsaparameters_fields(v=vs.71).aspx) structure to the modulus and exponent values that specify the public key.

The following code shows the creation of an **RSAParameters** structure. The **Modulus** property is set to the value of a byte array calledModulusData and the **Exponent** property is set to the value of a byte array called ExponentData.

VB

Dim RSAKeyInfo As RSAParameters

RSAKeyInfo.Modulus = ModulusData

RSAKeyInfo.Exponent = ExponentData

[C#]

RSAParameters RSAKeyInfo;

RSAKeyInfo.Modulus = ModulusData;

RSAKeyInfo.Exponent = ExponentData;

After you have created the **RSAParameters**object, you can initialize a new instance of the **RSACryptoServiceProvider** class to the values specified in **RSAParameters**. The **RSACryptoServiceProvider** is, in turn, passed to the constructor of an **RSAPKCS1SignatureDeformatter**to transfer the key.

The following example illustrates this process. In this example, HashValue and SignedHashValue are arrays of bytes provided by a remote party. The remote party has signed the HashValue using the SHA1 algorithm, producing the digital signature SignedHashValue. The**RSAPKCS1SignatureDeformatter.VerifySignature**method verifies that the digital signature is valid and was used to sign the HashValue.

VB

Dim RSA As New RSACryptoServiceProvider()

RSA.ImportParameters(RSAKeyInfo)

Dim RSADeformatter As New RSAPKCS1SignatureDeformatter(RSA)

RSADeformatter.SetHashAlgorithm("SHA1")

If RSADeformatter.VerifySignature(HashValue, SignedHashValue) Then

Console.WriteLine("The signature is valid.")

Else

Console.WriteLine("The signture is not valid.")

End If

[C#]

RSACryptoServiceProvider RSA = new RSACryptoServiceProvider();

RSA2.ImportParameters(RSAKeyInfo);

RSAPKCS1SignatureDeformatter RSADeformatter = new RSAPKCS1SignatureDeformatter(RSA);

RSADeformatter.SetHashAlgorithm("SHA1");

if(RSADeformatter.VerifySignature(HashValue, SignedHashValue))

{

Console.WriteLine("The signature is valid.");

}

else

{

Console.WriteLine("The signature is not valid.");

}

This code fragment will display "The signature is valid" if the signature is valid and "The signature is not valid" if it is not.

#### Ensuring Data Integrity with Hash Codes

**.NET Framework 1.1, 2.0, 3.0**

A hash value is a numeric value of a fixed length that uniquely identifies data. Hash values are used with digital signatures because hash values represent a large amount of data as a much smaller numeric value. You can efficiently sign a hash value instead of signing the larger value. Hash values are also useful for verifying the integrity of data sent through insecure channels. The hash value of received data can be compared to the hash value of data as it was sent to determine whether the data was altered. This section describes how to generate hash codes using the classes in the [System.Security.Cryptography](https://msdn.microsoft.com/en-us/library/system.security.cryptography(v=vs.71).aspx) namespace.

3.5

A hash value is a numeric value of a fixed length that uniquely identifies data. Hash values are used with digital signatures because hash values represent a large amount of data as a much smaller numeric value. You can efficiently sign a hash value instead of signing the larger value. Hash values are also useful for verifying the integrity of data sent through insecure channels. The hash value of received data can be compared to the hash value of data as it was sent to determine whether the data was altered.

The topics in this section ([Generating a Hash](https://msdn.microsoft.com/en-us/library/w1t5hx6k(v=vs.90).aspx) and [Verifying a Hash](https://msdn.microsoft.com/en-us/library/yeyw8w2d(v=vs.90).aspx)) describe how to generate hash codes using the classes in the[System.Security.Cryptography](https://msdn.microsoft.com/en-us/library/system.security.cryptography(v=vs.90).aspx) namespace.

##### Generating a Hash

**.NET Framework 1.1, 2.0, 3.0, 3.5**

The managed hash classes can hash either an array of bytes or a managed stream object. The following example uses the SHA1 hash algorithm to create a hash value for a string. The example uses the [UnicodeEncoding](https://msdn.microsoft.com/en-us/library/system.text.unicodeencoding(v=vs.85).aspx) class to convert the string into an array of bytes that are hashed using the [SHA1Managed](https://msdn.microsoft.com/en-us/library/system.security.cryptography.sha1managed(v=vs.85).aspx) class. The hash value is then displayed to the console.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/w1t5hx6k(v=vs.85).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

using System;

using System.IO;

using System.Security.Cryptography;

using System.Text;

class Class1

{

static void Main(string[] args)

{

byte[] HashValue;

string MessageString = "This is the original message!";

//Create a new instance of the UnicodeEncoding class to

//convert the string into an array of Unicode bytes.

UnicodeEncoding UE = new UnicodeEncoding();

//Convert the string into an array of bytes.

byte[] MessageBytes = UE.GetBytes(MessageString);

//Create a new instance of the SHA1Managed class to create

//the hash value.

SHA1Managed SHhash = new SHA1Managed();

//Create the hash value from the array of bytes.

HashValue = SHhash.ComputeHash(MessageBytes);

//Display the hash value to the console.

foreach(byte b in HashValue)

{

Console.Write("{0} ", b);

}

}

}

This code will display the following string to the console:

59 4 248 102 77 97 142 201 210 12 224 93 25 41 100 197 213 134 130 135

##### Verifying a Hash

**.NET Framework 1.1, 2.0, 3.0, 3.5**

Data can be compared to a hash value to determine its integrity. Usually, data is hashed at a certain time and the hash value is protected in some way. At a later time, the data can be hashed again and compared to the protected value. If the hash values match, then the data has not been altered. However, if the values do not match, the data has been corrupted. For this system to work, the protected hash must be encrypted or kept secret from all untrusted parties.

The following example compares the previous hash value of a string to a new hash value. This example loops through each byte of the hash values and makes a comparison.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/yeyw8w2d(v=vs.90).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

using System;

using System.Security.Cryptography;

using System.Text;

class Class1

{

static void Main()

{

//This hash value is produced from "This is the original message!"

//using SHA1Managed.

byte[] SentHashValue = {59,4,248,102,77,97,142,201,210,12,224,93,25,41,100,197,213,134,130,135};

//This is the string that corresponds to the previous hash value.

string MessageString = "This is the original message!";

byte[] CompareHashValue;

//Create a new instance of the UnicodeEncoding class to

//convert the string into an array of Unicode bytes.

UnicodeEncoding UE = new UnicodeEncoding();

//Convert the string into an array of bytes.

byte[] MessageBytes = UE.GetBytes(MessageString);

//Create a new instance of the SHA1Managed class to create

//the hash value.

SHA1Managed SHhash = new SHA1Managed();

//Create the hash value from the array of bytes.

CompareHashValue = SHhash.ComputeHash(MessageBytes);

bool Same = true;

//Compare the values of the two byte arrays.

for(int x = 0; x < SentHashValue.Length;x++)

{

if (SentHashValue[x] != CompareHashValue[x])

{

Same = false;

}

}

//Display whether or not the hash values are the same.

if(Same)

{

Console.WriteLine("The hash codes match.");

}

else

{

Console.WriteLine("The hash codes do not match.");

}

}

}

If the two hash values match, this code displays the following to the console:

The hash codes match.

However, if they do not match, the code displays the following:

The hash codes do not match.

#### Ensuring Data Integrity with Hash Codes

**.NET Framework 4, 4.5, 4.6**

A hash value is a numeric value of a fixed length that uniquely identifies data. Hash values represent large amounts of data as much smaller numeric values, so they are used with digital signatures. You can sign a hash value more efficiently than signing the larger value. Hash values are also useful for verifying the integrity of data sent through insecure channels. The hash value of received data can be compared to the hash value of data as it was sent to determine whether the data was altered.

This topic describes how to generate and verify hash codes by using the classes in the [System.Security.Cryptography](https://msdn.microsoft.com/en-us/library/system.security.cryptography(v=vs.100).aspx) namespace.

[**Generating a Hash**](javascript:void(0))

The managed hash classes can hash either an array of bytes or a managed stream object. The following example uses the SHA1 hash algorithm to create a hash value for a string. The example uses the [UnicodeEncoding](https://msdn.microsoft.com/en-us/library/system.text.unicodeencoding(v=vs.100).aspx) class to convert the string into an array of bytes that are hashed by using the [SHA1Managed](https://msdn.microsoft.com/en-us/library/system.security.cryptography.sha1managed(v=vs.100).aspx) class. The hash value is then displayed to the console.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/f9ax34y5(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

using System;

using System.IO;

using System.Security.Cryptography;

using System.Text;

class Class1

{

static void Main(string[] args)

{

byte[] HashValue;

string MessageString = "This is the original message!";

//Create a new instance of the UnicodeEncoding class to

//convert the string into an array of Unicode bytes.

UnicodeEncoding UE = new UnicodeEncoding();

//Convert the string into an array of bytes.

byte[] MessageBytes = UE.GetBytes(MessageString);

//Create a new instance of the SHA1Managed class to create

//the hash value.

SHA1Managed SHhash = new SHA1Managed();

//Create the hash value from the array of bytes.

HashValue = SHhash.ComputeHash(MessageBytes);

//Display the hash value to the console.

foreach (byte b in HashValue)

{

Console.Write("{0} ", b);

}

}

}

This code will display the following string to the console:

59 4 248 102 77 97 142 201 210 12 224 93 25 41 100 197 213 134 130 135

[**Verifying a Hash**](javascript:void(0))

Data can be compared to a hash value to determine its integrity. Usually, data is hashed at a certain time and the hash value is protected in some way. At a later time, the data can be hashed again and compared to the protected value. If the hash values match, the data has not been altered. If the values do not match, the data has been corrupted. For this system to work, the protected hash must be encrypted or kept secret from all untrusted parties.

The following example compares the previous hash value of a string to a new hash value. This example loops through each byte of the hash values and makes a comparison.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/f9ax34y5(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

using System;

using System.Security.Cryptography;

using System.Text;

class Class1

{

static void Main()

{

//This hash value is produced from "This is the original message!"

//using SHA1Managed.

byte[] SentHashValue = { 59, 4, 248, 102, 77, 97, 142, 201, 210, 12, 224, 93, 25, 41, 100, 197, 213, 134, 130, 135 };

//This is the string that corresponds to the previous hash value.

string MessageString = "This is the original message!";

byte[] CompareHashValue;

//Create a new instance of the UnicodeEncoding class to

//convert the string into an array of Unicode bytes.

UnicodeEncoding UE = new UnicodeEncoding();

//Convert the string into an array of bytes.

byte[] MessageBytes = UE.GetBytes(MessageString);

//Create a new instance of the SHA1Managed class to create

//the hash value.

SHA1Managed SHhash = new SHA1Managed();

//Create the hash value from the array of bytes.

CompareHashValue = SHhash.ComputeHash(MessageBytes);

bool Same = true;

//Compare the values of the two byte arrays.

for (int x = 0; x < SentHashValue.Length; x++)

{

if (SentHashValue[x] != CompareHashValue[x])

{

Same = false;

}

}

//Display whether or not the hash values are the same.

if (Same)

{

Console.WriteLine("The hash codes match.");

}

else

{

Console.WriteLine("The hash codes do not match.");

}

}

}

If the two hash values match, this code displays the following to the console:

The hash codes match.

If they do not match, the code displays the following:

The hash codes do not match.

#### Ensuring Data Integrity with Hash Codes

**Silverlight**

In Silverlight-based applications, you can use hash codes to help ensure data integrity.

A hash value is a numeric value of a fixed length that uniquely identifies data. Hash values are created with a hashing algorithm derived from the [System.Security.Cryptography.HashAlgorithm](https://msdn.microsoft.com/en-us/library/system.security.cryptography.hashalgorithm(v=vs.95).aspx) class. A hashing algorithm is a well-defined mathematical procedure that represents a (potentially) large amount of data as a much smaller integer.

Hash values are also useful for verifying the integrity of data sent through insecure channels. The hash value of received data can be compared to the hash value of data when it was sent to determine whether the data was altered.

Digital signatures often consist of encrypted hash values. Encrypting hash values is an efficient use of computational and bandwidth resources, because hash values represent a large amount of data as a much smaller numeric value. Depending upon your security requirements, you may choose to encrypt a message's hash value instead of encrypting the message itself.

This topic describes how to generate and verify hash codes in Silverlight using the classes in the [System.Security.Cryptography](https://msdn.microsoft.com/en-us/library/system.security.cryptography(v=vs.95).aspx) namespace. It contains the following sections:

* [Generating a Hash Code](https://msdn.microsoft.com/en-us/library/f9ax34y5(v=vs.95).aspx#generate)
* [Verifying a Hash Code](https://msdn.microsoft.com/en-us/library/f9ax34y5(v=vs.95).aspx#verify)

[**Generating a Hash Code**](javascript:void(0))

The managed hash classes can hash either an array of bytes or a managed stream object. The two examples in this section demonstrate these scenarios.

The following example shows how to use the HMACSHA256 constructor and an [IsolatedStorageFileStream](https://msdn.microsoft.com/en-us/library/system.io.isolatedstorage.isolatedstoragefilestream(v=vs.95).aspx) object to encrypt an isolated storage file. The key that is used to calculate the hash code is derived using a password that is specified by the user. This code example is part of a larger example provided for the [HMACSHA256](https://msdn.microsoft.com/en-us/library/system.security.cryptography.hmacsha256(v=vs.95).aspx) class.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/f9ax34y5(v=vs.95).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

// Initialize the keyed hash object.

HMACSHA256 myhmacsha256 = new HMACSHA256(key);

IsolatedStorageFileStream inStream = isoStore.OpenFile(sourceFilePath, FileMode.Open);

inStream.Position = 0;

// Compute the hash of the input file.

byte[] hashValue = myhmacsha256.ComputeHash(inStream);

// Reset inStream to the beginning of the file.

inStream.Position = 0;

// Write the computed hash value to the output file.

outStream.Write(hashValue, 0, hashValue.Length);

// Copy the contents of the sourceFile to the destFile.

int bytesRead;

// read 1K at a time

byte[] buffer = new byte[1024];

do

{

// Read from the wrapping CryptoStream.

bytesRead = inStream.Read(buffer, 0, 1024);

outStream.Write(buffer, 0, bytesRead);

} while (bytesRead > 0);

myhmacsha256.Clear();

The following example shows how to use the [System.Text.UnicodeEncoding](https://msdn.microsoft.com/en-us/library/system.text.unicodeencoding(v=vs.95).aspx) class to convert a string into an array of bytes that can be hashed using one of the hashing algorithms.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/f9ax34y5(v=vs.95).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

byte[] HashValue;

string MessageString = "This is the original message!";

// Create a new instance of the UnicodeEncoding class to

// convert the string into an array of Unicode bytes.

UnicodeEncoding UE = new UnicodeEncoding();

// Convert the string into an array of bytes.

byte[] MessageBytes = UE.GetBytes(MessageString);

// Create a new instance of the SHA1Managed class to create

// the hash value.

SHA1Managed SHhash = new SHA1Managed();

// Create the hash value from the array of bytes.

HashValue = SHhash.ComputeHash(MessageBytes);

[**Verifying a Hash Code**](javascript:void(0))

Data can be compared to a hash value to determine its integrity. Usually, data is hashed at a certain time and the hash value is protected in some way. At a later time, the data can be hashed again and compared to the protected value. If the hash values match, the data has not been altered. If the values do not match, the data has been corrupted.

The following example shows how to use the [ComputeHash](https://msdn.microsoft.com/en-us/library/system.security.cryptography.hashalgorithm.computehash(v=vs.95).aspx) method to validate the signature of an isolated storage file. The key that is used to calculate the hash code is derived from a password that is specified by the user. This code example is part of a larger example provided for the [HMACSHA256](https://msdn.microsoft.com/en-us/library/system.security.cryptography.hmacsha256(v=vs.95).aspx) class.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/f9ax34y5(v=vs.95).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-3)

// Initialize the keyed hash object.

HMACSHA256 hmacsha256 = new HMACSHA256(key);

// Create an array to hold the keyed hash value read from the file.

byte[] storedHash = new byte[hmacsha256.HashSize / 8];

// Create a FileStream for the source file

// Read in the storedHash.

inStream.Read(storedHash, 0, storedHash.Length);

// Compute the hash of the remaining contents of the file.

// The stream is properly positioned at the beginning of the content,

// immediately after the stored hash value.

byte[] computedHash = hmacsha256.ComputeHash(inStream);

// compare the computed hash with the stored value

for (int i = 0; i < storedHash.Length; i++)

{

if (computedHash[i] != storedHash[i])

{

outputBlock.Text = "Hash values differ! Either wrong password or file has changed.";

return false;

}

}

outputBlock.Text = "Hash values agree -- no tampering occurred.";

#### Creating a Cryptographic Scheme

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4, 4.5, 4.6**

The cryptographic components of the .NET Framework can be combined to create different schemes to encrypt and decrypt data.

A simple cryptographic scheme for encrypting and decrypting data might specify the following steps:

1. Each party generates a public/private key pair.
2. The parties exchange their public keys.
3. Each party generates a secret key for TripleDES encryption, for example, and encrypts the newly created key using the other's public key.
4. Each party sends the data to the other and combines the other's secret key with its own, in a particular order, to create a new secret key.
5. The parties then initiate a conversation using symmetric encryption.

Creating a cryptographic scheme is not a trivial task. For more information on using cryptography, see the Cryptography topic in the Platform SDK documentation at http://msdn.microsoft.com/library.

#### Extending the KeyedHashAlgorithm Class

**.NET Framework 2.0, 3.0, 3.5, 4**

The cryptographic classes provided by the .NET Framework are extremely extensible. This section provides an example of how to extend these cryptography classes by creating a keyed hash algorithm class that implements the [MD5](https://msdn.microsoft.com/en-US/library/system.security.cryptography.md5(v=vs.80).aspx) hash algorithm.

A keyed hash algorithm is a key-dependent, one-way hash function used as a message authentication code. The .NET Framework provides two such keyed hash algorithm classes ([HMACSHA1](https://msdn.microsoft.com/en-US/library/system.security.cryptography.hmacsha1(v=vs.80).aspx) and [MACTripleDES](https://msdn.microsoft.com/en-US/library/system.security.cryptography.mactripledes(v=vs.80).aspx)), both of which derive from the abstract [KeyedHashAlgorithm Class](https://msdn.microsoft.com/en-US/library/system.security.cryptography.keyedhashalgorithm(v=vs.80).aspx). If you want to implement a keyed hash algorithm class that implements a hash algorithm different from the ones provided in the .NET Framework, you can create a new class derived from **KeyedHashAlgorithm** that uses your preferred hash algorithm.

The following example creates a new keyed hash algorithm class using the **MD5** hash algorithm. Because this class inherits from the**KeyedHashAlgorithm** class, it can easily be used with other managed cryptography classes like the [CryptoStream](https://msdn.microsoft.com/en-US/library/system.security.cryptography.cryptostream(v=vs.80).aspx) class.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/33kc9tdw(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

using System;

using System.Security.Cryptography;

public class TestHMACMD5

{

static private void PrintByteArray(Byte[] arr)

{

int i;

Console.WriteLine("Length: " + arr.Length);

for (i=0; i<arr.Length; i++)

{

Console.Write("{0:X}", arr[i]);

Console.Write(" ");

if ( (i+9)%8 == 0 ) Console.WriteLine();

}

if (i%8 != 0) Console.WriteLine();

}

public static void Main()

{

// Create a key.

byte[] key1 = {0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b};

// Pass the key to the constructor of the HMACMD5 class.

HMACMD5 hmac1 = new HMACMD5(key1);

// Create another key.

byte[] key2 = System.Text.Encoding.ASCII.GetBytes("KeyString");

// Pass the key to the constructor of the HMACMD5 class.

HMACMD5 hmac2 = new HMACMD5(key2);

// Encode a string into a byte array, create a hash of the array,

// and print the hash to the screen.

byte[] data1 = System.Text.Encoding.ASCII.GetBytes("Hi There");

PrintByteArray(hmac1.ComputeHash(data1));

// Encode a string into a byte array, create a hash of the array,

// and print the hash to the screen.

byte[] data2 = System.Text.Encoding.ASCII.GetBytes("This data will be hashed.");

PrintByteArray(hmac2.ComputeHash(data2));

}

}

public class HMACMD5 : KeyedHashAlgorithm

{

private MD5 hash1;

private MD5 hash2;

private bool bHashing = false;

private byte[] rgbInner = new byte[64];

private byte[] rgbOuter = new byte[64];

public HMACMD5 (byte[] rgbKey)

{

HashSizeValue = 128;

// Create the hash algorithms.

hash1 = MD5.Create();

hash2 = MD5.Create();

// Get the key.

if (rgbKey.Length > 64)

{

KeyValue = hash1.ComputeHash(rgbKey);

// No need to call Initialize; ComputeHash does it automatically.

}

else

{

KeyValue = (byte[]) rgbKey.Clone();

}

// Compute rgbInner and rgbOuter.

int i = 0;

for (i=0; i<64; i++)

{

rgbInner[i] = 0x36;

rgbOuter[i] = 0x5C;

}

for (i=0; i<KeyValue.Length; i++)

{

rgbInner[i] ^= KeyValue[i];

rgbOuter[i] ^= KeyValue[i];

}

}

public override byte[] Key

{

get { return (byte[]) KeyValue.Clone(); }

set

{

if (bHashing)

{

throw new Exception("Cannot change key during hash operation");

}

if (value.Length > 64)

{

KeyValue = hash1.ComputeHash(value);

// No need to call Initialize; ComputeHash does it automatically.

}

else

{

KeyValue = (byte[]) value.Clone();

}

// Compute rgbInner and rgbOuter.

int i = 0;

for (i=0; i<64; i++)

{

rgbInner[i] = 0x36;

rgbOuter[i] = 0x5C;

}

for (i=0; i<KeyValue.Length; i++)

{

rgbInner[i] ^= KeyValue[i];

rgbOuter[i] ^= KeyValue[i];

}

}

}

public override void Initialize()

{

hash1.Initialize();

hash2.Initialize();

bHashing = false;

}

protected override void HashCore(byte[] rgb, int ib, int cb)

{

if (bHashing == false)

{

hash1.TransformBlock(rgbInner, 0, 64, rgbInner, 0);

bHashing = true;

}

hash1.TransformBlock(rgb, ib, cb, rgb, ib);

}

protected override byte[] HashFinal()

{

if (bHashing == false)

{

hash1.TransformBlock(rgbInner, 0, 64, rgbInner, 0);

bHashing = true;

}

// Finalize the original hash.

hash1.TransformFinalBlock(new byte[0], 0, 0);

// Write the outer array.

hash2.TransformBlock(rgbOuter, 0, 64, rgbOuter, 0);

// Write the inner hash and finalize the hash.

hash2.TransformFinalBlock(hash1.Hash, 0, hash1.Hash.Length);

bHashing = false;

return hash2.Hash;

}

}

This example extends the **KeyedHashAlgorithm** class to use the **MD5** hash algorithm in a class called HMACMD5. When executed, this code creates two instances of the HMACMD5 class, passing different keys to each constructor. It then hashes the values of two strings and prints the following to the console.

Length: 16

92 94 72 7A 36 38 BB 1C

13 F4 8E F8 15 8B FC 9D

Length: 16

1 10 3A 9 C2 10 6E E1

1 13 9B 7D DA EA A3 EC

**See Also**

#### XML Encryption and Digital Signatures

**.NET Framework 2.0, 3.0, 3.5**

The .NET Framework provides several classes that allow you to encrypt and decrypt XML data, and create and verify XML digital signatures. These classes provide a way to maintain the confidentiality and integrity of your XML data.

**In This Section**

[How to: Encrypt XML Elements with Symmetric Keys](https://msdn.microsoft.com/en-US/library/sb7w85t6(v=vs.80).aspx)

Describes how to encrypt an XML element using the [Rijndael](https://msdn.microsoft.com/en-US/library/system.security.cryptography.rijndael(v=vs.80).aspx) algorithm.

[How to: Decrypt XML Elements with Symmetric Keys](https://msdn.microsoft.com/en-US/library/ms229740(v=vs.80).aspx)

Describes how to decrypt an XML element that was encrypted using the **Rijndael** algorithm.

[How to: Encrypt XML Elements with Asymmetric Keys](https://msdn.microsoft.com/en-US/library/ms229746(v=vs.80).aspx)

Describes how to encrypt an XML element using the [RSA](https://msdn.microsoft.com/en-US/library/system.security.cryptography.rsa(v=vs.80).aspx) algorithm.

[How to: Decrypt XML Elements with Asymmetric Keys](https://msdn.microsoft.com/en-US/library/ms229919(v=vs.80).aspx)

Describes how to decrypt an XML element using the **RSA** algorithm.

[How to: Encrypt XML Elements with X.509 Certificates](https://msdn.microsoft.com/en-US/library/ms229744(v=vs.80).aspx)

Describes how to encrypt an XML element using an X.509 certificate from a certificate store.

[How to: Decrypt XML Elements with X.509 Certificates](https://msdn.microsoft.com/en-US/library/ms229943(v=vs.80).aspx)

Describes how to decrypt an XML element using an X.509 certificate from a certificate store.

[How to: Sign XML Documents with Digital Signatures](https://msdn.microsoft.com/en-US/library/ms229745(v=vs.80).aspx)

Describes how to sign an XML document using the **RSA** algorithm.

[How to: Verify the Digital Signatures of XML Documents](https://msdn.microsoft.com/en-US/library/ms229950(v=vs.80).aspx)

Describes how to verify an XML document using the **RSA** algorithm.

**Reference**

[System.Security.Cryptography.Xml](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml(v=vs.80).aspx)

Provides links to reference documentation for classes used for XML encryption and digital signatures.

[System.Security.Cryptography.Xml.SignedXml](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml.signedxml(v=vs.80).aspx)

Describes the features of the **SignedXml** class, which is used to sign and verify XML data.

[System.Security.Cryptography.Xml.EncryptedXml](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml.encryptedxml(v=vs.80).aspx)

Describes the features of the **EncryptedXml** class, which is used to encrypt and decrypt XML data.

##### How to: Encrypt XML Elements with Symmetric Keys

**.NET Framework 2.0, 3.0, 3.5, 4, 4.5, 4.6**

You can use the classes in the [System.Security.Cryptography.Xml](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml(v=vs.80).aspx) namespace to encrypt an element within an XML document. XML Encryption allows you to store or transport sensitive XML, without worrying about the data being easily read. This procedure decrypts an XML element using the Advanced Encryption Standard (AES) algorithm, also known as Rijndael.

For information about how to decrypt an XML element that was encrypted using this procedure, see [How to: Decrypt XML Elements with Symmetric Keys](https://msdn.microsoft.com/en-US/library/ms229740(v=vs.80).aspx).

When you use a symmetric algorithm like AES to encrypt XML data, you must use the same key to encrypt and decrypt the XML data. The example in this procedure assumes that the encrypted XML will be decrypted using the same key, and that the encrypting and decrypting parties agree on the algorithm and key to use. This example does not store or encrypt the AES key within the encrypted XML.

This example is appropriate for situations where a single application needs to encrypt data based on a session key stored in memory, or based on a cryptographically strong key derived from a password. For situations where two or more applications need to share encrypted XML data, consider using an encryption scheme based on an asymmetric algorithm or an X.509 certificate.

**To encrypt an XML element with a symmetric key**

1. Generate a symmetric key using the [RijndaelManaged](https://msdn.microsoft.com/en-US/library/system.security.cryptography.rijndaelmanaged(v=vs.80).aspx) class. This key will be used to encrypt the XML element.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/sb7w85t6(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

RijndaelManaged key = null;

try

{

// Create a new Rijndael key.

key = new RijndaelManaged();

1. Create an [XmlDocument](https://msdn.microsoft.com/en-US/library/system.xml.xmldocument(v=vs.80).aspx) object by loading an XML file from disk. The **XmlDocument** object contains the XML element to encrypt.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/sb7w85t6(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

// Load an XML document.

XmlDocument xmlDoc = new XmlDocument();

xmlDoc.PreserveWhitespace = true;

xmlDoc.Load("test.xml");

1. Find the specified element in the **XmlDocument** object and create a new [XmlElement](https://msdn.microsoft.com/en-US/library/system.xml.xmlelement(v=vs.80).aspx) object to represent the element you want to encrypt. In this example, the "creditcard" element is encrypted.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/sb7w85t6(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-3)

XmlElement elementToEncrypt = Doc.GetElementsByTagName(ElementName)[0] as XmlElement;

1. Create a new instance of the [EncryptedXml](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml.encryptedxml(v=vs.80).aspx) class and use it to encrypt the **XmlElement** with the symmetric key. The [EncryptData](https://msdn.microsoft.com/en-US/library/y259436w(v=vs.80).aspx)method returns the encrypted element as an array of encrypted bytes.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/sb7w85t6(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-4)

EncryptedXml eXml = new EncryptedXml();

byte[] encryptedElement = eXml.EncryptData(elementToEncrypt, Key, false);

1. Construct an [EncryptedData](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml.encrypteddata(v=vs.80).aspx) object and populate it with the URL identifier of the XML Encryption element. This URL identifier lets a decrypting party know that the XML contains an encrypted element. You can use the [XmlEncElementUrl](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml.encryptedxml.xmlencelementurl(v=vs.80).aspx) field to specify the URL identifier.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/sb7w85t6(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-5)

EncryptedData edElement = new EncryptedData();

edElement.Type = EncryptedXml.XmlEncElementUrl;

1. Create an [EncryptionMethod](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml.encryptionmethod(v=vs.80).aspx) object that is initialized to the URL identifier of the cryptographic algorithm used to generate the key. Pass the **EncryptionMethod** object to the [EncryptionMethod](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml.encryptedtype.encryptionmethod(v=vs.80).aspx) property.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/sb7w85t6(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-6)

string encryptionMethod = null;

if (Key is TripleDES)

{

encryptionMethod = EncryptedXml.XmlEncTripleDESUrl;

}

else if (Key is DES)

{

encryptionMethod = EncryptedXml.XmlEncDESUrl;

}

if (Key is Rijndael)

{

switch (Key.KeySize)

{

case 128:

encryptionMethod = EncryptedXml.XmlEncAES128Url;

break;

case 192:

encryptionMethod = EncryptedXml.XmlEncAES192Url;

break;

case 256:

encryptionMethod = EncryptedXml.XmlEncAES256Url;

break;

}

}

else

{

// Throw an exception if the transform is not in the previous categories

throw new CryptographicException("The specified algorithm is not supported for XML Encryption.");

}

edElement.EncryptionMethod = new EncryptionMethod(encryptionMethod);

1. Add the encrypted element data to the **EncryptedData** object.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/sb7w85t6(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-7)

edElement.CipherData.CipherValue = encryptedElement;

1. Replace the element from the original **XmlDocument** object with the **EncryptedData** element.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/sb7w85t6(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-8)

EncryptedXml.ReplaceElement(elementToEncrypt, edElement, false);

**Example**

C#

[**VB**](https://msdn.microsoft.com/en-US/library/sb7w85t6(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-9)

using System;

using System.Xml;

using System.Security.Cryptography;

using System.Security.Cryptography.Xml;

namespace CSCrypto

{

class Program

{

static void Main(string[] args)

{

RijndaelManaged key = null;

try

{

// Create a new Rijndael key.

key = new RijndaelManaged();

// Load an XML document.

XmlDocument xmlDoc = new XmlDocument();

xmlDoc.PreserveWhitespace = true;

xmlDoc.Load("test.xml");

// Encrypt the "creditcard" element.

Encrypt(xmlDoc, "creditcard", key);

Console.WriteLine("The element was encrypted");

Console.WriteLine(xmlDoc.InnerXml);

Decrypt(xmlDoc, key);

Console.WriteLine("The element was decrypted");

Console.WriteLine(xmlDoc.InnerXml);

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

finally

{

// Clear the key.

if (key != null)

{

key.Clear();

}

}

}

public static void Encrypt(XmlDocument Doc, string ElementName, SymmetricAlgorithm Key)

{

// Check the arguments.

if (Doc == null)

throw new ArgumentNullException("Doc");

if (ElementName == null)

throw new ArgumentNullException("ElementToEncrypt");

if (Key == null)

throw new ArgumentNullException("Alg");

////////////////////////////////////////////////

// Find the specified element in the XmlDocument

// object and create a new XmlElemnt object.

////////////////////////////////////////////////

XmlElement elementToEncrypt = Doc.GetElementsByTagName(ElementName)[0] as XmlElement;

// Throw an XmlException if the element was not found.

if (elementToEncrypt == null)

{

throw new XmlException("The specified element was not found");

}

//////////////////////////////////////////////////

// Create a new instance of the EncryptedXml class

// and use it to encrypt the XmlElement with the

// symmetric key.

//////////////////////////////////////////////////

EncryptedXml eXml = new EncryptedXml();

byte[] encryptedElement = eXml.EncryptData(elementToEncrypt, Key, false);

////////////////////////////////////////////////

// Construct an EncryptedData object and populate

// it with the desired encryption information.

////////////////////////////////////////////////

EncryptedData edElement = new EncryptedData();

edElement.Type = EncryptedXml.XmlEncElementUrl;

// Create an EncryptionMethod element so that the

// receiver knows which algorithm to use for decryption.

// Determine what kind of algorithm is being used and

// supply the appropriate URL to the EncryptionMethod element.

string encryptionMethod = null;

if (Key is TripleDES)

{

encryptionMethod = EncryptedXml.XmlEncTripleDESUrl;

}

else if (Key is DES)

{

encryptionMethod = EncryptedXml.XmlEncDESUrl;

}

if (Key is Rijndael)

{

switch (Key.KeySize)

{

case 128:

encryptionMethod = EncryptedXml.XmlEncAES128Url;

break;

case 192:

encryptionMethod = EncryptedXml.XmlEncAES192Url;

break;

case 256:

encryptionMethod = EncryptedXml.XmlEncAES256Url;

break;

}

}

else

{

// Throw an exception if the transform is not in the previous categories

throw new CryptographicException("The specified algorithm is not supported for XML Encryption.");

}

edElement.EncryptionMethod = new EncryptionMethod(encryptionMethod);

// Add the encrypted element data to the

// EncryptedData object.

edElement.CipherData.CipherValue = encryptedElement;

////////////////////////////////////////////////////

// Replace the element from the original XmlDocument

// object with the EncryptedData element.

////////////////////////////////////////////////////

EncryptedXml.ReplaceElement(elementToEncrypt, edElement, false);

}

public static void Decrypt(XmlDocument Doc, SymmetricAlgorithm Alg)

{

// Check the arguments.

if (Doc == null)

throw new ArgumentNullException("Doc");

if (Alg == null)

throw new ArgumentNullException("Alg");

// Find the EncryptedData element in the XmlDocument.

XmlElement encryptedElement = Doc.GetElementsByTagName("EncryptedData")[0] as XmlElement;

// If the EncryptedData element was not found, throw an exception.

if (encryptedElement == null)

{

throw new XmlException("The EncryptedData element was not found.");

}

// Create an EncryptedData object and populate it.

EncryptedData edElement = new EncryptedData();

edElement.LoadXml(encryptedElement);

// Create a new EncryptedXml object.

EncryptedXml exml = new EncryptedXml();

// Decrypt the element using the symmetric key.

byte[] rgbOutput = exml.DecryptData(edElement, Alg);

// Replace the encryptedData element with the plaintext XML element.

exml.ReplaceData(encryptedElement, rgbOutput);

}

}

}

This example assumes that a file named "test.xml" exists in the same directory as the compiled program. It also assumes that "test.xml"contains a "creditcard" element. You can place the following XML into a file called test.xml and use it with this example.

<root>

<creditcard>

<number>19834209</number>

<expiry>02/02/2002</expiry>

</creditcard>

</root>

**Compiling the Code**

* To compile this example, you need to include a reference to **System.Security.dll**.
* Include the following namespaces: [System.Xml](https://msdn.microsoft.com/en-US/library/system.xml(v=vs.80).aspx), [System.Security.Cryptography](https://msdn.microsoft.com/en-US/library/system.security.cryptography(v=vs.80).aspx), and **System.Security.Cryptography.Xml**.

**Security**

Never store a cryptographic key in plaintext or transfer a key between machines in plaintext. Instead, use a secure key container to store cryptographic keys.

When you are done using a cryptographic key, clear it from memory by setting each byte to zero or by calling the [Clear](https://msdn.microsoft.com/en-US/library/system.security.cryptography.symmetricalgorithm.clear(v=vs.80).aspx) method of the managed cryptography class.

##### How to: Decrypt XML Elements with Symmetric Keys

**.NET Framework 2.0 - - -**

You can use the classes in the [System.Security.Cryptography.Xml](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml(v=vs.80).aspx) namespace to encrypt an element within an XML document. XML Encryption allows you to store or transport sensitive XML, without worrying about the data being easily read. This code example decrypts an XML element using the Advanced Encryption Standard (AES) algorithm, also known as Rijndael.

For information about how to encrypt an XML element using this procedure, see [How to: Encrypt XML Elements with Symmetric Keys](https://msdn.microsoft.com/en-US/library/sb7w85t6(v=vs.80).aspx).

When you use a symmetric algorithm like AES to encrypt XML data, you must use the same key to encrypt and decrypt the XML data. The example in this procedure assumes that the encrypted XML was encrypted using the same key, and that the encrypting and decrypting parties agree on the algorithm and key to use. This example does not store or encrypt the AES key within the encrypted XML.

This example is appropriate for situations where a single application needs to encrypt data based on a session key stored in memory, or based on a cryptographically strong key derived from a password. For situations where two or more applications need to share encrypted XML data, consider using an encryption scheme based on an asymmetric algorithm or an X.509 certificate.

**To decrypt an XML element with a symmetric key**

1. Encrypt an XML element with the previously generated key using the techniques described in [How to: Encrypt XML Elements with Symmetric Keys](https://msdn.microsoft.com/en-US/library/sb7w85t6(v=vs.80).aspx).
2. Find the <**EncryptedData**> element (defined by the XML Encryption standard) in an [XmlDocument](https://msdn.microsoft.com/en-US/library/system.xml.xmldocument(v=vs.80).aspx) object that contains the encrypted XML and create a new [XmlElement](https://msdn.microsoft.com/en-US/library/system.xml.xmlelement(v=vs.80).aspx) object to represent that element.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229740(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

XmlElement encryptedElement = Doc.GetElementsByTagName("EncryptedData")[0] as XmlElement;

1. Create an [EncryptedData](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml.encrypteddata(v=vs.80).aspx) object by loading the raw XML data from the previously created **XmlElement** object.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229740(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

EncryptedData edElement = new EncryptedData();

edElement.LoadXml(encryptedElement);

1. Create a new [EncryptedXml](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml.encryptedxml(v=vs.80).aspx) object and use it to decrypt the XML data using the same key that was used for encryption.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229740(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-3)

EncryptedXml exml = new EncryptedXml();

// Decrypt the element using the symmetric key.

byte[] rgbOutput = exml.DecryptData(edElement, Alg);

1. Replace the encrypted element with the newly decrypted plaintext element within the XML document.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229740(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-4)

exml.ReplaceData(encryptedElement, rgbOutput);

Example

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229740(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-5)

using System;

using System.Xml;

using System.Security.Cryptography;

using System.Security.Cryptography.Xml;

namespace CSCrypto

{

class Program

{

static void Main(string[] args)

{

RijndaelManaged key = null;

try

{

// Create a new Rijndael key.

key = new RijndaelManaged();

// Load an XML document.

XmlDocument xmlDoc = new XmlDocument();

xmlDoc.PreserveWhitespace = true;

xmlDoc.Load("test.xml");

// Encrypt the "creditcard" element.

Encrypt(xmlDoc, "creditcard", key);

Console.WriteLine("The element was encrypted");

Console.WriteLine(xmlDoc.InnerXml);

Decrypt(xmlDoc, key);

Console.WriteLine("The element was decrypted");

Console.WriteLine(xmlDoc.InnerXml);

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

finally

{

// Clear the key.

if (key != null)

{

key.Clear();

}

}

}

public static void Encrypt(XmlDocument Doc, string ElementName, SymmetricAlgorithm Key)

{

// Check the arguments.

if (Doc == null)

throw new ArgumentNullException("Doc");

if (ElementName == null)

throw new ArgumentNullException("ElementToEncrypt");

if (Key == null)

throw new ArgumentNullException("Alg");

////////////////////////////////////////////////

// Find the specified element in the XmlDocument

// object and create a new XmlElemnt object.

////////////////////////////////////////////////

XmlElement elementToEncrypt = Doc.GetElementsByTagName(ElementName)[0] as XmlElement;

// Throw an XmlException if the element was not found.

if (elementToEncrypt == null)

{

throw new XmlException("The specified element was not found");

}

//////////////////////////////////////////////////

// Create a new instance of the EncryptedXml class

// and use it to encrypt the XmlElement with the

// symmetric key.

//////////////////////////////////////////////////

EncryptedXml eXml = new EncryptedXml();

byte[] encryptedElement = eXml.EncryptData(elementToEncrypt, Key, false);

////////////////////////////////////////////////

// Construct an EncryptedData object and populate

// it with the desired encryption information.

////////////////////////////////////////////////

EncryptedData edElement = new EncryptedData();

edElement.Type = EncryptedXml.XmlEncElementUrl;

// Create an EncryptionMethod element so that the

// receiver knows which algorithm to use for decryption.

// Determine what kind of algorithm is being used and

// supply the appropriate URL to the EncryptionMethod element.

string encryptionMethod = null;

if (Key is TripleDES)

{

encryptionMethod = EncryptedXml.XmlEncTripleDESUrl;

}

else if (Key is DES)

{

encryptionMethod = EncryptedXml.XmlEncDESUrl;

}

if (Key is Rijndael)

{

switch (Key.KeySize)

{

case 128:

encryptionMethod = EncryptedXml.XmlEncAES128Url;

break;

case 192:

encryptionMethod = EncryptedXml.XmlEncAES192Url;

break;

case 256:

encryptionMethod = EncryptedXml.XmlEncAES256Url;

break;

}

}

else

{

// Throw an exception if the transform is not in the previous categories

throw new CryptographicException("The specified algorithm is not supported for XML Encryption.");

}

edElement.EncryptionMethod = new EncryptionMethod(encryptionMethod);

// Add the encrypted element data to the

// EncryptedData object.

edElement.CipherData.CipherValue = encryptedElement;

////////////////////////////////////////////////////

// Replace the element from the original XmlDocument

// object with the EncryptedData element.

////////////////////////////////////////////////////

EncryptedXml.ReplaceElement(elementToEncrypt, edElement, false);

}

public static void Decrypt(XmlDocument Doc, SymmetricAlgorithm Alg)

{

// Check the arguments.

if (Doc == null)

throw new ArgumentNullException("Doc");

if (Alg == null)

throw new ArgumentNullException("Alg");

// Find the EncryptedData element in the XmlDocument.

XmlElement encryptedElement = Doc.GetElementsByTagName("EncryptedData")[0] as XmlElement;

// If the EncryptedData element was not found, throw an exception.

if (encryptedElement == null)

{

throw new XmlException("The EncryptedData element was not found.");

}

// Create an EncryptedData object and populate it.

EncryptedData edElement = new EncryptedData();

edElement.LoadXml(encryptedElement);

// Create a new EncryptedXml object.

EncryptedXml exml = new EncryptedXml();

// Decrypt the element using the symmetric key.

byte[] rgbOutput = exml.DecryptData(edElement, Alg);

// Replace the encryptedData element with the plaintext XML element.

exml.ReplaceData(encryptedElement, rgbOutput);

}

}

}

This example assumes that a file named "test.xml" exists in the same directory as the compiled program. It also assumes that "test.xml"contains a "creditcard" element. You can place the following XML into a file called test.xml and use it with this example.

<root>

<creditcard>

<number>19834209</number>

<expiry>02/02/2002</expiry>

</creditcard>

</root>

**Compiling the Code**

* To compile this example, you need to include a reference to **System.Security.dll**.
* Include the following namespaces: [System.Xml](https://msdn.microsoft.com/en-US/library/system.xml(v=vs.80).aspx), [System.Security.Cryptography](https://msdn.microsoft.com/en-US/library/system.security.cryptography(v=vs.80).aspx), and **System.Security.Cryptography.Xml**.

**Security**

Never store a cryptographic key in plaintext or transfer a key between machines in plaintext.

When you are done using a symmetric cryptographic key, clear it from memory by setting each byte to zero or by calling the [Clear](https://msdn.microsoft.com/en-US/library/system.security.cryptography.symmetricalgorithm.clear(v=vs.80).aspx) method of the managed cryptography class.

##### How to: Encrypt XML Elements with Asymmetric Keys

**.NET Framework 2.0**

[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635736413806036745

You can use the classes in the [System.Security.Cryptography.Xml](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml(v=vs.80).aspx) namespace to encrypt an element within an XML document. XML Encryption is a standard way to exchange or store encrypted XML data, without worrying about the data being easily read. For more information about the XML Encryption standard, see the World Wide Web Consortium (W3C) specification for XML Encryption located at http://www.w3.org/TR/xmldsig-core/.

You can use XML Encryption to replace any XML element or document with an <**EncryptedData**> element that contains the encrypted XML data. The <**EncryptedData**> element can also contain sub elements that include information about the keys and processes used during encryption. XML Encryption allows a document to contain multiple encrypted elements and allows an element to be encrypted multiple times. The code example in this procedure shows how to create an <**EncryptedData**> element along with several other sub elements that you can use later during decryption.

This example encrypts an XML element using two keys. It generates an RSA public/private key pair and saves the key pair to a secure key container. The example then creates a separate session key using the Advanced Encryption Standard (AES) algorithm, also called the Rijndael algorithm. The example uses the AES session key to encrypt the XML document and then uses the RSA public key to encrypt the AES session key. Finally, the example saves the encrypted AES session key and the encrypted XML data to the XML document within a new <**EncryptedData**> element.

To decrypt the XML element, you retrieve the RSA private key from the key container, use it to decrypt the session key, and then use the session key to decrypt the document. For more information about how to decrypt an XML element that was encrypted using this procedure, see [How to: Decrypt XML Elements with Asymmetric Keys](https://msdn.microsoft.com/en-US/library/ms229919(v=vs.80).aspx).

This example is appropriate for situations where multiple applications need to share encrypted data or where an application needs to save encrypted data between the times that it runs.

**To encrypt an XML element with an asymmetric key**

1. Create a [CspParameters](https://msdn.microsoft.com/en-US/library/system.security.cryptography.cspparameters(v=vs.80).aspx) object and specify the name of the key container.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229746(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

CspParameters cspParams = new CspParameters();

cspParams.KeyContainerName = "XML\_ENC\_RSA\_KEY";

1. Generate a symmetric key using the [RSACryptoServiceProvider](https://msdn.microsoft.com/en-US/library/system.security.cryptography.rsacryptoserviceprovider(v=vs.80).aspx) class. The key is automatically saved to the key container when you pass the **CspParameters** object to the constructor of the **RSACryptoServiceProvider** class. This key will be used to encrypt the AES session key and can be retrieved later to decrypt it.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229746(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

RSACryptoServiceProvider rsaKey = new RSACryptoServiceProvider(cspParams);

1. Create an [XmlDocument](https://msdn.microsoft.com/en-US/library/system.xml.xmldocument(v=vs.80).aspx) object by loading an XML file from disk. The **XmlDocument** object contains the XML element to encrypt.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229746(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-3)

// Create an XmlDocument object.

XmlDocument xmlDoc = new XmlDocument();

// Load an XML file into the XmlDocument object.

try

{

xmlDoc.PreserveWhitespace = true;

xmlDoc.Load("test.xml");

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

1. Find the specified element in the **XmlDocument** object and create a new [XmlElement](https://msdn.microsoft.com/en-US/library/system.xml.xmlelement(v=vs.80).aspx) object to represent the element you want to encrypt. In this example, the "creditcard" element is encrypted.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229746(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-4)

XmlElement elementToEncrypt = Doc.GetElementsByTagName(ElementToEncrypt)[0] as XmlElement;

// Throw an XmlException if the element was not found.

if (elementToEncrypt == null)

{

throw new XmlException("The specified element was not found");

}

1. Create a new session key using the [RijndaelManaged](https://msdn.microsoft.com/en-US/library/system.security.cryptography.rijndaelmanaged(v=vs.80).aspx) class. This key will encrypt the XML element, and then be encrypted itself and placed in the XML document.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229746(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-5)

// Create a 256 bit Rijndael key.

sessionKey = new RijndaelManaged();

sessionKey.KeySize = 256;

1. Create a new instance of the [EncryptedXml](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml.encryptedxml(v=vs.80).aspx) class and use it to encrypt the specified element using the session key. The [EncryptData](https://msdn.microsoft.com/en-US/library/y259436w(v=vs.80).aspx)method returns the encrypted element as an array of encrypted bytes.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229746(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-6)

EncryptedXml eXml = new EncryptedXml();

byte[] encryptedElement = eXml.EncryptData(elementToEncrypt, sessionKey, false);

1. Construct an [EncryptedData](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml.encrypteddata(v=vs.80).aspx) object and populate it with the URL identifier of the encrypted XML element. This URL identifier lets a decrypting party know that the XML contains an encrypted element. You can use the [XmlEncElementUrl](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml.encryptedxml.xmlencelementurl(v=vs.80).aspx) field to specify the URL identifier. The plaintext XML element will be replaced by an <**EncrypotedData**> element encapsulated by this **EncryptedData** object.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229746(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-7)

EncryptedData edElement = new EncryptedData();

edElement.Type = EncryptedXml.XmlEncElementUrl;

edElement.Id = EncryptionElementID;

1. Create an [EncryptionMethod](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml.encryptionmethod(v=vs.80).aspx) object that is initialized to the URL identifier of the cryptographic algorithm used to generate the session key. Pass the **EncryptionMethod** object to the [EncryptionMethod](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml.encryptedtype.encryptionmethod(v=vs.80).aspx) property.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229746(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-8)

edElement.EncryptionMethod = new EncryptionMethod(EncryptedXml.XmlEncAES256Url);

1. Create an [EncryptedKey](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml.encryptedkey(v=vs.80).aspx) object to contain the encrypted session key. Encrypt the session key, add it to the **EncryptedKey** object, and enter a session key name and key identifier URL.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229746(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-9)

EncryptedKey ek = new EncryptedKey();

byte[] encryptedKey = EncryptedXml.EncryptKey(sessionKey.Key, Alg, false);

ek.CipherData = new CipherData(encryptedKey);

ek.EncryptionMethod = new EncryptionMethod(EncryptedXml.XmlEncRSA15Url);

1. Create a new [DataReference](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml.datareference(v=vs.80).aspx) object that maps the encrypted data to a particular session key. This optional step allows you to easily specify that multiple parts of an XML document were encrypted by a single key.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229746(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-10)

DataReference dRef = new DataReference();

// Specify the EncryptedData URI.

dRef.Uri = "#" + EncryptionElementID;

// Add the DataReference to the EncryptedKey.

ek.AddReference(dRef);

1. Add the encrypted key to the **EncryptedData** object.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229746(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-11)

edElement.KeyInfo.AddClause(new KeyInfoEncryptedKey(ek));

1. Create a new [KeyInfo](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml.keyinfo(v=vs.80).aspx) object to specify the name of the RSA key. Add it to the **EncryptedData** object. This helps the decrypting party identify the correct asymmetric key to use when decrypting the session key.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229746(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-12)

// Create a new KeyInfo element.

edElement.KeyInfo = new KeyInfo();

// Create a new KeyInfoName element.

KeyInfoName kin = new KeyInfoName();

// Specify a name for the key.

kin.Value = KeyName;

// Add the KeyInfoName element to the

// EncryptedKey object.

ek.KeyInfo.AddClause(kin);

1. Add the encrypted element data to the **EncryptedData** object.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229746(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-13)

edElement.CipherData.CipherValue = encryptedElement;

1. Replace the element from the original **XmlDocument** object with the **EncryptedData** element.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229746(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-14)

EncryptedXml.ReplaceElement(elementToEncrypt, edElement, false);

1. Save the **XmlDocument** object.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229746(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-15)

xmlDoc.Save("test.xml");

Example

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229746(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-16)

using System;

using System.Xml;

using System.Security.Cryptography;

using System.Security.Cryptography.Xml;

class Program

{

static void Main(string[] args)

{

// Create an XmlDocument object.

XmlDocument xmlDoc = new XmlDocument();

// Load an XML file into the XmlDocument object.

try

{

xmlDoc.PreserveWhitespace = true;

xmlDoc.Load("test.xml");

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

// Create a new CspParameters object to specify

// a key container.

CspParameters cspParams = new CspParameters();

cspParams.KeyContainerName = "XML\_ENC\_RSA\_KEY";

// Create a new RSA key and save it in the container. This key will encrypt

// a symmetric key, which will then be encryped in the XML document.

RSACryptoServiceProvider rsaKey = new RSACryptoServiceProvider(cspParams);

try

{

// Encrypt the "creditcard" element.

Encrypt(xmlDoc, "creditcard", "EncryptedElement1", rsaKey, "rsaKey");

// Save the XML document.

xmlDoc.Save("test.xml");

// Display the encrypted XML to the console.

Console.WriteLine("Encrypted XML:");

Console.WriteLine();

Console.WriteLine(xmlDoc.OuterXml);

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

finally

{

// Clear the RSA key.

rsaKey.Clear();

}

Console.ReadLine();

}

public static void Encrypt(XmlDocument Doc, string ElementToEncrypt, string EncryptionElementID, RSA Alg, string KeyName)

{

// Check the arguments.

if (Doc == null)

throw new ArgumentNullException("Doc");

if (ElementToEncrypt == null)

throw new ArgumentNullException("ElementToEncrypt");

if (EncryptionElementID == null)

throw new ArgumentNullException("EncryptionElementID");

if (Alg == null)

throw new ArgumentNullException("Alg");

if (KeyName == null)

throw new ArgumentNullException("KeyName");

////////////////////////////////////////////////

// Find the specified element in the XmlDocument

// object and create a new XmlElemnt object.

////////////////////////////////////////////////

XmlElement elementToEncrypt = Doc.GetElementsByTagName(ElementToEncrypt)[0] as XmlElement;

// Throw an XmlException if the element was not found.

if (elementToEncrypt == null)

{

throw new XmlException("The specified element was not found");

}

RijndaelManaged sessionKey = null;

try

{

//////////////////////////////////////////////////

// Create a new instance of the EncryptedXml class

// and use it to encrypt the XmlElement with the

// a new random symmetric key.

//////////////////////////////////////////////////

// Create a 256 bit Rijndael key.

sessionKey = new RijndaelManaged();

sessionKey.KeySize = 256;

EncryptedXml eXml = new EncryptedXml();

byte[] encryptedElement = eXml.EncryptData(elementToEncrypt, sessionKey, false);

////////////////////////////////////////////////

// Construct an EncryptedData object and populate

// it with the desired encryption information.

////////////////////////////////////////////////

EncryptedData edElement = new EncryptedData();

edElement.Type = EncryptedXml.XmlEncElementUrl;

edElement.Id = EncryptionElementID;

// Create an EncryptionMethod element so that the

// receiver knows which algorithm to use for decryption.

edElement.EncryptionMethod = new EncryptionMethod(EncryptedXml.XmlEncAES256Url);

// Encrypt the session key and add it to an EncryptedKey element.

EncryptedKey ek = new EncryptedKey();

byte[] encryptedKey = EncryptedXml.EncryptKey(sessionKey.Key, Alg, false);

ek.CipherData = new CipherData(encryptedKey);

ek.EncryptionMethod = new EncryptionMethod(EncryptedXml.XmlEncRSA15Url);

// Create a new DataReference element

// for the KeyInfo element. This optional

// element specifies which EncryptedData

// uses this key. An XML document can have

// multiple EncryptedData elements that use

// different keys.

DataReference dRef = new DataReference();

// Specify the EncryptedData URI.

dRef.Uri = "#" + EncryptionElementID;

// Add the DataReference to the EncryptedKey.

ek.AddReference(dRef);

// Add the encrypted key to the

// EncryptedData object.

edElement.KeyInfo.AddClause(new KeyInfoEncryptedKey(ek));

// Set the KeyInfo element to specify the

// name of the RSA key.

// Create a new KeyInfo element.

edElement.KeyInfo = new KeyInfo();

// Create a new KeyInfoName element.

KeyInfoName kin = new KeyInfoName();

// Specify a name for the key.

kin.Value = KeyName;

// Add the KeyInfoName element to the

// EncryptedKey object.

ek.KeyInfo.AddClause(kin);

// Add the encrypted element data to the

// EncryptedData object.

edElement.CipherData.CipherValue = encryptedElement;

////////////////////////////////////////////////////

// Replace the element from the original XmlDocument

// object with the EncryptedData element.

////////////////////////////////////////////////////

EncryptedXml.ReplaceElement(elementToEncrypt, edElement, false);

}

catch(Exception e)

{

// re-throw the exception.

throw e;

}

finally

{

if (sessionKey != null)

{

sessionKey.Clear();

}

}

}

}

This example assumes that a file named "test.xml" exists in the same directory as the compiled program. It also assumes that "test.xml"contains a "creditcard" element. You can place the following XML into a file called test.xml and use it with this example.

<root>

<creditcard>

<number>19834209</number>

<expiry>02/02/2002</expiry>

</creditcard>

</root>

**Compiling the Code**

* To compile this example, you need to include a reference to **System.Security.dll**.
* Include the following namespaces: [System.Xml](https://msdn.microsoft.com/en-US/library/system.xml(v=vs.80).aspx), [System.Security.Cryptography](https://msdn.microsoft.com/en-US/library/system.security.cryptography(v=vs.80).aspx), and **System.Security.Cryptography.Xml**.

**Security**

Never store a symmetric cryptographic key in plaintext or transfer a symmetric key between machines in plaintext. Additionally, never store or transfer the private key of an asymmetric key pair in plaintext. For more information about symmetric and asymmetric cryptographic keys, see[Generating Keys for Encryption and Decryption](https://msdn.microsoft.com/en-US/library/5e9ft273(v=vs.80).aspx).

Never embed a key directly into your source code. Embedded keys can be easily read from an assembly using the [MSIL Disassembler (Ildasm.exe)](https://msdn.microsoft.com/en-US/library/f7dy01k1(v=vs.80).aspx) or by opening the assembly in a text editor such as Notepad.

When you are done using a cryptographic key, clear it from memory by setting each byte to zero or by calling the [Clear](https://msdn.microsoft.com/en-US/library/system.security.cryptography.symmetricalgorithm.clear(v=vs.80).aspx) method of the managed cryptography class. Cryptographic keys can sometimes be read from memory by a debugger or read from a hard drive if the memory location is paged to disk.

##### How to: Decrypt XML Elements with Asymmetric Keys

**.NET Framework 2.0 ---**

[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635736413806036745

You can use the classes in the [System.Security.Cryptography.Xml](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml(v=vs.80).aspx) namespace to encrypt and decrypt an element within an XML document. XML Encryption is a standard way to exchange or store encrypted XML data, without worrying about the data being easily read. For more information about the XML Encryption standard, see the World Wide Web Consortium (W3C) specification for XML Encryption located at http://www.w3.org/TR/xmldsig-core/.

The example in this procedure decrypts an XML element that was encrypted using the methods described in: [How to: Encrypt XML Elements with Asymmetric Keys](https://msdn.microsoft.com/en-US/library/ms229746(v=vs.80).aspx). It finds an <**EncryptedData**> element, decrypts the element, and then replaces the element with the original plaintext XML element.

This example decrypts an XML element using two keys. It retrieves a previously generated RSA private key from a key container, and then uses the RSA key to decrypt a session key stored in the <**EncryptedKey**> element of the <**EncryptedData**> element. The example then uses the session key to decrypt the XML element.

This example is appropriate for situations where multiple applications need to share encrypted data or where an application needs to save encrypted data between the times that it runs.

**To decrypt an XML element with an asymmetric key**

1. Create a [CspParameters](https://msdn.microsoft.com/en-US/library/system.security.cryptography.cspparameters(v=vs.80).aspx) object and specify the name of the key container.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229919(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

CspParameters cspParams = new CspParameters();

cspParams.KeyContainerName = "XML\_ENC\_RSA\_KEY";

1. Retrieve a previously generated asymmetric key from the container using the [RSACryptoServiceProvider](https://msdn.microsoft.com/en-US/library/system.security.cryptography.rsacryptoserviceprovider(v=vs.80).aspx) object. The key is automatically retrieved from the key container when you pass the **CspParameters** object to the constructor of the**RSACryptoServiceProvider** constructor.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229919(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

RSACryptoServiceProvider rsaKey = new RSACryptoServiceProvider(cspParams);

1. Create new [EncryptedXml](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml.encryptedxml(v=vs.80).aspx) object to decrypt the document.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229919(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-3)

XmlDocument xmlDoc = new XmlDocument();

// Load an XML file into the XmlDocument object.

try

{

xmlDoc.PreserveWhitespace = true;

xmlDoc.Load("test.xml");

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

1. Add a key/name mapping to associate the RSA key with the element within the document that should be decrypted. You must use the same name for the key that you used when you encrypted the document. Note that this name is separate from the name used to identify the key in the key container specified in step 1.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229919(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-4)

// Create a new EncryptedXml object.

EncryptedXml exml = new EncryptedXml(Doc);

1. Call the [DecryptDocument](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml.encryptedxml.decryptdocument(v=vs.80).aspx) method to decrypt the <**EncryptedData**> element. This method uses the RSA key to decrypt the session key and automatically uses the session key to decrypt the XML element. It also automatically replaces the <**EncryptedData**> element with the original plaintext.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229919(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-5)

exml.AddKeyNameMapping(KeyName, Alg);

1. Save the XML document.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229919(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-6)

exml.DecryptDocument();

Example

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229919(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-7)

using System;

using System.Xml;

using System.Security.Cryptography;

using System.Security.Cryptography.Xml;

class Program

{

static void Main(string[] args)

{

// Create an XmlDocument object.

XmlDocument xmlDoc = new XmlDocument();

// Load an XML file into the XmlDocument object.

try

{

xmlDoc.PreserveWhitespace = true;

xmlDoc.Load("test.xml");

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

CspParameters cspParams = new CspParameters();

cspParams.KeyContainerName = "XML\_ENC\_RSA\_KEY";

// Get the RSA key from the key container. This key will decrypt

// a symmetric key that was imbedded in the XML document.

RSACryptoServiceProvider rsaKey = new RSACryptoServiceProvider(cspParams);

try

{

// Decrypt the elements.

Decrypt(xmlDoc, rsaKey, "rsaKey");

// Save the XML document.

xmlDoc.Save("test.xml");

// Display the encrypted XML to the console.

Console.WriteLine();

Console.WriteLine("Decrypted XML:");

Console.WriteLine();

Console.WriteLine(xmlDoc.OuterXml);

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

finally

{

// Clear the RSA key.

rsaKey.Clear();

}

Console.ReadLine();

}

public static void Decrypt(XmlDocument Doc, RSA Alg, string KeyName)

{

// Check the arguments.

if (Doc == null)

throw new ArgumentNullException("Doc");

if (Alg == null)

throw new ArgumentNullException("Alg");

if (KeyName == null)

throw new ArgumentNullException("KeyName");

// Create a new EncryptedXml object.

EncryptedXml exml = new EncryptedXml(Doc);

// Add a key-name mapping.

// This method can only decrypt documents

// that present the specified key name.

exml.AddKeyNameMapping(KeyName, Alg);

// Decrypt the element.

exml.DecryptDocument();

}

}

This example assumes that a file named "test.xml" exists in the same directory as the compiled program. It also assumes that "test.xml"contains an XML element that was encrypted using the techniques described in [How to: Encrypt XML Elements with Asymmetric Keys](https://msdn.microsoft.com/en-US/library/ms229746(v=vs.80).aspx).

**Compiling the Code**

* To compile this example, you need to include a reference to **System.Security.dll**.
* Include the following namespaces: [System.Xml](https://msdn.microsoft.com/en-US/library/system.xml(v=vs.80).aspx), [System.Security.Cryptography](https://msdn.microsoft.com/en-US/library/system.security.cryptography(v=vs.80).aspx), and **System.Security.Cryptography.Xml**.

**Security**

Never store a symmetric cryptographic key in plaintext or transfer a symmetric key between machines in plaintext. Additionally, never store or transfer the private key of an asymmetric key pair in plaintext. For more information about symmetric and asymmetric cryptographic keys, see[Generating Keys for Encryption and Decryption](https://msdn.microsoft.com/en-US/library/5e9ft273(v=vs.80).aspx).

Never embed a key directly into your source code. Embedded keys can be easily read from an assembly using the [MSIL Disassembler (Ildasm.exe)](https://msdn.microsoft.com/en-US/library/f7dy01k1(v=vs.80).aspx) or by opening the assembly in a text editor such as Notepad.

When you are done using a cryptographic key, clear it from memory by setting each byte to zero or by calling the [Clear](https://msdn.microsoft.com/en-US/library/system.security.cryptography.symmetricalgorithm.clear(v=vs.80).aspx) method of the managed cryptography class. Cryptographic keys can sometimes be read from memory by a debugger or read from a hard drive if the memory location is paged to disk.

##### How to: Encrypt XML Elements with X.509 Certificates

**.NET Framework 2.0**

[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635736413806036745

You can use the classes in the [System.Security.Cryptography.Xml](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml(v=vs.80).aspx) namespace to encrypt an element within an XML document. XML Encryption is a standard way to exchange or store encrypted XML data, without worrying about the data being easily read. For more information about the XML Encryption standard, see the World Wide Web Consortium (W3C) specification for XML Encryption located at http://www.w3.org/TR/xmldsig-core/.

You can use XML Encryption to replace any XML element or document with an <**EncryptedData**> element that contains the encrypted XML data. The <**EncryptedData**> element can contain sub elements that include information about the keys and processes used during encryption. XML Encryption allows a document to contain multiple encrypted elements and allows an element to be encrypted multiple times. The code example in this procedure shows you how to create an <**EncryptedData**> element along with several other sub elements that you can use later during decryption.

This example encrypts an XML element using two keys. It generates a test X.509 certificate using the [Certificate Creation Tool (Makecert.exe)](https://msdn.microsoft.com/en-US/library/bfsktky3(v=vs.80).aspx)and saves the certificate to a certificate store. The example then programmatically retrieves the certificate and uses it to encrypt an XML element using the [Encrypt](https://msdn.microsoft.com/en-US/library/ms148633(v=vs.80).aspx) method. Internally, the **Encrypt** method creates a separate session key and uses it to encrypt the XML document. This method encrypts the session key and saves it along with the encrypted XML within a new <**EncryptedData**> element.

To decrypt the XML element, simply call the [DecryptDocument](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml.encryptedxml.decryptdocument(v=vs.80).aspx) method, which automatically retrieves the X.509 certificate from the store and performs the necessary decryption. For more information about how to decrypt an XML element that was encrypted using this procedure, see[How to: Decrypt XML Elements with X.509 Certificates](https://msdn.microsoft.com/en-US/library/ms229943(v=vs.80).aspx).

This example is appropriate for situations where multiple applications need to share encrypted data or where an application needs to save encrypted data between the times that it runs.

**To encrypt an XML element with an X.509 certificate**

1. Use the [Certificate Creation Tool (Makecert.exe)](https://msdn.microsoft.com/en-US/library/bfsktky3(v=vs.80).aspx) to generate a test X.509 certificate and place it in the local user store. You must generate an exchange key and you must make the key exportable. Run the following command:
2. makecert -r -pe -n "CN=XML\_ENC\_TEST\_CERT" -b 01/01/2005 -e 01/01/2010 -sky exchange -ss my
3. Create an [X509Store](https://msdn.microsoft.com/en-US/library/system.security.cryptography.x509certificates.x509store(v=vs.80).aspx) object and initialize it to open the current user store.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229744(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

X509Store store = new X509Store(StoreLocation.CurrentUser);

1. Open the store in read-only mode.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229744(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-3)

store.Open(OpenFlags.ReadOnly);

1. Initialize an [X509Certificate2Collection](https://msdn.microsoft.com/en-US/library/system.security.cryptography.x509certificates.x509certificate2collection(v=vs.80).aspx) with all of the certificates in the store.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229744(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-4)

X509Certificate2Collection certCollection = store.Certificates;

1. Enumerate through the certificates in the store and find the certificate with the appropriate name. In this example, the certificate is named "CN=XML\_ENC\_TEST\_CERT".

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229744(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-5)

X509Certificate2 cert = null;

// Loop through each certificate and find the certificate

// with the appropriate name.

foreach (X509Certificate2 c in certCollection)

{

if (c.Subject == "CN=XML\_ENC\_TEST\_CERT")

{

cert = c;

break;

}

}

1. Close the store after the certificate is located.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229744(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-6)

store.Close();

1. Create an [XmlDocument](https://msdn.microsoft.com/en-US/library/system.xml.xmldocument(v=vs.80).aspx) object by loading an XML file from disk. The **XmlDocument** object contains the XML element to encrypt.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229744(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-7)

XmlDocument xmlDoc = new XmlDocument();

1. Find the specified element in the **XmlDocument** object and create a new [XmlElement](https://msdn.microsoft.com/en-US/library/system.xml.xmlelement(v=vs.80).aspx) object to represent the element you want to encrypt. In this example, the "creditcard" element is encrypted.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229744(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-8)

XmlElement elementToEncrypt = Doc.GetElementsByTagName(ElementToEncrypt)[0] as XmlElement;

1. Create a new instance of the [EncryptedXml](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml.encryptedxml(v=vs.80).aspx) class and use it to encrypt the specified element using the X.509 certificate. The **Encrypt**method returns the encrypted element as an [EncryptedData](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml.encrypteddata(v=vs.80).aspx) object.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229744(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-9)

EncryptedXml eXml = new EncryptedXml();

// Encrypt the element.

EncryptedData edElement = eXml.Encrypt(elementToEncrypt, Cert);

1. Replace the element from the original **XmlDocument** object with the **EncryptedData** element.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229744(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-10)

EncryptedXml.ReplaceElement(elementToEncrypt, edElement, false);

1. Save the **XmlDocument** object.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229744(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-11)

xmlDoc.Save("test.xml");

Example

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229744(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-12)

using System;

using System.Xml;

using System.Security.Cryptography;

using System.Security.Cryptography.Xml;

using System.Security.Cryptography.X509Certificates;

class Program

{

static void Main(string[] args)

{

try

{

// Create an XmlDocument object.

XmlDocument xmlDoc = new XmlDocument();

// Load an XML file into the XmlDocument object.

xmlDoc.PreserveWhitespace = true;

xmlDoc.Load("test.xml");

// Open the X.509 "Current User" store in read only mode.

X509Store store = new X509Store(StoreLocation.CurrentUser);

store.Open(OpenFlags.ReadOnly);

// Place all certificates in an X509Certificate2Collection object.

X509Certificate2Collection certCollection = store.Certificates;

X509Certificate2 cert = null;

// Loop through each certificate and find the certificate

// with the appropriate name.

foreach (X509Certificate2 c in certCollection)

{

if (c.Subject == "CN=XML\_ENC\_TEST\_CERT")

{

cert = c;

break;

}

}

if (cert == null)

{

throw new CryptographicException("The X.509 certificate could not be found.");

}

// Close the store.

store.Close();

// Encrypt the "creditcard" element.

Encrypt(xmlDoc, "creditcard", cert);

// Save the XML document.

xmlDoc.Save("test.xml");

// Display the encrypted XML to the console.

Console.WriteLine("Encrypted XML:");

Console.WriteLine();

Console.WriteLine(xmlDoc.OuterXml);

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

}

public static void Encrypt(XmlDocument Doc, string ElementToEncrypt, X509Certificate2 Cert)

{

// Check the arguments.

if (Doc == null)

throw new ArgumentNullException("Doc");

if (ElementToEncrypt == null)

throw new ArgumentNullException("ElementToEncrypt");

if (Cert == null)

throw new ArgumentNullException("Cert");

////////////////////////////////////////////////

// Find the specified element in the XmlDocument

// object and create a new XmlElemnt object.

////////////////////////////////////////////////

XmlElement elementToEncrypt = Doc.GetElementsByTagName(ElementToEncrypt)[0] as XmlElement;

// Throw an XmlException if the element was not found.

if (elementToEncrypt == null)

{

throw new XmlException("The specified element was not found");

}

//////////////////////////////////////////////////

// Create a new instance of the EncryptedXml class

// and use it to encrypt the XmlElement with the

// X.509 Certificate.

//////////////////////////////////////////////////

EncryptedXml eXml = new EncryptedXml();

// Encrypt the element.

EncryptedData edElement = eXml.Encrypt(elementToEncrypt, Cert);

////////////////////////////////////////////////////

// Replace the element from the original XmlDocument

// object with the EncryptedData element.

////////////////////////////////////////////////////

EncryptedXml.ReplaceElement(elementToEncrypt, edElement, false);

}

}

This example assumes that a file named "test.xml" exists in the same directory as the compiled program. It also assumes that "test.xml"contains a "creditcard" element. You can place the following XML into a file called test.xml and use it with this example.

<root>

<creditcard>

<number>19834209</number>

<expiry>02/02/2002</expiry>

</creditcard>

</root>

**Compiling the Code**

* To compile this example, you need to include a reference to **System.Security.dll**.
* Include the following namespaces: [System.Xml](https://msdn.microsoft.com/en-US/library/system.xml(v=vs.80).aspx), [System.Security.Cryptography](https://msdn.microsoft.com/en-US/library/system.security.cryptography(v=vs.80).aspx), and **System.Security.Cryptography.Xml**.

**Security**

The X.509 certificate used in this example is for test purposes only. Applications should use an X.509 certificate generated by a trusted certificate authority or use a certificate generated by the Microsoft Windows Certificate Server.

##### How to: Decrypt XML Elements with X.509 Certificates

**.NET Framework 2.0**

[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635736413806036745

You can use the classes in the [System.Security.Cryptography.Xml](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml(v=vs.80).aspx) namespace to encrypt and decrypt an element within an XML document. XML Encryption is a standard way to exchange or store encrypted XML data, without worrying about the data being easily read. For more information about the XML Encryption standard, see the World Wide Web Consortium (W3C) specification for XML Encryption located at http://www.w3.org/TR/xmldsig-core/.

This example decrypts an XML element that was encrypted using the methods described in: [How to: Encrypt XML Elements with X.509 Certificates](https://msdn.microsoft.com/en-US/library/ms229744(v=vs.80).aspx). It finds an <**EncryptedData**> element, decrypts the element, and then replaces the element with the original plaintext XML element.

The code example in this procedure decrypts an XML element using an X.509 certificate from the local certificate store of the current user account. The example uses the [DecryptDocument](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml.encryptedxml.decryptdocument(v=vs.80).aspx) method to automatically retrieve the X.509 certificate and decrypt a session key stored in the <**EncryptedKey**> element of the <**EncryptedData**> element. The **DecryptDocument** method then automatically uses the session key to decrypt the XML element.

This example is appropriate for situations where multiple applications need to share encrypted data or where an application needs to save encrypted data between the times that it runs.

**To decrypt an XML element with an X.509 certificate**

1. Create an [XmlDocument](https://msdn.microsoft.com/en-US/library/system.xml.xmldocument(v=vs.80).aspx) object by loading an XML file from disk. The **XmlDocument** object contains the XML element to decrypt.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229943(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

XmlDocument xmlDoc = new XmlDocument();

1. Create a new [EncryptedXml](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml.encryptedxml(v=vs.80).aspx) object by passing the **XmlDocument** object to the constructor.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229943(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

EncryptedXml exml = new EncryptedXml(Doc);

1. Decrypt the XML document using the **DecryptDocument** method.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229943(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-3)

exml.DecryptDocument();

1. Save the **XmlDocument** object.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229943(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-4)

xmlDoc.Save("test.xml");

Example

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229943(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-5)

using System;

using System.Xml;

using System.Security.Cryptography;

using System.Security.Cryptography.Xml;

using System.Security.Cryptography.X509Certificates;

class Program

{

static void Main(string[] args)

{

try

{

// Create an XmlDocument object.

XmlDocument xmlDoc = new XmlDocument();

// Load an XML file into the XmlDocument object.

xmlDoc.PreserveWhitespace = true;

xmlDoc.Load("test.xml");

// Decrypt the document.

Decrypt(xmlDoc);

// Save the XML document.

xmlDoc.Save("test.xml");

// Display the decrypted XML to the console.

Console.WriteLine("Decrypted XML:");

Console.WriteLine();

Console.WriteLine(xmlDoc.OuterXml);

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

}

public static void Decrypt(XmlDocument Doc)

{

// Check the arguments.

if (Doc == null)

throw new ArgumentNullException("Doc");

// Create a new EncryptedXml object.

EncryptedXml exml = new EncryptedXml(Doc);

// Decrypt the XML document.

exml.DecryptDocument();

}

}

This example assumes that a file named "test.xml" exists in the same directory as the compiled program. It also assumes that "test.xml"contains a "creditcard" element. You can place the following XML into a file called test.xml and use it with this example.

<root>

<creditcard>

<number>19834209</number>

<expiry>02/02/2002</expiry>

</creditcard>

</root>

**Compiling the Code**

* To compile this example, you need to include a reference to **System.Security.dll**.
* Include the following namespaces: [System.Xml](https://msdn.microsoft.com/en-US/library/system.xml(v=vs.80).aspx), [System.Security.Cryptography](https://msdn.microsoft.com/en-US/library/system.security.cryptography(v=vs.80).aspx), and **System.Security.Cryptography.Xml**.

**Security**

The X.509 certificate used in this example is for test purposes only. Applications should use an X.509 certificate generated by a trusted certificate authority or use a certificate generated by the Microsoft Windows Certificate Server.

##### How to: Sign XML Documents with Digital Signatures

**.NET Framework 2.0**

[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635736413806036745

You can use the classes in the [System.Security.Cryptography.Xml](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml(v=vs.80).aspx) namespace to sign an XML document or part of an XML document with a digital signature. XML digital signatures (XMLDSIG) allow you to verify that data was not altered after it was signed. For more information about the XMLDSIG standard, see the World Wide Web Consortium (W3C) specification at http://www.w3.org/TR/xmldsig-core/.

The code example in this procedure demonstrates how to digitally sign an entire XML document and attach the signature to the document in a <**Signature**> element. The example creates an RSA signing key, adds the key to a secure key container, and then uses the key to digitally sign an XML document. The key can then be retrieved to verify the XML digital signature, or be used to sign another XML document.

For information about how to verify an XML digital signature that was created using this procedure, see [How to: Verify the Digital Signatures of XML Documents](https://msdn.microsoft.com/en-US/library/ms229950(v=vs.80).aspx).

**To digitally sign an XML document**

1. Create a [CspParameters](https://msdn.microsoft.com/en-US/library/system.security.cryptography.cspparameters(v=vs.80).aspx) object and specify the name of the key container.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229745(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

CspParameters cspParams = new CspParameters();

cspParams.KeyContainerName = "XML\_DSIG\_RSA\_KEY";

1. Generate a symmetric key using the [RSACryptoServiceProvider](https://msdn.microsoft.com/en-US/library/system.security.cryptography.rsacryptoserviceprovider(v=vs.80).aspx) class. The key is automatically saved to the key container when you pass the **CspParameters** object to the constructor of the **RSACryptoServiceProvider** class. This key will be used to sign the XML document.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229745(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

RSACryptoServiceProvider rsaKey = new RSACryptoServiceProvider(cspParams);

1. Create an [XmlDocument](https://msdn.microsoft.com/en-US/library/system.xml.xmldocument(v=vs.80).aspx) object by loading an XML file from disk. The **XmlDocument** object contains the XML element to encrypt.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229745(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-3)

XmlDocument xmlDoc = new XmlDocument();

// Load an XML file into the XmlDocument object.

xmlDoc.PreserveWhitespace = true;

xmlDoc.Load("test.xml");

1. Create a new [SignedXml](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml.signedxml(v=vs.80).aspx) object and pass the **XmlDocument** object to it.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229745(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-4)

SignedXml signedXml = new SignedXml(Doc);

1. Add the signing RSA key to the **SignedXml** object.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229745(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-5)

signedXml.SigningKey = Key;

1. Create a [Reference](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml.reference(v=vs.80).aspx) object that describes what to sign. To sign the entire document, set the [Uri](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml.reference.uri(v=vs.80).aspx) property to "".

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229745(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-6)

// Create a reference to be signed.

Reference reference = new Reference();

reference.Uri = "";

1. Add an [XmlDsigEnvelopedSignatureTransform](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml.xmldsigenvelopedsignaturetransform(v=vs.80).aspx) object to the **Reference** object. A transformation allows the verifier to represent the XML data in the identical manner that the signer used. XML data can be represented in different ways, so this step is vital to verification.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229745(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-7)

XmlDsigEnvelopedSignatureTransform env = new XmlDsigEnvelopedSignatureTransform();

reference.AddTransform(env);

1. Add the **Reference** object to the **SignedXml** object.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229745(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-8)

signedXml.AddReference(reference);

1. Compute the signature by calling the [ComputeSignature](https://msdn.microsoft.com/en-US/library/x8xwex4b(v=vs.80).aspx) method.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229745(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-9)

signedXml.ComputeSignature();

1. Retrieve the XML representation of the signature (a <**Signature**> element) and save it to a new [XmlElement](https://msdn.microsoft.com/en-US/library/system.xml.xmlelement(v=vs.80).aspx) object.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229745(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-10)

XmlElement xmlDigitalSignature = signedXml.GetXml();

1. Append the element to the **XmlDocument** object.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229745(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-11)

Doc.DocumentElement.AppendChild(Doc.ImportNode(xmlDigitalSignature, true));

1. Save the document.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229745(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-12)

xmlDoc.Save("test.xml");

Example

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229745(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-13)

using System;

using System.Security.Cryptography;

using System.Security.Cryptography.Xml;

using System.Xml;

public class SignXML

{

public static void Main(String[] args)

{

try

{

// Create a new CspParameters object to specify

// a key container.

CspParameters cspParams = new CspParameters();

cspParams.KeyContainerName = "XML\_DSIG\_RSA\_KEY";

// Create a new RSA signing key and save it in the container.

RSACryptoServiceProvider rsaKey = new RSACryptoServiceProvider(cspParams);

// Create a new XML document.

XmlDocument xmlDoc = new XmlDocument();

// Load an XML file into the XmlDocument object.

xmlDoc.PreserveWhitespace = true;

xmlDoc.Load("test.xml");

// Sign the XML document.

SignXml(xmlDoc, rsaKey);

Console.WriteLine("XML file signed.");

// Save the document.

xmlDoc.Save("test.xml");

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

}

// Sign an XML file.

// This document cannot be verified unless the verifying

// code has the key with which it was signed.

public static void SignXml(XmlDocument Doc, RSA Key)

{

// Check arguments.

if (Doc == null)

throw new ArgumentException("Doc");

if (Key == null)

throw new ArgumentException("Key");

// Create a SignedXml object.

SignedXml signedXml = new SignedXml(Doc);

// Add the key to the SignedXml document.

signedXml.SigningKey = Key;

// Create a reference to be signed.

Reference reference = new Reference();

reference.Uri = "";

// Add an enveloped transformation to the reference.

XmlDsigEnvelopedSignatureTransform env = new XmlDsigEnvelopedSignatureTransform();

reference.AddTransform(env);

// Add the reference to the SignedXml object.

signedXml.AddReference(reference);

// Compute the signature.

signedXml.ComputeSignature();

// Get the XML representation of the signature and save

// it to an XmlElement object.

XmlElement xmlDigitalSignature = signedXml.GetXml();

// Append the element to the XML document.

Doc.DocumentElement.AppendChild(Doc.ImportNode(xmlDigitalSignature, true));

}

}

This example assumes that a file named "test.xml" exists in the same directory as the compiled program. You can place the following XML into a file called test.xml and use it with this example.

<root>

<creditcard>

<number>19834209</number>

<expiry>02/02/2002</expiry>

</creditcard>

</root>

**Compiling the Code**

* To compile this example, you need to include a reference to **System.Security.dll**.
* Include the following namespaces: [System.Xml](https://msdn.microsoft.com/en-US/library/system.xml(v=vs.80).aspx), [System.Security.Cryptography](https://msdn.microsoft.com/en-US/library/system.security.cryptography(v=vs.80).aspx), and **System.Security.Cryptography.Xml**.

**Security**

Never store or transfer the private key of an asymmetric key pair in plaintext. For more information about symmetric and asymmetric cryptographic keys, see [Generating Keys for Encryption and Decryption](https://msdn.microsoft.com/en-US/library/5e9ft273(v=vs.80).aspx).

Never embed a private key directly into your source code. Embedded keys can be easily read from an assembly using the [MSIL Disassembler (Ildasm.exe)](https://msdn.microsoft.com/en-US/library/f7dy01k1(v=vs.80).aspx) or by opening the assembly in a text editor such as Notepad.

##### How to: Verify the Digital Signatures of XML Documents

**.NET Framework 2.0**

[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635736413806036745

You can use the classes in the [System.Security.Cryptography.Xml](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml(v=vs.80).aspx) namespace to verify XML data signed with a digital signature. XML digital signatures (XMLDSIG) allow you to verify that data was not altered after it was signed. For more information about the XMLDSIG standard, see the World Wide Web Consortium (W3C) specification at http://www.w3.org/TR/xmldsig-core/.

The code example in this procedure demonstrates how to verify an XML digital signature contained in a <**Signature**> element. The example retrieves an RSA public key from a key container and then uses the key to verify the signature.

For information about how create a digital signature that can be verified using this technique, see [How to: Sign XML Documents with Digital Signatures](https://msdn.microsoft.com/en-US/library/ms229745(v=vs.80).aspx).

**To verify the digital signature of an XML document**

1. To verify the document, you must use the same asymmetric key that was used for signing. Create a [CspParameters](https://msdn.microsoft.com/en-US/library/system.security.cryptography.cspparameters(v=vs.80).aspx) object and specify the name of the key container that was used for signing.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229950(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

CspParameters cspParams = new CspParameters();

cspParams.KeyContainerName = "XML\_DSIG\_RSA\_KEY";

1. Retrieve the public key using the [RSACryptoServiceProvider](https://msdn.microsoft.com/en-US/library/system.security.cryptography.rsacryptoserviceprovider(v=vs.80).aspx) class. The key is automatically loaded from the key container by name when you pass the **CspParameters** object to the constructor of the **RSACryptoServiceProvider** class.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229950(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

RSACryptoServiceProvider rsaKey = new RSACryptoServiceProvider(cspParams);

1. Create an [XmlDocument](https://msdn.microsoft.com/en-US/library/system.xml.xmldocument(v=vs.80).aspx) object by loading an XML file from disk. The **XmlDocument** object contains the signed XML document to verify.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229950(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-3)

XmlDocument xmlDoc = new XmlDocument();

// Load an XML file into the XmlDocument object.

xmlDoc.PreserveWhitespace = true;

xmlDoc.Load("test.xml");

1. Create a new [SignedXml](https://msdn.microsoft.com/en-US/library/system.security.cryptography.xml.signedxml(v=vs.80).aspx) object and pass the **XmlDocument** object to it.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229950(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-4)

SignedXml signedXml = new SignedXml(Doc);

1. Find the <**signature**> element and create a new [XmlNodeList](https://msdn.microsoft.com/en-US/library/system.xml.xmlnodelist(v=vs.80).aspx) object.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229950(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-5)

XmlNodeList nodeList = Doc.GetElementsByTagName("Signature");

1. Load the XML of the first <**signature**> element into the **SignedXml** object.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229950(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-6)

signedXml.LoadXml((XmlElement)nodeList[0]);

1. Check the signature using the [CheckSignature](https://msdn.microsoft.com/en-US/library/kd4wwa16(v=vs.80).aspx) method and the RSA public key. This method returns a Boolean value that indicates success or failure.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229950(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-7)

return signedXml.CheckSignature(Key);

Example

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229950(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-8)

using System;

using System.Security.Cryptography;

using System.Security.Cryptography.Xml;

using System.Xml;

public class VerifyXML

{

public static void Main(String[] args)

{

try

{

// Create a new CspParameters object to specify

// a key container.

CspParameters cspParams = new CspParameters();

cspParams.KeyContainerName = "XML\_DSIG\_RSA\_KEY";

// Create a new RSA signing key and save it in the container.

RSACryptoServiceProvider rsaKey = new RSACryptoServiceProvider(cspParams);

// Create a new XML document.

XmlDocument xmlDoc = new XmlDocument();

// Load an XML file into the XmlDocument object.

xmlDoc.PreserveWhitespace = true;

xmlDoc.Load("test.xml");

// Verify the signature of the signed XML.

Console.WriteLine("Verifying signature...");

bool result = VerifyXml(xmlDoc, rsaKey);

// Display the results of the signature verification to

// the console.

if (result)

{

Console.WriteLine("The XML signature is valid.");

}

else

{

Console.WriteLine("The XML signature is not valid.");

}

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

}

// Verify the signature of an XML file against an asymmetric

// algorithm and return the result.

public static Boolean VerifyXml(XmlDocument Doc, RSA Key)

{

// Check arguments.

if (Doc == null)

throw new ArgumentException("Doc");

if (Key == null)

throw new ArgumentException("Key");

// Create a new SignedXml object and pass it

// the XML document class.

SignedXml signedXml = new SignedXml(Doc);

// Find the "Signature" node and create a new

// XmlNodeList object.

XmlNodeList nodeList = Doc.GetElementsByTagName("Signature");

// Throw an exception if no signature was found.

if (nodeList.Count <= 0)

{

throw new CryptographicException("Verification failed: No Signature was found in the document.");

}

// This example only supports one signature for

// the entire XML document. Throw an exception

// if more than one signature was found.

if (nodeList.Count >= 2)

{

throw new CryptographicException("Verification failed: More that one signature was found for the document.");

}

// Load the first <signature> node.

signedXml.LoadXml((XmlElement)nodeList[0]);

// Check the signature and return the result.

return signedXml.CheckSignature(Key);

}

}

This example assumes that a file named "test.xml" exists in the same directory as the compiled program. The "test.xml" file must be signed using the techniques described in [How to: Sign XML Documents with Digital Signatures](https://msdn.microsoft.com/en-US/library/ms229745(v=vs.80).aspx).

**Compiling the Code**

* To compile this example, you need to include a reference to **System.Security.dll**.
* Include the following namespaces: [System.Xml](https://msdn.microsoft.com/en-US/library/system.xml(v=vs.80).aspx), [System.Security.Cryptography](https://msdn.microsoft.com/en-US/library/system.security.cryptography(v=vs.80).aspx), and **System.Security.Cryptography.Xml**.

**Security**

Never store or transfer the private key of an asymmetric key pair in plaintext. For more information about symmetric and asymmetric cryptographic keys, see [Generating Keys for Encryption and Decryption](https://msdn.microsoft.com/en-US/library/5e9ft273(v=vs.80).aspx).

Never embed a private key directly into your source code. Embedded keys can be easily read from an assembly using the [MSIL Disassembler (Ildasm.exe)](https://msdn.microsoft.com/en-US/library/f7dy01k1(v=vs.80).aspx) or by opening the assembly in a text editor such as Notepad.

#### How to: Use Data Protection

**.NET Framework 2.0, 3.0, 3.5, 4, 4.5, 4.6**

The .NET Framework provides access to the data protection API (DPAPI), which allows you to encrypt data using information from the current user account or computer. When you use the DPAPI, you alleviate the difficult problem of explicitly generating and storing a cryptographic key.

Use the [ProtectedMemory](https://msdn.microsoft.com/en-US/library/system.security.cryptography.protectedmemory(v=vs.80).aspx) class to encrypt an array of in-memory bytes. This functionality is available in Microsoft Windows XP and later operating systems. You can specify that memory encrypted by the current process can be decrypted by the current process only, by all processes, or from the same user context. See the [MemoryProtectionScope](https://msdn.microsoft.com/en-US/library/system.security.cryptography.memoryprotectionscope(v=vs.80).aspx) enumeration for a detailed description of **ProtectedMemory**options.

Use the [ProtectedData](https://msdn.microsoft.com/en-US/library/system.security.cryptography.protecteddata(v=vs.80).aspx) class to encrypt a copy of an array of bytes. This functionality is available in Microsoft Windows 2000 and later operating systems. You can specify that data encrypted by the current user account can be decrypted only by the same user account, or you can specify that data encrypted by the current user account can be decrypted by any account on the computer. See the [DataProtectionScope](https://msdn.microsoft.com/en-US/library/system.security.cryptography.dataprotectionscope(v=vs.80).aspx)enumeration for a detailed description of **ProtectedData** options.

**To encrypt in-memory data using data protection**

* Call the static [Protect](https://msdn.microsoft.com/en-US/library/system.security.cryptography.protectedmemory.protect(v=vs.80).aspx) method while passing an array of bytes to encrypt, the entropy, and the memory protection scope.

**To decrypt in-memory data using data protection**

* Call the static [Unprotect](https://msdn.microsoft.com/en-US/library/system.security.cryptography.protectedmemory.unprotect(v=vs.80).aspx) method while passing an array of bytes to decrypt and the memory protection scope.

**To encrypt data to a file or stream using data protection**

1. Create random entropy.
2. Call the static [Protect](https://msdn.microsoft.com/en-US/library/2fh8203k(v=vs.80).aspx) method while passing an array of bytes to encrypt, the entropy, and the data protection scope.
3. Write the encrypted data to a file or stream.

**To decrypt data from a file or stream using data protection**

1. Read the encrypted data from a file or stream.
2. Call the static [Unprotect](https://msdn.microsoft.com/en-US/library/xh68ketz(v=vs.80).aspx) method while passing an array of bytes to decrypt and the data protection scope.

Example

The following code example demonstrates two forms of encryption and decryption. First, the code example encrypts and then decrypts an in-memory array of bytes. Next, the code example encrypts a copy of a byte array, saves it to a file, loads the data back from the file, and then decrypts the data. The example displays the original data, the encrypted data, and the decrypted data.

C#

[**VB**](https://msdn.microsoft.com/en-US/library/ms229741(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

using System;

using System.IO;

using System.Text;

using System.Security.Cryptography;

public class MemoryProtectionSample

{

public static void Main()

{

Run();

}

public static void Run()

{

try

{

///////////////////////////////

//

// Memory Encryption - ProtectedMemory

//

///////////////////////////////

// Create the original data to be encrypted (The data length should be a multiple of 16).

byte[] toEncrypt = UnicodeEncoding.ASCII.GetBytes("ThisIsSomeData16");

Console.WriteLine("Original data: " + UnicodeEncoding.ASCII.GetString(toEncrypt));

Console.WriteLine("Encrypting...");

// Encrypt the data in memory.

EncryptInMemoryData(toEncrypt, MemoryProtectionScope.SameLogon);

Console.WriteLine("Encrypted data: " + UnicodeEncoding.ASCII.GetString(toEncrypt));

Console.WriteLine("Decrypting...");

// Decrypt the data in memory.

DecryptInMemoryData(toEncrypt, MemoryProtectionScope.SameLogon);

Console.WriteLine("Decrypted data: " + UnicodeEncoding.ASCII.GetString(toEncrypt));

///////////////////////////////

//

// Data Encryption - ProtectedData

//

///////////////////////////////

// Create the original data to be encrypted

toEncrypt = UnicodeEncoding.ASCII.GetBytes("This is some data of any length.");

// Create a file.

FileStream fStream = new FileStream("Data.dat", FileMode.OpenOrCreate);

// Create some random entropy.

byte[] entropy = CreateRandomEntropy();

Console.WriteLine();

Console.WriteLine("Original data: " + UnicodeEncoding.ASCII.GetString(toEncrypt));

Console.WriteLine("Encrypting and writing to disk...");

// Encrypt a copy of the data to the stream.

int bytesWritten = EncryptDataToStream(toEncrypt, entropy, DataProtectionScope.CurrentUser, fStream);

fStream.Close();

Console.WriteLine("Reading data from disk and decrypting...");

// Open the file.

fStream = new FileStream("Data.dat", FileMode.Open);

// Read from the stream and decrypt the data.

byte[] decryptData = DecryptDataFromStream(entropy, DataProtectionScope.CurrentUser, fStream, bytesWritten);

fStream.Close();

Console.WriteLine("Decrypted data: " + UnicodeEncoding.ASCII.GetString(decryptData));

}

catch (Exception e)

{

Console.WriteLine("ERROR: " + e.Message);

}

}

public static void EncryptInMemoryData(byte[] Buffer, MemoryProtectionScope Scope )

{

if (Buffer.Length <= 0)

throw new ArgumentException("Buffer");

if (Buffer == null)

throw new ArgumentNullException("Buffer");

// Encrypt the data in memory. The result is stored in the same same array as the original data.

ProtectedMemory.Protect(Buffer, Scope);

}

public static void DecryptInMemoryData(byte[] Buffer, MemoryProtectionScope Scope)

{

if (Buffer.Length <= 0)

throw new ArgumentException("Buffer");

if (Buffer == null)

throw new ArgumentNullException("Buffer");

// Decrypt the data in memory. The result is stored in the same same array as the original data.

ProtectedMemory.Unprotect(Buffer, Scope);

}

public static byte[] CreateRandomEntropy()

{

// Create a byte array to hold the random value.

byte[] entropy = new byte[16];

// Create a new instance of the RNGCryptoServiceProvider.

// Fill the array with a random value.

new RNGCryptoServiceProvider().GetBytes(entropy);

// Return the array.

return entropy;

}

public static int EncryptDataToStream(byte[] Buffer, byte[] Entropy, DataProtectionScope Scope, Stream S)

{

if (Buffer.Length <= 0)

throw new ArgumentException("Buffer");

if (Buffer == null)

throw new ArgumentNullException("Buffer");

if (Entropy.Length <= 0)

throw new ArgumentException("Entropy");

if (Entropy == null)

throw new ArgumentNullException("Entropy");

if (S == null)

throw new ArgumentNullException("S");

int length = 0;

// Encrypt the data in memory. The result is stored in the same same array as the original data.

byte[] encrptedData = ProtectedData.Protect(Buffer, Entropy, Scope);

// Write the encrypted data to a stream.

if (S.CanWrite && encrptedData != null)

{

S.Write(encrptedData, 0, encrptedData.Length);

length = encrptedData.Length;

}

// Return the length that was written to the stream.

return length;

}

public static byte[] DecryptDataFromStream(byte[] Entropy, DataProtectionScope Scope, Stream S, int Length)

{

if (S == null)

throw new ArgumentNullException("S");

if (Length <= 0 )

throw new ArgumentException("Length");

if (Entropy == null)

throw new ArgumentNullException("Entropy");

if (Entropy.Length <= 0)

throw new ArgumentException("Entropy");

byte[] inBuffer = new byte[Length];

byte[] outBuffer;

// Read the encrypted data from a stream.

if (S.CanRead)

{

S.Read(inBuffer, 0, Length);

outBuffer = ProtectedData.Unprotect(inBuffer, Entropy, Scope);

}

else

{

throw new IOException("Could not read the stream.");

}

// Return the length that was written to the stream.

return outBuffer;

}

}

**Compiling the Code**

* Include a reference to **System.Security.dll**.
* Include the [System](https://msdn.microsoft.com/en-US/library/system(v=vs.80).aspx), [System.IO](https://msdn.microsoft.com/en-US/library/system.io(v=vs.80).aspx), [System.Security.Cryptography](https://msdn.microsoft.com/en-US/library/system.security.cryptography(v=vs.80).aspx), and [System.Text](https://msdn.microsoft.com/en-US/library/system.text(v=vs.80).aspx) namespace.

#### How to: Access Hardware Encryption Devices

**.NET Framework 2.0, 3.0, 3.5, 4, 4.5, 4.6**

You can use the [CspParameters](https://msdn.microsoft.com/en-US/library/system.security.cryptography.cspparameters(v=vs.90).aspx) class to access hardware encryption devices. For example, you can use this class to integrate your application with a smart card, a hardware random number generator, or a hardware implementation of a particular cryptographic algorithm.

The [CspParameters](https://msdn.microsoft.com/en-US/library/system.security.cryptography.cspparameters(v=vs.90).aspx) class creates a cryptographic service provider (CSP) that accesses a properly installed hardware encryption device. You can verify the availability of a CSP by inspecting the following registry key using the Registry Editor (Regedit.exe): HKEY\_LOCAL\_MACHINE\Software\Microsoft\Cryptography\Defaults\Provider.

**To sign data using a key card**

1. Create a new instance of the [CspParameters](https://msdn.microsoft.com/en-US/library/system.security.cryptography.cspparameters(v=vs.90).aspx) class, passing the integer provider type and the provider name to the constructor.
2. Pass the appropriate flags to the [Flags](https://msdn.microsoft.com/en-US/library/system.security.cryptography.cspparameters.flags(v=vs.90).aspx) property of the newly created [CspParameters](https://msdn.microsoft.com/en-US/library/system.security.cryptography.cspparameters(v=vs.90).aspx) object. For example, pass the[UseDefaultKeyContainer](https://msdn.microsoft.com/en-US/library/system.security.cryptography.cspproviderflags(v=vs.90).aspx) flag.
3. Create a new instance of an [AsymmetricAlgorithm](https://msdn.microsoft.com/en-US/library/system.security.cryptography.asymmetricalgorithm(v=vs.90).aspx) class (for example, the [RSACryptoServiceProvider](https://msdn.microsoft.com/en-US/library/system.security.cryptography.rsacryptoserviceprovider(v=vs.90).aspx) class), passing the [CspParameters](https://msdn.microsoft.com/en-US/library/system.security.cryptography.cspparameters(v=vs.90).aspx)object to the constructor.
4. Sign your data using one of the **Sign** methods and verify your data using one of the **Verify** methods.

**To generate a random number using a hardware random number generator**

1. Create a new instance of the [CspParameters](https://msdn.microsoft.com/en-US/library/system.security.cryptography.cspparameters(v=vs.90).aspx) class, passing the integer provider type and the provider name to the constructor.
2. Create a new instance of the [RNGCryptoServiceProvider](https://msdn.microsoft.com/en-US/library/system.security.cryptography.rngcryptoserviceprovider(v=vs.90).aspx), passing the [CspParameters](https://msdn.microsoft.com/en-US/library/system.security.cryptography.cspparameters(v=vs.90).aspx) object to the constructor.
3. Create a random value using the [GetBytes](https://msdn.microsoft.com/en-US/library/wb9c8c67(v=vs.90).aspx) or [GetNonZeroBytes](https://msdn.microsoft.com/en-US/library/system.security.cryptography.rngcryptoserviceprovider.getnonzerobytes(v=vs.90).aspx) method.

[**Example**](javascript:void(0))

The following code example demonstrates how to sign data using a smart card. The code example creates a [CspParameters](https://msdn.microsoft.com/en-US/library/system.security.cryptography.cspparameters(v=vs.90).aspx) object that exposes a smart card, and then initializes an [RSACryptoServiceProvider](https://msdn.microsoft.com/en-US/library/system.security.cryptography.rsacryptoserviceprovider(v=vs.90).aspx) object using the CSP. The code example then signs and verifies some data.

C#

[**C++**](https://msdn.microsoft.com/en-US/library/ms229931(v=vs.90).aspx?cs-save-lang=1&cs-lang=cpp#code-snippet-1)

[**VB**](https://msdn.microsoft.com/en-US/library/ms229931(v=vs.90).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

using System;

using System.Security.Cryptography;

namespace SmartCardSign

{

class SCSign

{

static void Main(string[] args)

{

// To idendify the Smart Card CryptoGraphic Providers on your

// computer, use the Microsoft Registry Editor (Regedit.exe).

// The available Smart Card CryptoGraphic Providers are listed

// in HKEY\_LOCAL\_MACHINE\Software\Microsoft\Cryptography\Defaults\Provider.

// Create a new CspParameters object that identifies a

// Smart Card CryptoGraphic Provider.

// The 1st parameter comes from HKEY\_LOCAL\_MACHINE\Software\Microsoft\Cryptography\Defaults\Provider Types.

// The 2nd parameter comes from HKEY\_LOCAL\_MACHINE\Software\Microsoft\Cryptography\Defaults\Provider.

CspParameters csp = new CspParameters(1, "Schlumberger Cryptographic Service Provider");

csp.Flags = CspProviderFlags.UseDefaultKeyContainer;

// Initialize an RSACryptoServiceProvider object using

// the CspParameters object.

RSACryptoServiceProvider rsa = new RSACryptoServiceProvider(csp);

// Create some data to sign.

byte[] data = new byte[] { 0, 1, 2, 3, 4, 5, 6, 7 };

Console.WriteLine("Data : " + BitConverter.ToString(data));

// Sign the data using the Smart Card CryptoGraphic Provider.

byte[] sig = rsa.SignData(data, "SHA1");

Console.WriteLine("Signature : " + BitConverter.ToString(sig));

// Verify the data using the Smart Card CryptoGraphic Provider.

bool verified = rsa.VerifyData(data, "SHA1", sig);

Console.WriteLine("Verified : " + verified);

}

}

}

[**Compiling the Code**](javascript:void(0))

* Include the [System](https://msdn.microsoft.com/en-US/library/system(v=vs.90).aspx) and [System.Security.Cryptography](https://msdn.microsoft.com/en-US/library/system.security.cryptography(v=vs.90).aspx) namespaces.
* You must have a smart card reader and drivers installed on your computer.
* You must initialize the [CspParameters](https://msdn.microsoft.com/en-US/library/system.security.cryptography.cspparameters(v=vs.90).aspx) object using information specific to your card reader. For more information, see the documentation of your card reader.

#### Walkthrough: Creating a Cryptographic Application

**.NET Framework 3.5, 4, 4.6 and 4.5**

[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635736413806036745

This walkthrough demonstrates how to encrypt and decrypt content. The code examples are designed for a Windows Forms application. This application does not demonstrate real world scenarios, such as using smart cards. Instead, it demonstrates the fundamentals of encryption and decryption.

This walkthrough uses the following guidelines for encryption:

* Use the [RijndaelManaged](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rijndaelmanaged(v=vs.110).aspx) class, a symmetric algorithm, to encrypt and decrypt data by using its automatically generated [Key](https://msdn.microsoft.com/en-us/library/system.security.cryptography.symmetricalgorithm.key(v=vs.110).aspx) and [IV](https://msdn.microsoft.com/en-us/library/system.security.cryptography.symmetricalgorithm.iv(v=vs.110).aspx).
* Use the [RSACryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsacryptoserviceprovider(v=vs.110).aspx), an asymmetric algorithm, to encrypt and decrypt the key to the data encrypted by[RijndaelManaged](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rijndaelmanaged(v=vs.110).aspx). Asymmetric algorithms are best used for smaller amounts of data, such as a key.

|  |
| --- |
| **Note Note** |
| If you want to protect data on your computer instead of exchanging encrypted content with other people, consider using the[ProtectedData](https://msdn.microsoft.com/en-us/library/system.security.cryptography.protecteddata(v=vs.110).aspx) or [ProtectedMemory](https://msdn.microsoft.com/en-us/library/system.security.cryptography.protectedmemory(v=vs.110).aspx) classes. |

The following table summarizes the cryptographic tasks in this topic.

|  |  |
| --- | --- |
| **Task** | **Description** |
| Creating a Windows Forms application | Lists the controls that are required to run the application. |
| Declaring global objects | Declares string path variables, the [CspParameters](https://msdn.microsoft.com/en-us/library/system.security.cryptography.cspparameters(v=vs.110).aspx), and the [RSACryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsacryptoserviceprovider(v=vs.110).aspx) to have global context of the [Form](https://msdn.microsoft.com/en-us/library/system.windows.forms.form(v=vs.110).aspx) class. |
| Creating an asymmetric key | Creates an asymmetric public and private key value pair and assigns it a key container name. |
| Encrypting a file | Displays a dialog box to select a file for encryption and encrypts the file. |
| Decrypting a file | Displays a dialog box to select an encrypted file for decryption and decrypts the file. |
| Getting a private key | Gets the full key pair using the key container name. |
| Exporting a public key | Saves the key to an XML file with only public parameters. |
| Importing a public key | Loads the key from an XML file into the key container. |
| Testing the application | Lists procedures for testing this application. |

##### [Prerequisites](javascript:void(0))

You need the following components to complete this walkthrough:

* References to the [System.IO](https://msdn.microsoft.com/en-us/library/system.io(v=vs.110).aspx) and [System.Security.Cryptography](https://msdn.microsoft.com/en-us/library/system.security.cryptography(v=vs.110).aspx) namespaces.

##### [Creating a Windows Forms Application](javascript:void(0))

Most of the code examples in this walkthrough are designed to be event handlers for button controls. The following table lists the controls required for the sample application and their required names to match the code examples.

|  |  |  |
| --- | --- | --- |
| **Control** | **Name** | **Text property (as needed)** |
| [Button](https://msdn.microsoft.com/en-us/library/system.windows.forms.button(v=vs.110).aspx) | buttonEncryptFile | Encrypt File |
| [Button](https://msdn.microsoft.com/en-us/library/system.windows.forms.button(v=vs.110).aspx) | buttonDecryptFile | Decrypt File |
| [Button](https://msdn.microsoft.com/en-us/library/system.windows.forms.button(v=vs.110).aspx) | buttonCreateAsmKeys | Create Keys |
| [Button](https://msdn.microsoft.com/en-us/library/system.windows.forms.button(v=vs.110).aspx) | buttonExportPublicKey | Export Public Key |
| [Button](https://msdn.microsoft.com/en-us/library/system.windows.forms.button(v=vs.110).aspx) | buttonImportPublicKey | Import Public Key |
| [Button](https://msdn.microsoft.com/en-us/library/system.windows.forms.button(v=vs.110).aspx) | buttonGetPrivateKey | Get Private Key |
| [Label](https://msdn.microsoft.com/en-us/library/system.windows.forms.label(v=vs.110).aspx) | label1 |  |
| [OpenFileDialog](https://msdn.microsoft.com/en-us/library/system.windows.forms.openfiledialog(v=vs.110).aspx) | openFileDialog1 |  |
| [OpenFileDialog](https://msdn.microsoft.com/en-us/library/system.windows.forms.openfiledialog(v=vs.110).aspx) | openFileDialog2 |  |

Double-click the buttons in the Visual Studio designer to create their event handlers.

##### [Declaring Global Objects](javascript:void(0))

Add the following code to the Form's constructor. Edit the string variables for your environment and preferences.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/bb397867(v=vs.110).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

// Declare CspParmeters and RsaCryptoServiceProvider

// objects with global scope of your Form class.

CspParameters cspp = new CspParameters();

RSACryptoServiceProvider rsa;

// Path variables for source, encryption, and

// decryption folders. Must end with a backslash.

const string EncrFolder = @"c:\Encrypt\";

const string DecrFolder = @"c:\Decrypt\";

const string SrcFolder = @"c:\docs\";

// Public key file

const string PubKeyFile = @"c:\encrypt\rsaPublicKey.txt";

// Key container name for

// private/public key value pair.

const string keyName = "Key01";

##### [Creating an Asymmetric Key](javascript:void(0))

This task creates an asymmetric key that encrypts and decrypts the [RijndaelManaged](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rijndaelmanaged(v=vs.110).aspx) key. This key was used to encrypt the content and it displays the key container name on the label control.

Add the following code as the Click event handler for the Create Keys button (buttonCreateAsmKeys\_Click).

C#

[**VB**](https://msdn.microsoft.com/en-us/library/bb397867(v=vs.110).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

private void buttonCreateAsmKeys\_Click(object sender, System.EventArgs e)

{

// Stores a key pair in the key container.

cspp.KeyContainerName = keyName;

rsa = new RSACryptoServiceProvider(cspp);

rsa.PersistKeyInCsp = true;

if (rsa.PublicOnly == true)

label1.Text = "Key: " + cspp.KeyContainerName + " - Public Only";

else

label1.Text = "Key: " + cspp.KeyContainerName + " - Full Key Pair";

}

##### [Encrypting a File](javascript:void(0))

This task involves two methods: the event handler method for the Encrypt Filebutton (buttonEncryptFile\_Click) and theEncryptFile method. The first method displays a dialog box for selecting a file and passes the file name to the second method, which performs the encryption.

The encrypted content, key, and IV are all saved to one [FileStream](https://msdn.microsoft.com/en-us/library/system.io.filestream(v=vs.110).aspx), which is referred to as the encryption package.

The EncryptFile method does the following:

1. Creates a [RijndaelManaged](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rijndaelmanaged(v=vs.110).aspx) symmetric algorithm to encrypt the content.
2. Creates an [RSACryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsacryptoserviceprovider(v=vs.110).aspx) object to encrypt the [RijndaelManaged](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rijndaelmanaged(v=vs.110).aspx) key.
3. Uses a [CryptoStream](https://msdn.microsoft.com/en-us/library/system.security.cryptography.cryptostream(v=vs.110).aspx) object to read and encrypt the [FileStream](https://msdn.microsoft.com/en-us/library/system.io.filestream(v=vs.110).aspx) of the source file, in blocks of bytes, into a destination [FileStream](https://msdn.microsoft.com/en-us/library/system.io.filestream(v=vs.110).aspx)object for the encrypted file.
4. Determines the lengths of the encrypted key and IV, and creates byte arrays of their length values.
5. Writes the Key, IV, and their length values to the encrypted package.

The encryption package uses the following format:

* Key length, bytes 0 - 3
* IV length, bytes 4 - 7
* Encrypted key
* IV
* Cipher text

You can use the lengths of the key and IV to determine the starting points and lengths of all parts of the encryption package, which can then be used to decrypt the file.

Add the following code as the Click event handler for the Encrypt File button (buttonEncryptFile\_Click).

C#

[**VB**](https://msdn.microsoft.com/en-us/library/bb397867(v=vs.110).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-3)

private void buttonEncryptFile\_Click(object sender, System.EventArgs e)

{

if (rsa == null)

MessageBox.Show("Key not set.");

else

{

// Display a dialog box to select a file to encrypt.

openFileDialog1.InitialDirectory = SrcFolder;

if (openFileDialog1.ShowDialog() == DialogResult.OK)

{

string fName = openFileDialog1.FileName;

if (fName != null)

{

FileInfo fInfo = new FileInfo(fName);

// Pass the file name without the path.

string name = fInfo.FullName;

EncryptFile(name);

}

}

}

}

Add the following EncryptFile method to the form.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/bb397867(v=vs.110).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-4)

private void EncryptFile(string inFile)

{

// Create instance of Rijndael for

// symetric encryption of the data.

RijndaelManaged rjndl = new RijndaelManaged();

rjndl.KeySize = 256;

rjndl.BlockSize = 256;

rjndl.Mode = CipherMode.CBC;

ICryptoTransform transform = rjndl.CreateEncryptor();

// Use RSACryptoServiceProvider to

// enrypt the Rijndael key.

// rsa is previously instantiated:

// rsa = new RSACryptoServiceProvider(cspp);

byte[] keyEncrypted = rsa.Encrypt(rjndl.Key, false);

// Create byte arrays to contain

// the length values of the key and IV.

byte[] LenK = new byte[4];

byte[] LenIV = new byte[4];

int lKey = keyEncrypted.Length;

LenK = BitConverter.GetBytes(lKey);

int lIV = rjndl.IV.Length;

LenIV = BitConverter.GetBytes(lIV);

// Write the following to the FileStream

// for the encrypted file (outFs):

// - length of the key

// - length of the IV

// - ecrypted key

// - the IV

// - the encrypted cipher content

int startFileName = inFile.LastIndexOf("\\") + 1;

// Change the file's extension to ".enc"

string outFile = EncrFolder + inFile.Substring(startFileName, inFile.LastIndexOf(".")- startFileName) + ".enc";

using (FileStream outFs = new FileStream(outFile, FileMode.Create))

{

outFs.Write(LenK, 0, 4);

outFs.Write(LenIV, 0, 4);

outFs.Write(keyEncrypted, 0, lKey);

outFs.Write(rjndl.IV, 0, lIV);

// Now write the cipher text using

// a CryptoStream for encrypting.

using (CryptoStream outStreamEncrypted = new CryptoStream(outFs, transform, CryptoStreamMode.Write))

{

// By encrypting a chunk at

// a time, you can save memory

// and accommodate large files.

int count = 0;

int offset = 0;

// blockSizeBytes can be any arbitrary size.

int blockSizeBytes = rjndl.BlockSize / 8;

byte[] data = new byte[blockSizeBytes];

int bytesRead = 0;

using (FileStream inFs = new FileStream(inFile, FileMode.Open))

{

do

{

count = inFs.Read(data, 0, blockSizeBytes);

offset += count;

outStreamEncrypted.Write(data, 0, count);

bytesRead += blockSizeBytes;

}

while (count > 0);

inFs.Close();

}

outStreamEncrypted.FlushFinalBlock();

outStreamEncrypted.Close();

}

outFs.Close();

}

}

##### [Decrypting a File](javascript:void(0))

This task involves two methods, the event handler method for the Decrypt File button (buttonEncryptFile\_Click), and theDecryptFile method. The first method displays a dialog box for selecting a file and passes its file name to the second method, which performs the decryption.

The Decrypt method does the following:

1. Creates a [RijndaelManaged](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rijndaelmanaged(v=vs.110).aspx) symmetric algorithm to decrypt the content.
2. Reads the first eight bytes of the [FileStream](https://msdn.microsoft.com/en-us/library/system.io.filestream(v=vs.110).aspx) of the encrypted package into byte arrays to obtain the lengths of the encrypted key and the IV.
3. Extracts the key and IV from the encryption package into byte arrays.
4. Creates an [RSACryptoServiceProvider](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsacryptoserviceprovider(v=vs.110).aspx) object to decrypt the [RijndaelManaged](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rijndaelmanaged(v=vs.110).aspx) key.
5. Uses a [CryptoStream](https://msdn.microsoft.com/en-us/library/system.security.cryptography.cryptostream(v=vs.110).aspx) object to read and decrypt the cipher text section of the [FileStream](https://msdn.microsoft.com/en-us/library/system.io.filestream(v=vs.110).aspx) encryption package, in blocks of bytes, into the [FileStream](https://msdn.microsoft.com/en-us/library/system.io.filestream(v=vs.110).aspx) object for the decrypted file. When this is finished, the decryption is completed.

Add the following code as the Click event handler for the Decrypt File button.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/bb397867(v=vs.110).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-5)

private void buttonDecryptFile\_Click(object sender, EventArgs e)

{

if (rsa == null)

MessageBox.Show("Key not set.");

else

{

// Display a dialog box to select the encrypted file.

openFileDialog2.InitialDirectory = EncrFolder;

if (openFileDialog2.ShowDialog() == DialogResult.OK)

{

string fName = openFileDialog2.FileName;

if (fName != null)

{

FileInfo fi = new FileInfo(fName);

string name = fi.Name;

DecryptFile(name);

}

}

}

}

Add the following DecryptFile method to the form.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/bb397867(v=vs.110).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-6)

private void DecryptFile(string inFile)

{

// Create instance of Rijndael for

// symetric decryption of the data.

RijndaelManaged rjndl = new RijndaelManaged();

rjndl.KeySize = 256;

rjndl.BlockSize = 256;

rjndl.Mode = CipherMode.CBC;

// Create byte arrays to get the length of

// the encrypted key and IV.

// These values were stored as 4 bytes each

// at the beginning of the encrypted package.

byte[] LenK = new byte[4];

byte[] LenIV = new byte[4];

// Consruct the file name for the decrypted file.

string outFile = DecrFolder + inFile.Substring(0, inFile.LastIndexOf(".")) + ".txt";

// Use FileStream objects to read the encrypted

// file (inFs) and save the decrypted file (outFs).

using (FileStream inFs = new FileStream(EncrFolder + inFile, FileMode.Open))

{

inFs.Seek(0, SeekOrigin.Begin);

inFs.Seek(0, SeekOrigin.Begin);

inFs.Read(LenK, 0, 3);

inFs.Seek(4, SeekOrigin.Begin);

inFs.Read(LenIV, 0, 3);

// Convert the lengths to integer values.

int lenK = BitConverter.ToInt32(LenK, 0);

int lenIV = BitConverter.ToInt32(LenIV, 0);

// Determine the start postition of

// the ciphter text (startC)

// and its length(lenC).

int startC = lenK + lenIV + 8;

int lenC = (int)inFs.Length - startC;

// Create the byte arrays for

// the encrypted Rijndael key,

// the IV, and the cipher text.

byte[] KeyEncrypted = new byte[lenK];

byte[] IV = new byte[lenIV];

// Extract the key and IV

// starting from index 8

// after the length values.

inFs.Seek(8, SeekOrigin.Begin);

inFs.Read(KeyEncrypted, 0, lenK);

inFs.Seek(8 + lenK, SeekOrigin.Begin);

inFs.Read(IV, 0, lenIV);

Directory.CreateDirectory(DecrFolder);

// Use RSACryptoServiceProvider

// to decrypt the Rijndael key.

byte[] KeyDecrypted = rsa.Decrypt(KeyEncrypted, false);

// Decrypt the key.

ICryptoTransform transform = rjndl.CreateDecryptor(KeyDecrypted, IV);

// Decrypt the cipher text from

// from the FileSteam of the encrypted

// file (inFs) into the FileStream

// for the decrypted file (outFs).

using (FileStream outFs = new FileStream(outFile, FileMode.Create))

{

int count = 0;

int offset = 0;

// blockSizeBytes can be any arbitrary size.

int blockSizeBytes = rjndl.BlockSize / 8;

byte[] data = new byte[blockSizeBytes];

// By decrypting a chunk a time,

// you can save memory and

// accommodate large files.

// Start at the beginning

// of the cipher text.

inFs.Seek(startC, SeekOrigin.Begin);

using (CryptoStream outStreamDecrypted = new CryptoStream(outFs, transform, CryptoStreamMode.Write))

{

do

{

count = inFs.Read(data, 0, blockSizeBytes);

offset += count;

outStreamDecrypted.Write(data, 0, count);

}

while (count > 0);

outStreamDecrypted.FlushFinalBlock();

outStreamDecrypted.Close();

}

outFs.Close();

}

inFs.Close();

}

}

##### [Exporting a Public Key](javascript:void(0))

This task saves the key created by the Create Keys button to a file. It exports only the public parameters.

This task simulates the scenario of Alice giving Bob her public key so that he can encrypt files for her. He and others who have that public key will not be able to decrypt them because they do not have the full key pair with private parameters.

Add the following code as the Click event handler for the Export Public Key button (buttonExportPublicKey\_Click).

C#

[**VB**](https://msdn.microsoft.com/en-us/library/bb397867(v=vs.110).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-7)

void buttonExportPublicKey\_Click(object sender, System.EventArgs e)

{

// Save the public key created by the RSA

// to a file. Caution, persisting the

// key to a file is a security risk.

Directory.CreateDirectory(EncrFolder);

StreamWriter sw = new StreamWriter(PubKeyFile, false);

sw.Write(rsa.ToXmlString(false));

sw.Close();

}

##### [Importing a Public Key](javascript:void(0))

This task loads the key with only public parameters, as created by the Export Public Key button, and sets it as the key container name.

This task simulates the scenario of Bob loading Alice's key with only public parameters so he can encrypt files for her.

Add the following code as the Click event handler for the Import Public Key button (buttonImportPublicKey\_Click).

C#

[**VB**](https://msdn.microsoft.com/en-us/library/bb397867(v=vs.110).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-8)

void buttonImportPublicKey\_Click(object sender, System.EventArgs e)

{

StreamReader sr = new StreamReader(PubKeyFile);

cspp.KeyContainerName = keyName;

rsa = new RSACryptoServiceProvider(cspp);

string keytxt = sr.ReadToEnd();

rsa.FromXmlString(keytxt);

rsa.PersistKeyInCsp = true;

if (rsa.PublicOnly == true)

label1.Text = "Key: " + cspp.KeyContainerName + " - Public Only";

else

label1.Text = "Key: " + cspp.KeyContainerName + " - Full Key Pair";

sr.Close();

}

##### [Getting a Private Key](javascript:void(0))

This task sets the key container name to the name of the key created by using the Create Keys button. The key container will contain the full key pair with private parameters.

This task simulates the scenario of Alice using her private key to decrypt files encrypted by Bob.

Add the following code as the Click event handler for the Get Private Key button (buttonGetPrivateKey\_Click).

C#

[**VB**](https://msdn.microsoft.com/en-us/library/bb397867(v=vs.110).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-9)

private void buttonGetPrivateKey\_Click(object sender, EventArgs e)

{

cspp.KeyContainerName = keyName;

rsa = new RSACryptoServiceProvider(cspp);

rsa.PersistKeyInCsp = true;

if (rsa.PublicOnly == true)

label1.Text = "Key: " + cspp.KeyContainerName + " - Public Only";

else

label1.Text = "Key: " + cspp.KeyContainerName + " - Full Key Pair";

}

##### [Testing the Application](javascript:void(0))

After you have built the application, perform the following testing scenarios.

###### To create keys, encrypt, and decrypt

1. Click the Create Keys button. The label displays the key name and shows that it is a full key pair.
2. Click the Export Public Key button. Note that exporting the public key parameters does not change the current key.
3. Click the Encrypt File button and select a file.
4. Click the Decrypt Filebutton and select the file just encrypted.
5. Examine the file just decrypted.
6. Close the application and restart it to test retrieving persisted key containers in the next scenario.

###### To encrypt using the public key

1. Click the Import Public Key button. The label displays the key name and shows that it is public only.
2. Click the Encrypt File button and select a file.
3. Click the Decrypt Filebutton and select the file just encrypted. This will fail because you must have the private key to decrypt.

This scenario demonstrates having only the public key to encrypt a file for another person. Typically that person would give you only the public key and withhold the private key for decryption.

###### To decrypt using the private key

1. Click the Get Private Key button. The label displays the key name and shows whether it is the full key pair.
2. Click the Decrypt File button and select the file just encrypted. This will be successful because you have the full key pair to decrypt.

## Security Policy Management

**.NET Framework 1.1, 2.0, 3.0, 3.5**

Security policy is the configurable set of rules that the common language runtime follows when determining the permissions to grant to code. The runtime examines identifiable characteristics of the code, such as the Web site or zone where the code originates, to determine the access that code can have to resources. During execution, the runtime ensures that code accesses only the resources that it has been granted permission to access.

Security policy defines several code groups and associates each of them with a set of permissions. Code groups categorize code by characteristics such as its publisher, digital signature, the URL from where it originates, and so on. After all evidence is considered, code is placed into code groups and the resulting permission grant is the total set of permissions associated with every code group that the code obtains membership in. Although the default security policy is suitable for most situations, administrators can modify or customize security policy to tailor it to the specific needs of their organizations. The runtime grants permissions to both assemblies and application domains based on security policy.

**In This Section**

[Security Policy Model](https://msdn.microsoft.com/en-us/library/ck90k585(v=vs.71).aspx)

Describes the components of the security policy system.

[Permission Grants](https://msdn.microsoft.com/en-us/library/abt16x18(v=vs.71).aspx)

Describes how the common language runtime grants permission to code.

[Default Security Policy](https://msdn.microsoft.com/en-us/library/03kwzyfc(v=vs.71).aspx)

Describes how security policy is configured by default.

[Administering Security Policy](https://msdn.microsoft.com/en-us/library/wztcyd2e(v=vs.71).aspx)

Describes how administrators can view and modify security policy.

[Internet Explorer Security and Managed Execution](https://msdn.microsoft.com/en-us/library/101853ac(v=vs.71).aspx)

Describes how Microsoft Internet Explorer security settings affect managed execution.

**Related Sections**

[Security Policy Best Practices](https://msdn.microsoft.com/en-us/library/sa4se9bc(v=vs.71).aspx)

Describes techniques that administrators can use to maintain security policy on a machine or in an enterprise.

[Key Security Concepts](https://msdn.microsoft.com/en-us/library/z164t8hs(v=vs.71).aspx)

Introduces fundamental concepts you must understand before using .NET Framework security.

[Permissions](https://msdn.microsoft.com/en-us/library/5ba4k1c5(v=vs.71).aspx)

Describes permission objects and how they are used by the runtime.

[Code Access Security](https://msdn.microsoft.com/en-us/library/930b76w0(v=vs.71).aspx)

Describes .NET Framework code access security in detail and provides instructions for using it in your code.

[Security Tools](https://msdn.microsoft.com/en-us/library/7w3fd0wb(v=vs.71).aspx)

Lists and briefly describes the security tools included in the .NET Framework SDK.

## Security Policy Management

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4**

|  |
| --- |
| **Important noteImportant** |
| In the .NET Framework version 4, the common language runtime (CLR) is moving away from providing security policy for computers. Microsoft is recommending the use of Windows Software Restriction Policies as a replacement for CLR security policy. The information in this topic applies to the .NET Framework version 3.5 and earlier; it does not apply to version 4.0 and later. For more information about this and other changes, see [Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx). |

Security policy is the configurable set of rules that the common language runtime follows when determining the permissions to grant to code. The code is identified by its assembly file, which is either an executable (.exe) or a library (.dll). The runtime examines identifiable characteristics of the assembly, such as the Web site or zone where the code originates, to determine the access that the code can have to resources. Those characteristics are defined as [System.Security.Policy.Evidence](https://msdn.microsoft.com/en-us/library/system.security.policy.evidence(v=vs.100).aspx), which is associated with the assembly. Typical [Evidence](https://msdn.microsoft.com/en-us/library/system.security.policy.evidence(v=vs.100).aspx) for an assembly is [Url](https://msdn.microsoft.com/en-us/library/system.security.policy.url(v=vs.100).aspx), [Zone](https://msdn.microsoft.com/en-us/library/system.security.policy.zone(v=vs.100).aspx), [StrongName](https://msdn.microsoft.com/en-us/library/system.security.policy.strongname(v=vs.100).aspx), and [Hash](https://msdn.microsoft.com/en-us/library/system.security.policy.hash(v=vs.100).aspx). During execution, the runtime uses the [Evidence](https://msdn.microsoft.com/en-us/library/system.security.policy.evidence(v=vs.100).aspx) to ensure that code accesses only the resources that it has been granted permission to access.

Security policy defines several code groups and associates each group with a set of permissions. The security system uses [Evidence](https://msdn.microsoft.com/en-us/library/system.security.policy.evidence(v=vs.100).aspx) to determine the code groups an assembly belongs in. After all evidence is considered, the assembly is associated with one or more code groups, and the resulting permission grant is the total set of permissions associated with all the matching code groups.

By default, assemblies that reside on the computer they are to run on are in the [MyComputer](https://msdn.microsoft.com/en-us/library/system.security.securityzone(v=vs.100).aspx) zone. The [MyComputer](https://msdn.microsoft.com/en-us/library/system.security.securityzone(v=vs.100).aspx) zone evidence places the code in the **My\_Computer\_Zone** code group and grants it full-trust permission. The full-trust permission set bypasses security checks and grants code access to all protected resources. For that reason, it is very important that you install applications on your computer only from sources that you completely trust. In the .NET Framework version 3.5 Service Pack 1 (SP1), full trust is extended to assemblies that you run from an intranet as well as from your computer. Again, you should only run intranet applications that you fully trust. Security policy administrators can decide to restore the policy for intranet applications to the partial-trust status they were previously granted.

Although the default security policy is suitable for most situations, administrators can modify or customize security policy according to the specific needs of their organizations. The runtime grants permissions to both assemblies and application domains based on security policy.

[**In This Section**](javascript:void(0))

[Security Policy Model](https://msdn.microsoft.com/en-us/library/ck90k585(v=vs.100).aspx)

Describes the components of the security policy system.

[Permission Grants](https://msdn.microsoft.com/en-us/library/abt16x18(v=vs.100).aspx)

Describes how the common language runtime grants permission to code.

[Default Security Policy](https://msdn.microsoft.com/en-us/library/03kwzyfc(v=vs.100).aspx)

Describes how security policy is configured by default.

[Administering Security Policy](https://msdn.microsoft.com/en-us/library/wztcyd2e(v=vs.100).aspx)

Describes how administrators can view and modify security policy.

[How to: Enable Internet Explorer Security Settings for Managed Execution](https://msdn.microsoft.com/en-us/library/101853ac(v=vs.100).aspx)

Describes how Microsoft Internet Explorer security settings affect managed execution.

[**Related Sections**](javascript:void(0))

[Security Policy Best Practices](https://msdn.microsoft.com/en-us/library/sa4se9bc(v=vs.100).aspx)

Describes techniques that administrators can use to maintain security policy on a machine or in an enterprise.

[Key Security Concepts](https://msdn.microsoft.com/en-us/library/z164t8hs(v=vs.100).aspx)

Introduces fundamental concepts you must understand before using .NET Framework security.

[Permissions](https://msdn.microsoft.com/en-us/library/5ba4k1c5(v=vs.100).aspx)

Describes permission objects and how they are used by the runtime.

[Code Access Security](https://msdn.microsoft.com/en-us/library/c5tk9z76(v=vs.100).aspx)

Describes .NET Framework code access security in detail and provides instructions for using it in your code.

[Security Tools](https://msdn.microsoft.com/en-us/library/dd233106(v=vs.100).aspx)

Lists and briefly describes the security tools included in the .NET Framework.

### Security Policy Model

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4**

|  |
| --- |
| **Important noteImportant** |
| In the .NET Framework version 4, the common language runtime (CLR) is moving away from providing security policy for computers. Microsoft is recommending the use of Windows Software Restriction Policies as a replacement for CLR security policy. The information in this topic applies to the .NET Framework version 3.5 and earlier; it does not apply to version 4.0 and later. For more information about this and other changes, see [Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx). |

The .NET Framework security policy model comprises the following elements:

* [Security policy levels](https://msdn.microsoft.com/en-us/library/628s5x1x(v=vs.71).aspx): enterprise, machine, user, and sometimes application domain.
* A hierarchy of [code groups](https://msdn.microsoft.com/en-us/library/ka9xc0ek(v=vs.71).aspx) within the enterprise, machine, and user policy levels.
* [Named permission sets](https://msdn.microsoft.com/en-us/library/4652tyx7(v=vs.71).aspx) associated with each code group.
* [Evidence](https://msdn.microsoft.com/en-us/library/7y5x1hcd(v=vs.71).aspx) that provides information about the identity of code.
* [Application domain hosts](https://msdn.microsoft.com/en-us/library/6700e49f(v=vs.71).aspx) that provide evidence about code to the common language runtime.

Each security policy level has its own hierarchy of [code groups](https://msdn.microsoft.com/en-us/library/ka9xc0ek(v=vs.71).aspx) that provides a framework for establishing and configuring security policy. Code groups map evidence to a set of allowed permissions. Often, code groups are associated with a [named permission set](https://msdn.microsoft.com/en-us/library/4652tyx7(v=vs.71).aspx) that specifies the allowable permissions for code in that group. The runtime uses evidence provided by a trusted host or by the loader to determine which code groups the code belongs to and, therefore, which permissions the code is granted.

#### Security Policy Levels

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4**

|  |
| --- |
| **Important noteImportant** |
| In the .NET Framework version 4, the common language runtime (CLR) is moving away from providing security policy for computers. Microsoft is recommending the use of Windows Software Restriction Policies as a replacement for CLR security policy. The information in this topic applies to the .NET Framework version 3.5 and earlier; it does not apply to version 4.0 and later. For more information about this and other changes, see [Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx). |

Four security policy levels are provided by the .NET Framework to compute the permission grant of an assembly or application domain. Each level contains its own hierarchy of code groups and permission sets. The runtime intersects the permission sets granted to an assembly from each level when [computing the allowed permission set](https://msdn.microsoft.com/en-us/library/kb207s5y(v=vs.71).aspx). The resulting grant is the sum of permissions allowed by all participating levels in a policy grant.

The following table describes the four security policy levels provided by .NET Framework security.

|  |  |  |
| --- | --- | --- |
| **Policy type** | **Specified by** | **Applies to** |
| Enterprise policy | Administrator | All managed code in an enterprise setting where an enterprise configuration file is distributed. |
| Machine policy | Administrator | All managed code on the computer. |
| User policy | Administrator or user | Code in all the processes associated with the current operating system user when the common language runtime starts. |
| Application domain policy | Application domain host code | Managed code in the host's application domain. |

The policy levels are a hierarchy, with enterprise policy on top, machine policy below that, user policy below that, and application domain policy on the bottom. The runtime starts at the top of the hierarchy and works its way down when computing permission grants. Lower policy levels cannot increase permissions granted at a higher levels; however, lower policy levels can decrease permissions. By default, user and application domain policies are less restrictive than machine and enterprise policy. The majority of the default policy exists on the machine level. For more information about default security settings, see [Default Security Policy](https://msdn.microsoft.com/en-us/library/03kwzyfc(v=vs.71).aspx).

When granting permissions to assemblies, the runtime considers the requirements of all existing policies (enterprise, machine, user, and application domain), together with the assembly's requested permissions.

When granting permissions to application domains, the runtime uses the enterprise, machine, and user policies.

#### Code Groups

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4**

|  |
| --- |
| **Important noteImportant** |
| In the .NET Framework version 4, the common language runtime (CLR) is moving away from providing security policy for computers. Microsoft is recommending the use of Windows Software Restriction Policies as a replacement for CLR security policy. The information in this topic applies to the .NET Framework version 3.5 and earlier; it does not apply to version 4.0 and later. For more information about this and other changes, see [Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx). |

A code group is a logical grouping of code that has a specified condition for membership. Any code that meets the membership condition is included in the group. Code groups have associated permission sets that are evaluated during a policy grant. Administrators configure security policy by managing code groups and their associated permission sets.

The following table shows the code group membership conditions provided by the .NET Framework. Membership conditions are implemented as classes.

|  |  |
| --- | --- |
| **Membership condition** | **Condition based on** |
| All code  [AllMembershipCondition](https://msdn.microsoft.com/en-us/library/system.security.policy.allmembershipcondition(v=vs.100).aspx) | Represents a membership condition that matches all code. |
| Application directory  [ApplicationDirectoryMembershipCondition](https://msdn.microsoft.com/en-us/library/system.security.policy.applicationdirectorymembershipcondition(v=vs.100).aspx) | The application's installation directory. |
| Cryptographic hash  [HashMembershipCondition](https://msdn.microsoft.com/en-us/library/system.security.policy.hashmembershipcondition(v=vs.100).aspx) | An MD5, SHA1, or other cryptographic hash. |
| Software publisher  [PublisherMembershipCondition](https://msdn.microsoft.com/en-us/library/system.security.policy.publishermembershipcondition(v=vs.100).aspx) | The public key of a valid Authenticode signature. |
| Site membership  [SiteMembershipCondition](https://msdn.microsoft.com/en-us/library/system.security.policy.sitemembershipcondition(v=vs.100).aspx) | The HTTP, HTTPS, and FTP site from which code originates. |
| Strong name  [StrongNameMembershipCondition](https://msdn.microsoft.com/en-us/library/system.security.policy.strongnamemembershipcondition(v=vs.100).aspx) | A cryptographically strong signature. |
| URL  [UrlMembershipCondition](https://msdn.microsoft.com/en-us/library/system.security.policy.urlmembershipcondition(v=vs.100).aspx) | The URL where the code originates, including the final wildcard; for example, http://site/app/\*. |
| Zone  [ZoneMembershipCondition](https://msdn.microsoft.com/en-us/library/system.security.policy.zonemembershipcondition(v=vs.100).aspx) | The zone where the code originates. |

The common language runtime uses identifying characteristics ([evidence](https://msdn.microsoft.com/en-us/library/7y5x1hcd(v=vs.100).aspx)) that describe the code to determine whether a group's membership condition has been met. For example, if the membership condition of the group is "Code from the www.microsoft.com Web site", the runtime examines the evidence to determine whether the code originates from www.microsoft.com.

Each code group is associated with a [named permission set](https://msdn.microsoft.com/en-us/library/4652tyx7(v=vs.100).aspx). Code groups can also have [attributes](https://msdn.microsoft.com/en-us/library/3wxtc9hf(v=vs.100).aspx) that affect how the code group is used to define security policy.

Enterprise, machine, and user policy levels are represented by a hierarchy of code groups. The application domain level cannot be administratively configured , but it does have a hierarchy of code groups that can be programmatically set. The root of each hierarchy is the group containing all code. The all code group has child nodes, and those child nodes have child nodes, and so on. If code is a member of the parent code group, then the code might be a member of one or more of that group's child code groups. If code is not a member of the parent code group, it cannot be a member of any of the code groups that are descended from that parent.

Code groups have optional description and name attributes that you can view using the [.NET Framework Configuration Tool](https://msdn.microsoft.com/en-us/library/2bc0cxhc(v=vs.100).aspx).

#### Named Permission Sets

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4**

|  |
| --- |
| **Important noteImportant** |
| In the .NET Framework version 4, the common language runtime (CLR) is moving away from providing security policy for computers. Microsoft is recommending the use of Windows Software Restriction Policies as a replacement for CLR security policy. The information in this topic applies to the .NET Framework version 3.5 and earlier; it does not apply to version 4.0 and later. For more information about this and other changes, see [Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx). |

A named permission set is a set of permissions that administrators can associate with a code group. A named permission set consists of at least one [permission](https://msdn.microsoft.com/en-us/library/5ba4k1c5(v=vs.100).aspx), and a name and description for the permission set. Administrators can use named permission sets to establish or modify the security policy for code groups. More than one code group can be associated with the same named permission set.

The following table shows the built-in named permission sets provided by the common language runtime.

|  |  |
| --- | --- |
| **Permission set** | **Description** |
| **Nothing** | No permissions (code cannot run). |
| **Execution** | Permission to run (execute), but no permissions to use protected resources. |
| **Internet** | The default policy permission set suitable for content from unknown origin. |
| **LocalIntranet** | The default policy permission set within an enterprise. |
| **Everything** | All standard (built-in) permissions, except permission to skip verification. |
| **FullTrust** | Full access to all resources. |

You cannot modify any of the built in named permission sets. However, it is possible to copy them and modify the copy using the .NET Configuration tool Microsoft Management Console (MMC) snap-in. Administrators can define custom named permission sets, as long as their names are different from the built-in named permission sets. Named permission sets cannot contain [identity permissions](https://msdn.microsoft.com/en-us/library/d3wktt6a(v=vs.100).aspx) because identity permissions are derived from evidence directly (for permission objects that implement [IIdentityPermissionFactory](https://msdn.microsoft.com/en-us/library/system.security.policy.iidentitypermissionfactory(v=vs.100).aspx)) and are therefore not a product of normal policy evaluation.

#### Evidence

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4**

|  |
| --- |
| **Important noteImportant** |
| In the .NET Framework version 4, the common language runtime (CLR) is moving away from providing security policy for computers. Microsoft is recommending the use of Windows Software Restriction Policies as a replacement for CLR security policy. The information in this topic applies to the .NET Framework version 3.5 and earlier; it does not apply to version 4.0 and later. For more information about this and other changes, see [Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx). |

Evidence is the information that the common language runtime uses to make decisions based on security policy. Evidence indicates to the runtime that code has a particular characteristic. Common forms of evidence include digital signatures and the location where code originates, but evidence can also be custom-designed to represent other information that is meaningful to the application. Both assemblies and application domains receive permission grants based on evidence.

The following table shows the common types of evidence that a host can present to the runtime.

|  |  |
| --- | --- |
| **Evidence** | **Description** |
| Application directory | The application's installation directory. |
| Hash | Cryptographic hash such as SHA1. |
| Publisher | Software publisher signature; that is, the Authenticode signer of the code. |
| Site | Site of origin, such as http://www.microsoft.com. |
| Strong name | Cryptographically strong name of the assembly. |
| URL | URL of origin. |
| Zone | Zone of origin, such as Internet Zone. |

In addition to the forms of evidence listed in the table, application- or system-defined evidence can also be presented to the runtime. Trusted[application domain hosts](https://msdn.microsoft.com/en-us/library/6700e49f(v=vs.100).aspx) can present evidence about an assembly or [application domain](https://msdn.microsoft.com/en-us/library/2bh4z9hs(v=vs.100).aspx) to the runtime. The runtime uses this information to evaluate enterprise, machine, and user policy (plus an application domain policy for assemblies, if set by the trusted application domain host) and return the set of permissions to grant to the assembly or application domain. If the trusted application domain host does not have permission to provide evidence, the assembly or application domain receives the permissions that have been granted to the host.

The runtime receives evidence about assemblies either from trusted application domain hosts or directly from the loader. Some evidence, such as where the code originates, usually comes from the trusted application domain host because only the host knows this information. Trusted application domain hosts can override evidence provided from the loader and can provide their own evidence.

Other evidence, such as an assembly's digital signature, is inherent in the code itself and can come from the loader or a trusted application domain host. Typically, the runtime validates each assembly's digital signature when the code is loaded. If the digital signature is valid, the trusted application domain host passes the signature information as evidence to the runtime's policy mechanism. In addition, an assembly or a trusted application domain host can provide custom evidence as a resource that is part of the assembly. Administrators and developers can define custom evidence and extend security policy to recognize and use it.

The runtime's policy mechanism uses the evidence from both the trusted application domain host and the assembly to determine a piece of code's membership in a code group.

#### Application Domain Hosts

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4**

|  |
| --- |
| **Important noteImportant** |
| In the .NET Framework version 4, the common language runtime (CLR) is moving away from providing security policy for computers. Microsoft is recommending the use of Windows Software Restriction Policies as a replacement for CLR security policy. The information in this topic applies to the .NET Framework version 3.5 and earlier; it does not apply to version 4.0 and later. For more information about this and other changes, see [Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx). |

Each .NET Framework application runs in an application domain under the control of a host that creates the application domain and loads assemblies into it. The host has access to information about the code ([evidence](https://msdn.microsoft.com/en-us/library/7y5x1hcd(v=vs.100).aspx)), such as the zone in which the code originates, or the digital signatures of the assemblies in the application domain. A trusted host is a host that has permission to provide the common language runtime with this kind of information. The [System.AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.100).aspx) class provides the application domain functionality used by hosts.

The following table shows the types of application hosts.

|  |  |
| --- | --- |
| **Application domain host** | **Description** |
| Browser host (for example, Microsoft Internet Explorer) | Runs code within the context of a Web site. |
| Custom-designed hosts | Creates domains and loads assemblies into domains, including dynamic assemblies. Can be written in managed or unmanaged code. |
| Server host (for example, ASP.NET) | Runs code that handles requests submitted to a server. |
| Shell host | Launches applications (.exe files) from the shell. |

After creating a new application domain, a host can specify the policy to apply to code within the application domain. This policy is always subject to the enterprise, machine and user policy. A host can reduce the set of permissions that machine and user policy allow but cannot expand it. Policy can be set only once for an application domain. To set application domain policy, the host must be granted the security permission (provided through the [SecurityPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermission(v=vs.100).aspx) class) for controlling domain policy.

After application domain policy is set, all subsequently loaded assemblies are granted permissions under the new policy (enterprise, machine, user, and application domain policy). Previously loaded assemblies get permission grants under the pre-existing policy (enterprise, machine and user policy only). The permissions granted to these assemblies are not reevaluated under the new application domain policy.

A trusted host can provide information (evidence) to the runtime about assemblies that are loaded into the application domain. If a domain host does not have the appropriate **SecurityPermission** for controlling evidence, the runtime uses the security enforced on the host to determine the security to enforce on the assembly.

In some situations, evidence that would normally be provided by a trusted application domain host is actually provided by the loader. Typically, after an application domain is created, the application domain host loads the first (main) assembly into the application domain and calls into that assembly to begin execution. When code in the first assembly references code in another assembly, the loader resolves the reference, loads the appropriate assembly into the application domain, and supplies the evidence about the assembly to the runtime. In this situation, the trusted application domain host that provided the evidence for the original assembly does not provide evidence to subsequently loaded assemblies.

### Permission Grants

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4**

|  |
| --- |
| **Important noteImportant** |
| In the .NET Framework version 4, the common language runtime (CLR) is moving away from providing security policy for computers. Microsoft is recommending the use of Windows Software Restriction Policies as a replacement for CLR security policy. The information in this topic applies to the .NET Framework version 3.5 and earlier; it does not apply to version 4.0 and later. For more information about this and other changes, see [Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx). |

The common language runtime grants permissions to both application domains and assemblies. The permission-granting process can involve one or both of the following steps:

1. [Compute the allowed permission set](https://msdn.microsoft.com/en-us/library/kb207s5y(v=vs.71).aspx).
   1. At load time, the runtime determines the set of permissions that each policy level allows the code to have.
   2. The runtime then intersects the allowed permission sets for each relevant policy level, resulting in one set of allowed permissions for the application domain or assembly.
2. [Determine the granted permissions](https://msdn.microsoft.com/en-us/library/9xyehxdk(v=vs.71).aspx).

The runtime compares the final set of allowed permissions with the permissions that the assembly requests, which results in a set of permissions that is granted to the assembly. This step does not apply to permission grants for application domains.

#### Computing the Allowed Permission Set

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4**

|  |
| --- |
| **Important noteImportant** |
| In the .NET Framework version 4, the common language runtime (CLR) is moving away from providing security policy for computers. Microsoft is recommending the use of Windows Software Restriction Policies as a replacement for CLR security policy. The information in this topic applies to the .NET Framework version 3.5 and earlier; it does not apply to version 4.0 and later. For more information about this and other changes, see [Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx). |

The common language runtime computes the allowed permission set for application domains and assemblies by traversing the code group hierarchies for the relevant policy levels. For application domains, the relevant policy levels are enterprise, machine, and user. For assemblies, the relevant policy levels are enterprise, machine, user, and application domain.

The runtime uses the following process to compute the allowed permission set:

1. For each relevant policy level, the runtime uses identity information provided by evidence to determine which groups the code belongs to. If the code is a member of a group, that group is said to be a match.

The search for a match begins at the top of the code group hierarchy in the all code group. The runtime searches the levels in the hierarchy, including child groups if a match is found in some parent group.

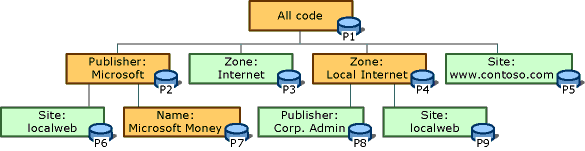
1. When all matches in the hierarchy have been identified, the permissions associated with each matching code group are combined in an additive manner (a union), resulting in the set of permissions allowed by that policy level.

The runtime computes the allowed permission set differently if the **Exclusive** or **LevelFinal** attribute is applied to the code group. For details, see [Code Group Attributes](https://msdn.microsoft.com/en-us/library/3wxtc9hf(v=vs.100).aspx).

1. The runtime then repeats the hierarchy search and intersects the permission sets for each policy level to compute the allowed permission set for the application domain or assembly. The resulting permission set contains only the permissions allowed for all policy levels.

The following illustration shows a code group hierarchy where Microsoft Money is a member of four code groups: All code, Microsoft (the Publisher), Local Internet (the Zone), and Microsoft Money (the Name). The allowed permission set for a given policy level (machine, user, or application domain) is the additive combination (the union) of the named permission sets associated with each of these code groups.

Code group hierarchy



##### Code Group Attributes

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4**

|  |
| --- |
| **Important noteImportant** |
| In the .NET Framework version 4, the common language runtime (CLR) is moving away from providing security policy for computers. Microsoft is recommending the use of Windows Software Restriction Policies as a replacement for CLR security policy. The information in this topic applies to the .NET Framework version 3.5 and earlier; it does not apply to version 4.0 and later. For more information about this and other changes, see [Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx). |

Code groups can have attributes that affect how the common language runtime determines an assembly's allowed permission set. The following table shows the attributes that can be applied to code groups.

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| **Exclusive** | The allowed permission set for the policy level is the permission set associated with the code group that has this attribute. When all policy levels are considered, the runtime never grants the code more permissions than those associated with the**Exclusive** code group. Within a given policy level, code can be a member of no more than one code group that has the**Exclusive** attribute. |
| **LevelFinal** | No policy level, except the application domain level, below the one containing this code group is considered when checking code group membership and granting permissions. Enterprise policy is the highest level of policy, followed by machine policy, user policy, and then application domain policy. For example, if the **LevelFinal** attribute is applied to a code group in enterprise policy and some code matches the membership condition of this code group, then machine and user level policy are not applied to that code. |

A code group can be marked with both the **Exclusive** and **LevelFinal** attributes.

#### Determining the Granted Permissions

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4**

|  |
| --- |
| **Important noteImportant** |
| In the .NET Framework version 4, the common language runtime (CLR) is moving away from providing security policy for computers. Microsoft is recommending the use of Windows Software Restriction Policies as a replacement for CLR security policy. The information in this topic applies to the .NET Framework version 3.5 and earlier; it does not apply to version 4.0 and later. For more information about this and other changes, see [Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx). |

For application domains, the granted permission set is simply the allowed permission set.

For assemblies, the common language runtime considers other factors at assembly-load time to determine the granted permission set. An assembly can contain declarative security requests that specify the permissions the code needs or wants to have. The following table describes the permission sets that code can request.

|  |  |
| --- | --- |
| **Permission set** | **Description** |
| Required | Specifies the minimum set of permissions the code must have to run. |
| Optional | Identifies permissions the code wants to have, in addition to the minimum set. This causes all permissions not identified in the minimum set or optional set to be implicitly refused. |
| Refused | Specifies permissions that should never be granted to the code. |

If all three permission requests are absent, the assembly is simply granted the permission set that policy allows. However, if at least one of the three permission requests is present, the runtime considers the requested permissions using the following process:

1. The runtime [computes the allowed permissions](https://msdn.microsoft.com/en-us/library/kb207s5y(v=vs.100).aspx) for the assembly and insures that the assembly has permission to execute. If permission to execute is not present, the runtime throws a [PolicyException](https://msdn.microsoft.com/en-us/library/system.security.policy.policyexception(v=vs.100).aspx) and the code is not allowed to run.
2. The runtime determines whether the set of required permissions is a subset of the allowed permission set. If not, the runtime throws a**PolicyException** and the code is not allowed to run.
3. The runtime intersects the optional requested permissions with the allowed permission set. If optional permissions are not requested, then the optional PermissionSet is assumed to be [FullTrust](https://msdn.microsoft.com/en-us/library/4652tyx7(v=vs.100).aspx).
4. The runtime unions the result of step 3 with the minimum requested permissions.
5. Finally, the runtime subtracts any permissions that are refused from the result of step 4.

### Default Security Policy

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4**

|  |
| --- |
| **Important noteImportant** |
| In the .NET Framework version 4, the common language runtime (CLR) is moving away from providing security policy for computers. Microsoft is recommending the use of Windows Software Restriction Policies as a replacement for CLR security policy. The information in this topic applies to the .NET Framework version 3.5 and earlier; it does not apply to version 4.0 and later. For more information about this and other changes, see [Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx). |

Machine policy is set by default to the values shown in the following tables. Because the policy levels are intersected when determining the allowed permission set, the machine policy settings actually determine the default security policy. Note that these tables represent the named permission sets and permissions used by the default policy settings and not all the named permission sets and permissions available to customize security policy.

The following table shows the default code groups for machine policy and the named permissions sets they receive by default. For example, code that originates from the local computer is assigned to the My Computer Zone and receives full trust by default.

|  |  |
| --- | --- |
| **Code group** | **Named permission set received by default** |
| * **My Computer Zone** (code from the local computer) * **Microsoft Strong Name** (code signed with the Microsoft Strong Name) * **ECMA Strong Name** (code signed with the ECMA strong name) | Full Trust |
| * **Local Intranet Zone** (Code from a local network) | Local Intranet |
| * **Internet Zone** (Code from the internet) * **Trusted Zone** (Code from trusted sites in Internet Explorer) | Internet |
| * **All Code** (All managed code) * **Restricted Zone** (Code from restricted sites) | Nothing |

**Note**   In .NET Framework 1.0 Service Pack 1 and Service Pack 2, the **Internet Zone** Code Group receives the Nothing named permission set. In all other releases of the .NET Framework, the **Internet Zone** code group receives the Internet named permissions set, as described in the preceding table.

Although the All Code group receives the **Nothing** permission set by default, this does not mean that none of the code has permissions, because matching code groups are combined (called a union) to calculate the allowed permission set.

The following table shows the individual permissions that constitute the default permission sets. The column on the left lists individual permission objects, while the columns on the right represent the configuration of those objects in the permission sets. For example, code that originates from the Local Intranet Zone receives the Local Intranet Permission Set. This table shows that the Local Intranet Permission Set consists of unrestricted **DNSPermission**, unrestricted **FileDialogPermission**, and so on. Note that code from the Internet Zone does not receive the Internet Permission Set by default; instead it receives the Nothing Permission Set.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Permission** | **Full Trust Permission Set**  **(Unrestricted access to all permissions including those not listed)** | **Nothing Permission Set**  **(No permissions, no right to execute)** | **Local Intranet Permission Set** | **Internet Permission Set** |
| [DnsPermission](https://msdn.microsoft.com/en-us/library/system.net.dnspermission(v=vs.71).aspx) | Unrestricted | No Access | [Unrestricted](https://msdn.microsoft.com/en-us/library/system.security.permissions.permissionstate(v=vs.71).aspx) | No Access |
| [EnvironmentPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.environmentpermission(v=vs.71).aspx) | Unrestricted | No Access | Read Username(environmental variable) | No Access |
| [EventLogPermission](https://msdn.microsoft.com/en-us/library/system.diagnostics.eventlogpermission(v=vs.71).aspx) | Unrestricted | No Access | [Instrument](https://msdn.microsoft.com/en-us/library/system.diagnostics.eventlogpermissionaccess(v=vs.71).aspx) | No Access |
| [FileDialogPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.filedialogpermission(v=vs.71).aspx) | Unrestricted | No Access | [Unrestricted](https://msdn.microsoft.com/en-us/library/system.security.permissions.permissionstate(v=vs.71).aspx) | [Open](https://msdn.microsoft.com/en-us/library/system.security.permissions.filedialogpermissionaccess(v=vs.71).aspx) |
| [IsolatedStoragePermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.isolatedstoragepermission(v=vs.71).aspx) | Unrestricted | No Access | [AssemblyIsolationByUser](https://msdn.microsoft.com/en-us/library/system.security.permissions.isolatedstoragecontainment(v=vs.71).aspx)  Disk Quota of 9223372036854775807 | [DomainIsolationByUser](https://msdn.microsoft.com/en-us/library/system.security.permissions.isolatedstoragecontainment(v=vs.71).aspx)  Disk Quota of 10240 |
| [PrintingPermission](https://msdn.microsoft.com/en-us/library/system.drawing.printing.printingpermission(v=vs.71).aspx) | Unrestricted | No Access | [DefaultPrinting](https://msdn.microsoft.com/en-us/library/system.drawing.printing.printingpermissionlevel(v=vs.71).aspx) | [SafePriniting](https://msdn.microsoft.com/en-us/library/system.drawing.printing.printingpermissionlevel(v=vs.71).aspx) |
| [ReflectionPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.reflectionpermission(v=vs.71).aspx) | Unrestricted | No Access | [ReflectionEmit](https://msdn.microsoft.com/en-us/library/system.security.permissions.reflectionpermissionflag(v=vs.71).aspx) | No Access |
| [SecurityPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermission(v=vs.71).aspx) | Unrestricted | No Access | [Execution](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag(v=vs.71).aspx)  [Assertion](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag(v=vs.71).aspx) | [Execution](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag(v=vs.71).aspx) |
| [UIPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.uipermission(v=vs.71).aspx) | Unrestricted | No Access | [Unrestricted](https://msdn.microsoft.com/en-us/library/system.security.permissions.permissionstate(v=vs.71).aspx) | [SafeTopLevelWindows](https://msdn.microsoft.com/en-us/library/system.security.permissions.uipermissionwindow(v=vs.71).aspx)  [OwnClipboard](https://msdn.microsoft.com/en-us/library/y173aaya(v=vs.71).aspx) |

Although the default security policy is suitable for many situations, administrators can modify or customize security policy to tailor it to the specific needs of their organizations. For details, see [Administering Security Policy](https://msdn.microsoft.com/en-us/library/wztcyd2e(v=vs.71).aspx).

Note that code is subject to the additional restrictions associated with the inherent **LinkDemand** for full trust automatically performed by assemblies placed in the global assembly cache. Unless the [AllowPartiallyTrustedCallersAttribute](https://msdn.microsoft.com/en-us/library/system.security.allowpartiallytrustedcallersattribute(v=vs.71).aspx) is applied to an assembly in the global assembly cache, all code that does not receive the Full Trust Permission Set will generate a [SecurityException](https://msdn.microsoft.com/en-us/library/system.security.securityexception(v=vs.71).aspx) when it attempts to link to the assembly. For more information and a list of assemblies that have the **AllowPartiallyTrustedCallersAttribute,** see [Using Libraries from Partially Trusted Code](https://msdn.microsoft.com/en-us/library/8skskf63(v=vs.71).aspx).

### Administering Security Policy

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4**

|  |
| --- |
| **Important noteImportant** |
| In the .NET Framework version 4, the common language runtime (CLR) is moving away from providing security policy for computers. Microsoft is recommending the use of Windows Software Restriction Policies as a replacement for CLR security policy. The information in this topic applies to the .NET Framework version 3.5 and earlier; it does not apply to version 4.0 and later. For more information about this and other changes, see [Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx). |

Administrators can configure security policy so that individual sites and publishers have more or fewer permissions than default policy allows. For example, an administrator can specify that all code downloaded from the Web site of a trusted business partner has the set of all permissions. The same administrator might specify that all other code from the Internet be given a more restricted set of permissions, such as limited access to isolated storage and to the use of safer user interface functionality.

To view or modify security policy, you must be granted the administrative access [SecurityPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermission(v=vs.100).aspx). Understanding the common language runtime's [security policy model](https://msdn.microsoft.com/en-us/library/ck90k585(v=vs.100).aspx) will help you administer security policy effectively.

You can use the [.NET Framework Configuration tool](https://msdn.microsoft.com/en-us/library/2bc0cxhc(v=vs.100).aspx) or the [Code Access Security Policy tool](https://msdn.microsoft.com/en-us/library/cb6t8dtz(v=vs.100).aspx) to administer security policy for the enterprise, machine, or user levels. These tools support the following tasks:

* Viewing policy, code groups, or permission sets.
* Creating, modifying, and removing named permission sets.
* Adding, modifying, and deleting code groups.
* Assigning permissions and attributes to code groups.
* Analyzing security settings on assemblies.
* Undoing policy changes.

### Internet Explorer Security and Managed Execution

**.NET Framework 1.1**

**Newer title**

### How to: Enable Internet Explorer Security Settings for Managed Execution

**.NET Framework 2.0, 3.0, 3.5, 4**

Because Microsoft Internet Explorer can act as a host for managed controls and components, its security settings affect managed execution. Your managed control might not run in Internet Explorer unless ActiveX controls and scripting are enabled.

**To enable Internet Explorer security settings**

1. In Internet Explorer, click **Internet Options** on the **Tools** menu.
2. Click the **Security** tab. You can access the settings for the Internet Zone, Local Intranet Zone, Trusted Sites, and Restricted Sites from this tab.
3. Choose the zone in which the managed control originates and click the **Custom Level** button. **Run ActiveX controls and plug-ins** and**Script ActiveX controls marked safe for scripting** must both be enabled for managed ActiveX controls to execute.

## Security Policy Best Practices

**.NET Framework 1.1**

[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635736413806036745

The .NET Framework provides a code access security model that allows administrators to modify security policy to meet their individual needs. While code access security generally increases the reliability and security of applications, improperly administering code access security policy can potentially create security weaknesses. This section explains basic administration concepts and describes some of the best practices to use when administering code access security policy.

**In This Section**

[Security Policy Administration Overview](https://msdn.microsoft.com/en-us/library/6x640z5e(v=vs.71).aspx)

Provides an overview of basic administration concepts related to code access security.

[General Security Policy Administration](https://msdn.microsoft.com/en-us/library/ed5htz45(v=vs.71).aspx)

Describes security practices that apply to all levels of policy.

[Enterprise Policy Administration](https://msdn.microsoft.com/en-us/library/9604xfcc(v=vs.71).aspx)

Describes security practices that apply to the enterprise policy level.

[Machine Policy Administration](https://msdn.microsoft.com/en-us/library/02a0bf51(v=vs.71).aspx)

Describes security practices that apply to the machine policy level.

[User Policy Administration](https://msdn.microsoft.com/en-us/library/wde04t0a(v=vs.71).aspx)

Describes security practices that apply to the user policy level.

### Security Policy Administration Overview

**.NET Framework 1.1, 2.0, 3.0 ,3.5, 4**

|  |
| --- |
| **Important noteImportant** |
| In the .NET Framework version 4, the common language runtime (CLR) is moving away from providing security policy for computers. Microsoft is recommending the use of Windows Software Restriction Policies as a replacement for CLR security policy. The information in this topic applies to the .NET Framework version 3.5 and earlier; it does not apply to version 4.0 and later. For more information about this and other changes, see [Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx). |

The .NET Framework security system is governed by a configurable set of rules called security policy. This policy allows the end user or administrator to adjust the settings that determine which resources code is allowed to access and ultimately decide which code is allowed to run at all.

For example, suppose you are an administrator in an enterprise setting and you do not trust the software that originates from a particular company. Perhaps that company produces software that employees find entertaining, but causes increased network traffic or causes workstations to become unstable. You can set an enterprise level security policy that restricts the access that software with a particular cryptographic strong name (a unique identifier of a program) has to your computer resources. You can also set a policy that prevents this publisher's software from running at all.

This topic provides an overview of security policy administration. For additional information, see [Security Policy Management](https://msdn.microsoft.com/en-us/library/c1k0eed6(v=vs.71).aspx).

**Evidence, Code Groups, and Permission Sets**

Code that targets the common language runtime is deployed in units called assemblies. At load time, the runtime examines each assembly for evidence, which is identifying information about the assembly (such as the digital signature of the code's author and the location where the code originates). Based on the evidence, the common language runtime security manager maps the assembly to a code group based on security policy. Code groups are defined to test for specific forms of evidence and have permission sets associated with them. Assemblies that belong to a code group receive the permissions defined by the associated permission sets. For more information about evidence, code groups, and permission sets, see [Security Policy Model](https://msdn.microsoft.com/en-us/library/ck90k585(v=vs.71).aspx).

**Permissions**

Permissions are simply objects that represent the right to access a protected resource. Permissions are configurable and a single permission object can assume several forms. For example, the **FileIOPermission** represents the right to access, create, read, write, or modify files on the local hard disk. To be meaningful, a permission must contain specific information about the type of access it represents. You might configure a **FileIOPermission** to represent the right to read one particular file, read and write to one particular file, or read the files in an entire directory. The rights that a permission represent and that assemblies receive are fully configurable by the computer's administrator. While applications can construct and configure permission objects like any other object, only security policy can grant a permission to an application. Administrators ultimately control the permission grant. For a list of common permissions, see [Code Access Permissions](https://msdn.microsoft.com/en-us/library/h846e9b3(v=vs.71).aspx).

**Security Policy Levels**

There are four levels of security policy defined by the security model, which correspond to the different administration and hosting scenarios. The following table describes each level. The enterprise policy level is the highest level and the application domain level is the lowest.

|  |  |
| --- | --- |
| **Policy level** | **Description** |
| Enterprise policy | Defined by enterprise administrators who set policy for enterprise domains. |
| Machine policy | Defined by machine administrators who set policy for one computer. |
| User policy | Defined by users who set policy for a single logon account. |
| Application domain policy | Defined by the runtime host (any application that hosts the common language runtime) for setting load-time policy. This level cannot be administered. |

Each policy level consists of a hierarchy of code groups. The administrators of each policy level can create their own code groups and associated permission sets. At load time, the code access security system examines all policy levels and the resulting permission grant is the intersection of all allowed permissions in each level. Administrators of a lower policy level cannot loosen a policy decision made on a higher level, but they can tighten policy as much as they want. The default security policy resides on the machine policy level.

The default security settings are as follows:

* User and enterprise levels are set to unrestricted.
* The machine level contains the specific policy settings and restrictions.
* The settings defined by all three levels constitute default settings.

Note that unrestricted user and enterprise levels do not result in unrestricted permissions being granted to an assembly. Because the machine level defines several restrictions, and all three levels are considered as a whole, the resulting permission grant is not unrestricted permission. For more information, see [Security Policy Model](https://msdn.microsoft.com/en-us/library/ck90k585(v=vs.71).aspx).

**Mapping Code Groups to Permission Sets**

You manage policy by mapping code groups to permission sets on the policy level or levels that you administer.

Code groups contain a membership condition, a permission set association, and code group attributes. The evidence that an assembly presents to the runtime is compared to the membership condition that you specify for a code group. If an assembly provides evidence that matches the membership condition, it is allowed entrance to the code group. Administrators identify and categorize assemblies into code groups by specifying membership conditions, and assigning permission sets to those code groups. Optionally, code group attributes can be used to specify that no policy level below the current level should be considered or that no code group except the current one should be considered when assigning permission.

The following types of built-in evidence can be used as membership conditions:

* The application's installation directory
* The cryptographic hash of the assembly
* The digital signature of the assembly publisher
* The site from which the assembly originates
* The cryptographic strong name of the assembly
* The URL from which the assembly originates
* The zone from which the assembly originates

You can reduce or increase the permissions to assemblies based on any combination of these membership conditions. Because custom membership conditions can be created, the previous list does not represent every possibility. For more information, see [Evidence](https://msdn.microsoft.com/en-us/library/7y5x1hcd(v=vs.71).aspx).

### General Security Policy Administration

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4**

|  |
| --- |
| **Important noteImportant** |
| In the .NET Framework version 4, the common language runtime (CLR) is moving away from providing security policy for computers. Microsoft is recommending the use of Windows Software Restriction Policies as a replacement for CLR security policy. The information in this topic applies to the .NET Framework version 3.5 and earlier; it does not apply to version 4.0 and later. For more information about this and other changes, see [Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx). |

This section provides a set of guidelines that you can use to help administrate policy for your machine or enterprise. This information is meant to help you decide when to perform administrative tasks and what consequences these decisions have. For information on using tools to perform specific tasks see [Security Policy Configuration](https://msdn.microsoft.com/en-us/library/7c9c2y1w(v=vs.100).aspx).

[**In This Section**](javascript:void(0))

[Determining When to Modify Security Policy](https://msdn.microsoft.com/en-us/library/xky659fc(v=vs.100).aspx)

Describes the factors that would influence your decision to modify the default security policy.

[Administration Tools](https://msdn.microsoft.com/en-us/library/w291dy3z(v=vs.100).aspx)

Describes the tools you can use to customize security policy.

[Administration with Code Group Attributes](https://msdn.microsoft.com/en-us/library/wad20y12(v=vs.100).aspx)

Describes how to limit the policy grants to specific policy levels or code groups by using code group attributes.

[Custom Permissions](https://msdn.microsoft.com/en-us/library/xt8xsee5(v=vs.100).aspx)

Describes the basic steps you must perform in order to incorporate custom permissions into your security policy.

[Administration Tips](https://msdn.microsoft.com/en-us/library/ezyxch3z(v=vs.100).aspx)

Describes some basic guidelines to use when administering security policy.

[Deploying Security Policy](https://msdn.microsoft.com/en-us/library/13wcxx6y(v=vs.100).aspx)

Describes the techniques that you can use to deploy security policy changes to multiple machines in an enterprise setting.

#### Determining When to Modify Security Policy

**.NET Framework 4 --**

|  |
| --- |
| **Important noteImportant** |
| In the .NET Framework version 4, the common language runtime (CLR) is moving away from providing security policy for computers. Microsoft is recommending the use of Windows Software Restriction Policies as a replacement for CLR security policy. The information in this topic applies to the .NET Framework version 3.5 and earlier; it does not apply to the .NET Framework 4 and later versions. For more information about this and other changes, see [Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx). |

You do not necessarily need to modify the default security settings. For many situations, the default security settings provide an adequate level of protection. Code that does not originate from the local computer (and is therefore likely to be less trustworthy) receives restricted access to protected resources under the default security policy. Code originating from the Internet and local intranet is restricted in the following ways:

* Code that originates from the Internet or local intranet does not have permission to read or write to a local drive.
* Code that originates from the Internet or local intranet does not have permission to read or write to the system registry.
* Code that originates from the Internet or local intranet can communicate with the Web site of origin.
* Code that originates from the local intranet has unrestricted access to UI elements, but code originating from the Internet only has access to sub windows and the Clipboard.

The default security policy is adequate for most, but not all, situations. You should consider modifying security policy when:

* You want to trust an application that requires more permission than the zone from which it originates receives by default.
* You use applications from a particular publisher that you completely trust and want these applications to have access to specific resources no matter where they are executed.
* You want applications on the local computer to have less than full trust. For example, you are an enterprise administrator and you want to prevent users from installing and running untrusted applications.

If you decide to edit policy, you must make sure that you do not decrease permission to applications to the point that they will not function properly.

#### Administration Tools

**.NET Framework 4 --**

[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635736413806036745

|  |
| --- |
| **Important noteImportant** |
| In the .NET Framework version 4, the common language runtime (CLR) is moving away from providing security policy for computers. Microsoft is recommending the use of Windows Software Restriction Policies as a replacement for CLR security policy. The information in this topic applies to the .NET Framework version 3.5 and earlier; it does not apply to version 4.0 and later. For more information about this and other changes, see [Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx). |

The recommended way to configure security policy is to use the [.NET Framework Configuration tool (Mscorcfg.msc)](https://msdn.microsoft.com/en-us/library/2bc0cxhc(v=vs.100).aspx). This tool provides wizards to help you adjust your user, machine, and enterprise security settings. The following table describes these wizards.

|  |  |
| --- | --- |
| **Wizard** | **Description** |
| Trust an application | Use this wizard to identify an application by publisher or strong name information and increase the application's level of trust. |
| Adjust security settings | Use this wizard to increase or decrease permissions to assemblies originating from one of the following zones: My Computer, Local intranet, Internet, Trusted Sites, and Untrusted Sites. |
| Create a deployment package | Use this wizard to create a Windows Installer package to deploy security policy across an enterprise. |

For more information on using the wizards, see the [.NET Framework Configuration tool (Mscorcfg.msc)](https://msdn.microsoft.com/en-us/library/2bc0cxhc(v=vs.100).aspx).

If the wizards do not provide the functionality you require to administer security policy, you can edit the permission sets and code groups directly by using either the .NET Framework Configuration toolor the [Code Access Security Policy tool (Caspol.exe)](https://msdn.microsoft.com/en-us/library/cb6t8dtz(v=vs.100).aspx). Caspol.exe is a command-line tool provided mainly for scripting security administration. For information on performing specific tasks using these tools see [Security Policy Configuration](https://msdn.microsoft.com/en-us/library/7c9c2y1w(v=vs.100).aspx).

#### Administration with Code Group Attributes

**.NET Framework 4**

[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635736413806036745

|  |
| --- |
| **Important noteImportant** |
| In the .NET Framework version 4, the common language runtime (CLR) is moving away from providing security policy for computers. Microsoft is recommending the use of Windows Software Restriction Policies as a replacement for CLR security policy. The information in this topic applies to the .NET Framework version 3.5 and earlier; it does not apply to version 4.0 and later. For more information about this and other changes, see [Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx). |

Suppose you are an enterprise administrator who is responsible for administering security policy for a number of workstations. In the typical enterprise domain, the network administrator has administrative privileges on every server and every client. However, it is not uncommon for individual users to have administrative privileges on a single workstation. As a result, the network administrator has administrative privileges on the enterprise policy level and the workstation administrator has administrative privileges on the machine policy level. In this situation, the network administrator seems to have more control over policy because enterprise policy is evaluated first and machine policy is not allowed to loosen security decisions made by the enterprise level administrator. However, the machine level administrator can still tighten security, potentially breaking trusted applications that would otherwise have been allowed to run. For this reason, higher policy levels can choose to exclude lower-level policy decisions from being evaluated.

You can do this by applying the **LevelFinal** or **Exclusive** attribute to a code group by using one of the security policy tools.

[**Level Final Attribute**](javascript:void(0))

When applied to a code group, the **LevelFinal** attribute excludes any policy level from being evaluated below the current level. For example, if you apply the **LevelFinal** attribute to the local intranet code group at the enterprise level, any code group at the machine level will not be evaluated even if a machine level administrator has made changes. Applying the **LevelFinal** attribute guarantees that an assembly associated with a code group marked with this attribute will never receive fewer permissions because of decisions made by a lower policy level administrator. For information on setting the **LevelFinal** attribute on a default or custom code group, see the [.NET Framework Configuration tool (Mscorcfg.msc)](https://msdn.microsoft.com/en-us/library/2bc0cxhc(v=vs.100).aspx) or the [Code Access Security Policy tool (Caspol.exe)](https://msdn.microsoft.com/en-us/library/cb6t8dtz(v=vs.100).aspx).

[**Exclusive Attribute**](javascript:void(0))

When applied to a code group, the **Exclusive** attribute prevents other code groups in the same policy level from being considered when the runtime computes permissions for assemblies that are in the exclusive code group. Policy levels above and below the current level are still evaluated, though. This attribute allows one specific code group to make the sole decision for the current policy level regarding what permissions are granted to assemblies that match that group. This is useful when you want to grant a specific set of permissions to specific assemblies, without allowing permissions from other code group matches on the same policy level.

Note that an assembly is not allowed to execute if it belongs to more than one code group marked as exclusive. Therefore, use the**Exclusive** attribute sparingly when administering custom security policy. For information on setting the **Exclusive** attribute on a built in or custom code group, see the [.NET Framework Configuration tool (Mscorcfg.msc)](https://msdn.microsoft.com/en-us/library/2bc0cxhc(v=vs.100).aspx) or the [Code Access Security Policy tool (Caspol.exe)](https://msdn.microsoft.com/en-us/library/cb6t8dtz(v=vs.100).aspx).

#### Custom Permissions

**.NET Framework 1.1**

**Newer title**

#### How to: Add Custom Permissions to Security Policy

**.NET Framework 2.0, 3.0, 3.5, 4**

|  |
| --- |
| **Important noteImportant** |
| In the .NET Framework version 4, the common language runtime (CLR) is moving away from providing security policy for computers. Microsoft is recommending the use of Windows Software Restriction Policies as a replacement for CLR security policy. The information in this topic applies to the .NET Framework version 3.5 and earlier; it does not apply to version 4.0 and later. For more information about this and other changes, see [Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx). |

A custom permission is one that is provided with an application or library, rather than being provided by the .NET Framework. To be meaningful, a custom permission must be added to the security policy on the computer (or computers, in the case of a network) where the application using the permission runs.

You must describe the specifics of the custom permission you want to add to the .NET Framework security system when you add it your policy. A single permission can assume several forms that represent the right to access specific resources. For example, the custom permission might have the ability to provide restricted or unrestricted access to a resource. As an administrator, you have the ability to decide which type of access (restricted or unrestricted) the permission will represent. Therefore, you must describe the configuration that you want the permission to have to the .NET Framework security system. You do this by creating an XML representation of the configuration you want the custom permission to have and importing the XML into your security policy. (The .NET Framework security system uses XML to serialize permissions.) The publisher of the permission must either provide a tool to create the XML representation or provide instructions for creating the XML file. Usually, the XML representations of the permissions that make up your security policy are stored in the policy configuration files.

**To add a custom permission to security policy**

1. Add the assembly implementing the custom permission to the global assembly cache and to the list of fully trusted assemblies.
2. Create an XML file that represents the type of permission you want to create and assign to assemblies.
3. Add the XML representation of the permission to the security policy by using either the [.NET Framework Configuration tool (Mscorcfg.msc)](https://msdn.microsoft.com/en-us/library/2bc0cxhc(v=vs.100).aspx) or the [Code Access Security Policy tool (Caspol.exe)](https://msdn.microsoft.com/en-us/library/cb6t8dtz(v=vs.100).aspx) .

For more information on adding custom permissions to your security policy, see [Updating Security Policy](https://msdn.microsoft.com/en-us/library/t0385k7y(v=vs.100).aspx).

#### Administration Tips

**.NET Framework 4**

[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635736413806036745

|  |
| --- |
| **Important noteImportant** |
| In the .NET Framework version 4, the common language runtime (CLR) is moving away from providing security policy for computers. Microsoft is recommending the use of Windows Software Restriction Policies as a replacement for CLR security policy. The information in this topic applies to the .NET Framework version 3.5 and earlier; it does not apply to version 4.0 and later. For more information about this and other changes, see [Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx). |

The practices described in this section are applicable to every administration scenario. You should keep them in mind as you set up and administer your security policy.

[**Long-term Strategies for Policy Administration**](javascript:void(0))

Define applicable categories of trust that you can use to administrate policy. Define several code groups to discriminate code that is likely to be run in your environment and define the permissions that each group should receive. Craft policy accordingly and deploy. You should do establish your overall policy when you initially set up your environment, rather than piece-by-piece as you need to run various applications.

[**Administration Level**](javascript:void(0))

The policy level that you choose to administer is determined by the scope that you want to affect. Always administer security policy on the lowest policy level that impacts the fewest users and still satisfies your administration scenario. For example:

* If you are administering a policy that affects every workstation and user in your enterprise, make the policy additions on the enterprise level. Never make an addition to the enterprise level of one computer that is not meant to affect every computer in your enterprise.
* If you are administering a policy that affects all users on a particular computer, make the policy additions on the machine level.
* If you are administering a policy for a particular user or group of users, make the policy additions on the user level.

[**File System**](javascript:void(0))

Use the NTFS file system whenever possible to store the security policy files. NTFS helps provide file protection based on users and groups, and only allows users with administrative privileges for a particular level to edit security configuration files. Systems that do not use the NTFS file system create security weaknesses by allowing unauthorized users to modify security policy.

#### Deploying Security Policy

**.NET Framework 4**

[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635736413806036745

|  |
| --- |
| **Important noteImportant** |
| In the .NET Framework version 4, the common language runtime (CLR) is moving away from providing security policy for computers. Microsoft is recommending the use of Windows Software Restriction Policies as a replacement for CLR security policy. The information in this topic applies to the .NET Framework version 3.5 and earlier; it does not apply to version 4.0 and later. For more information about this and other changes, see [Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx). |

Security policy can be easily deployed in a Windows Installer (.msi) file. An .msi file is a self-contained installation package that can be deployed, installed, and uninstalled in a number of ways. For example, you can deploy an .msi file in any of the following ways:

* Running the .msi file on the computer where you want to deploy the policy, either from the local disk or from a share.
* Using Group Policy on Microsoft Windows servers.
* Using Microsoft Systems Management Server (SMS).

[**Creating Windows Installer Files**](javascript:void(0))

The [Mscorcfg.msc (.NET Framework Configuration Tool)](https://msdn.microsoft.com/en-us/library/2bc0cxhc(v=vs.100).aspx) provides a wizard for creating Windows Installer files. The wizard can create an Installer file that corresponds to one of the three configurable policy levels, but not all of them concurrently. If you are administering security policy for all three configurable levels, you must create three different Windows Installer files and deploy them individually.

The wizard creates the Installer file using the current policy settings of the computer where the wizard executes. For example, to create a user policy for deployment to a group of users, you configure the user policy on your current computer, create the Installer file with the wizard, then return the user policy of the current computer to its original state.

To create a Windows Installer file:

1. Run the .NET Framework Configuration tool (Mscorcfg.msc).
   * In the .NET Framework versions 1.0 and 1.1, type the following at the command prompt:**%Systemroot%\Microsoft.NET\Framework\***versionNumber***\Mscorcfg.msc**.
   * In the .NET Framework 2.0 and later, start the [Visual Studio and Windows SDK Command Prompts](https://msdn.microsoft.com/en-us/library/ms229859(v=vs.100).aspx) and type **mscorcfg.msc**. The SDK Command Prompt, which automatically sets the SDK environment variables that enable you to easily use .NET Framework tools, is included in the [.NET Framework version 2.0 Software Development Kit (SDK)](http://go.microsoft.com/fwlink/?LinkId=115253). Subsequent releases of the .NET Framework are built incrementally on the .NET Framework version 2.0. Consequently, the SDK Command Prompt from the .NET Framework 2.0 SDK is the latest stand-alone SDK Command Prompt available. Alternatively, you may use the Visual Studio command prompts that are provided with Visual Studio 2005 and later versions.
2. In the left pane, right-click the **Runtime Security Policy** node.
3. From the menu, choose **Create Deployment Package**.
4. Follow the Deployment Package wizard instructions to create the .msi file.

When you deploy policy by using an installer file that is created by the [Mscorcfg.msc (.NET Framework Configuration Tool)](https://msdn.microsoft.com/en-us/library/2bc0cxhc(v=vs.100).aspx), the following applies:

* Policy installation affects only the version of the runtime that you targeted when you created the installation file. For example, if you use the .NET Framework Configuration tool version 2.0, your installation file changes only .NET Framework version 2.0 policy.
* In some cases, the installer does not generate an error if installation of a new policy failed. To verify that policy was installed succesfully, inspect policy by using the .NET Framework Configuration tool, the [Caspol.exe (Code Access Security Policy Tool)](https://msdn.microsoft.com/en-us/library/cb6t8dtz(v=vs.100).aspx), or by manually inspecting the policy files in a text editor after deployment.
* To update the policy displayed by the .NET Framework Configuration tool, you must shut down the tool and restart it.

[**Custom Deployment**](javascript:void(0))

You can deploy Windows Installer files in several ways, including a startup script, e-mail distribution, or distribution from a shared drive. The easiest way to deploy security policy from a Windows Installer file is to run the file from the computer where you want to update the security policy. You can do this by simply double-clicking the .msi file. To roll back the installation, right-click the .msi file and choose**Uninstall**.

Make sure that the user account under which the policy is installed has adequate privileges to access the configuration files you are modifying. For example, if you are currently logged on using an account that does not have permission to modify the enterprise configuration file, and the .msi file you are deploying must modify the enterprise configuration file, the installation will not succeed. Note that the Windows Installer package does not produce an error if the current account does not have sufficient permission to modify the configuration file.

[**Group Policy Deployment**](javascript:void(0))

If you use a Windows server for policy administration, you can use Group Policy with a Windows Installer file to deploy security policy to the workstations on your network. Simply import the Installer file using the group policy MMC snap-in, or place the Installer file in a pre-existing directory that you use as an installation point. After you have configured Group Policy to publish the Installer file, the security policy will be updated the next time users log on to the network. Note that you must have a domain controller present on your network to deploy security policy using Group Policy. For more information about using Group Policy, see the Microsoft Windows Server Help.

[**SMS Deployment**](javascript:void(0))

You can use Microsoft Systems Management Server (SMS) to publish security policy to computers on a network. SMS is a standalone server product that manages software installation and configuration in large enterprises. SMS is particularly useful in Windows Server-based networks because it provides the Group Policy functionality that Windows Server-based networks have. Use one of the compatible methods to convert the .msi file into an SMS software package, then use SMS to install the package in the same way as any other software package. For more information about creating and deploying SMS software packages, see the SMS documentation.

### Enterprise Policy Administration

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4**

|  |
| --- |
| **Important noteImportant** |
| In the .NET Framework version 4, the common language runtime (CLR) is moving away from providing security policy for computers. Microsoft is recommending the use of Windows Software Restriction Policies as a replacement for CLR security policy. The information in this topic applies to the .NET Framework version 3.5 and earlier; it does not apply to version 4.0 and later. For more information about this and other changes, see [Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx). |

The enterprise policy level affects every computer and user on the network and can only be administered by enterprise or domain administrators. See the section on [Deploying Security Policy](https://msdn.microsoft.com/en-us/library/13wcxx6y(v=vs.71).aspx) for information on deployment strategies.

Because the runtime evaluates enterprise policy first, you can apply the **LevelFinal**attribute to a code group on this level to exclude the lower levels from making policy changes. If you do not apply the **LevelFinal** attribute to code groups on this level, administrators of lower security levels will be able to assign more permissions to applications without your knowledge and potentially create security vulnerabilities.

You might consider administering policy on this level when every person in your enterprise uses an application and you want to make sure that it always receives sufficient permission to run.

### Machine Policy Administration

**.NET Framework 4 --**

|  |
| --- |
| **Important noteImportant** |
| In the .NET Framework version 4, the common language runtime (CLR) is moving away from providing security policy for computers. Microsoft is recommending the use of Windows Software Restriction Policies as a replacement for CLR security policy. The information in this topic applies to the .NET Framework version 3.5 and earlier; it does not apply to version 4.0 and later. For more information about this and other changes, see [Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx). |

The machine policy level holds most of the default security policy. All machine and domain administrators have access to the machine configuration files. Machine administrators can set policy that excludes modification from the user level but not from the enterprise level.

You might consider administering security policy on this level in the following situations:

* You are not on a network or are on a network without a domain controller.
* The computer you are administering serves a unique function. For example, if you are administering a public computer that is used for general Internet access by several people in a semi-public setting, you might want to have a unique machine policy, because the computer serves a unique function. Additionally, you might want to produce a specific machine policy that considers the security needs of specialized computers, like the servers in your enterprise.

### User Policy Administration

**.NET Framework 4 --**

|  |
| --- |
| **Important noteImportant** |
| In the .NET Framework version 4, the common language runtime (CLR) is moving away from providing security policy for computers. Microsoft is recommending the use of Windows Software Restriction Policies as a replacement for CLR security policy. The information in this topic applies to the .NET Framework version 3.5 and earlier; it does not apply to version 4.0 and later. For more information about this and other changes, see [Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx). |

* User policy is the lowest administrable policy level. Every user has an individual user policy configuration file. Any changes made to this policy level are applicable only to the current logged-on user. The user policy level is restricted in what it can specify.
* Because this level is configurable by the current logged-on user, enterprise level policy administrators should be aware that the user might potentially alter any policy changes made on the user policy level. The user policy level is not able to give more permissions to an assembly than is specified in the higher policy levels. However, the user policy level is allowed to decrease permissions, which might potentially cause applications to stop functioning properly. If the **LevelFinal**attribute is applied to a code group on the machine or enterprise level, the user level is not allowed to tighten policy decisions that have been made on those levels.
* User level administration is appropriate in some situations to tightening security. For example, a user might decide to tighten security policy for assemblies that originate from the local intranet zone if untrusted code is found there. You might consider administering policy on this level when you are a user on a corporate network and believe that the security settings are not tight enough.

## Secure Coding Guidelines

**.NET Framework 1.1**

Evidence-based security policy and code access security provide very powerful, explicit mechanisms to implement security. Most application code can simply use the infrastructure implemented by the .NET Framework. In some cases, additional application-specific security is required, built either by extending the security system or by using new ad hoc methods.

Using the .NET Framework-enforced permissions, and other enforcement in your code, you should erect barriers to prevent malicious code from obtaining information that you do not want it to have or performing other undesirable actions. Additionally, you must strike a balance between security and usability in all the expected scenarios using trusted code.

**In This Section**

[Secure Coding Overview](https://msdn.microsoft.com/en-us/library/8a3x2b7f(v=vs.71).aspx)

Provides an overview of basic secure coding techniques.

[Permission Requests](https://msdn.microsoft.com/en-us/library/d17fa5e4(v=vs.71).aspx)

Describes how to interact with the .NET Framework security system using security requests.

[Securing State Data](https://msdn.microsoft.com/en-us/library/39ww3547(v=vs.71).aspx)

Describes how to protect private members and boxed value types.

[Securing Method Access](https://msdn.microsoft.com/en-us/library/c09d4x9t(v=vs.71).aspx)

Describes how to help protect methods from being called by partially trusted code.

[Securing Wrapper Code](https://msdn.microsoft.com/en-us/library/6f5fa4y4(v=vs.71).aspx)

Describes security concerns for code that wraps other code.

[Securing Exception Handling](https://msdn.microsoft.com/en-us/library/8cd7yaws(v=vs.71).aspx)

Describes security concerns for handling exceptions.

[Security and User Input](https://msdn.microsoft.com/en-us/library/sbfk95yb(v=vs.71).aspx)

Describes security concerns for applications that accept user input.

[Security and Remoting Considerations](https://msdn.microsoft.com/en-us/library/82wf1hcz(v=vs.71).aspx)

Describes security concerns for applications that communicate across application domains.

[Security and Serialization](https://msdn.microsoft.com/en-us/library/ek7af9ck(v=vs.71).aspx)

Describes security concerns when serializing objects.

[Security and Race Conditions](https://msdn.microsoft.com/en-us/library/1az4z7cb(v=vs.71).aspx)

Describes how to avoid race conditions in your code.

[Security and On-the-Fly Code Generation](https://msdn.microsoft.com/en-us/library/x222e4ce(v=vs.71).aspx)

Describes security concerns for applications that generate dynamic code.

[Dangerous Permissions and Policy Administration](https://msdn.microsoft.com/en-us/library/wybyf7a0(v=vs.71).aspx)

Describes permissions that can potentially allow security to be circumvented.

[Security and Setup Issues](https://msdn.microsoft.com/en-us/library/sw5swefy(v=vs.71).aspx)

Describes considerations for testing and setup of your application.

Related Sections

[Securing ASP.NET Web Applications](https://msdn.microsoft.com/en-us/library/330a99hc(v=vs.71).aspx)

Describes ASP.NET security in detail and provides instructions for using it in your code.

[Code Access Security](https://msdn.microsoft.com/en-us/library/930b76w0(v=vs.71).aspx)

Describes .NET Framework code access security in detail and provides instructions for using it in your code.

[Role-Based Security](https://msdn.microsoft.com/en-us/library/52kd59t0(v=vs.71).aspx)

Describes .NET Framework role-based security in detail and provides instructions for using it in your code.

[Security Policy Management](https://msdn.microsoft.com/en-us/library/c1k0eed6(v=vs.71).aspx)

Describes the .NET Framework security policy model.

### Secure Coding Overview

**.NET Framework 1.1, 2.0, 3.0, 3.5**

This section provides an overview of the different ways code can be designed to work with the security system.

**Security-Neutral Code**

Security-neutral code does nothing explicit with the security system. It runs with whatever permissions it receives. Although applications that fail to catch security exceptions associated with protected operations (such as using files, networking, and so on) can result in an unhandled exception, security-neutral code still takes advantage of the .NET Framework security technologies.

A security-neutral library has special characteristics that you should understand. Suppose your library provides API elements that use files or call unmanaged code; if your code does not have the corresponding permission, it will not run as described. However, even if the code has the permission, any application code that calls it must have the same permission in order to work. If the calling code does not have the right permission, a [SecurityException](https://msdn.microsoft.com/en-us/library/system.security.securityexception(v=vs.71).aspx) appears as a result of the code access security stack walk.

**Application Code that is Not a Reusable Component**

If your code is part of an application that will not be called by other code, security is simple and special coding might not be required. However, remember that malicious code can call your code. While code access security might stop malicious code from accessing resources, such code could still read values of your fields or properties that might contain sensitive information.

Additionally, if your code accepts user input from the Internet or other unreliable sources, you must be careful about malicious input.

**Managed Wrapper to Native Code Implementation**

Typically in this scenario, some useful functionality is implemented in native code that you want to make available to managed code. Managed wrappers are easy to write using either platform invoke or COM interop. However, if you do this, callers of your wrappers must have unmanaged code rights in order to succeed. Under default policy, this means that code downloaded from an intranet or the Internet will not work with the wrappers.

Rather than giving all applications that use these wrappers unmanaged code rights, it is better to give these rights only to the wrapper code. If the underlying functionality exposes no resources and the implementation is likewise "safe," the wrapper only needs to assert its rights, which enables any code to call through it. When resources are involved, security coding should be the same as the library code case described in the next section. Because the wrapper is potentially exposing callers to these resources, careful verification of the safety of the native code is necessary and is the wrapper's responsibility.

**Library Code that Exposes Protected Resources**

This is the most powerful and hence potentially dangerous (if done incorrectly) approach for security coding: Your library serves as an interface for other code to access certain resources that are not otherwise available, just as the classes of the .NET Framework enforce permissions for the resources they use. Wherever you expose a resource, your code must first demand the permission appropriate to the resource (that is, it must perform a security check) and then typically assert its rights to perform the actual operation.

### Secure Coding Guidelines

**.NET Framework 4, 4.5, 4.6**

Evidence-based security and code access security provide very powerful, explicit mechanisms to implement security. Most application code can simply use the infrastructure implemented by the .NET Framework. In some cases, additional application-specific security is required, built either by extending the security system or by using new ad hoc methods.

Using the .NET Framework-enforced permissions and other enforcement in your code, you should erect barriers to prevent malicious code from obtaining information that you do not want it to have or performing other undesirable actions. Additionally, you must strike a balance between security and usability in all the expected scenarios using trusted code.

This overview describes the different ways code can be designed to work with the security system.

|  |
| --- |
| **NoteNote** |
| In the .NET Framework version 4, there have been important changes to the .NET Framework security model and terminology. For more information about these changes, see [Security Changes in the .NET Framework 4](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.100).aspx). |

[**Security-Neutral Code**](javascript:void(0))

Security-neutral code does nothing explicit with the security system. It runs with whatever permissions it receives. Although applications that fail to catch security exceptions associated with protected operations (such as using files, networking, and so on) can result in an unhandled exception, security-neutral code still takes advantage of the .NET Framework security technologies.

A security-neutral library has special characteristics that you should understand. Suppose your library provides API elements that use files or call unmanaged code; if your code does not have the corresponding permission, it will not run as described. However, even if the code has the permission, any application code that calls it must have the same permission in order to work. If the calling code does not have the right permission, a [SecurityException](https://msdn.microsoft.com/en-us/library/system.security.securityexception(v=vs.100).aspx) appears as a result of the code access security stack walk.

[**Application Code That Is Not a Reusable Component**](javascript:void(0))

If your code is part of an application that will not be called by other code, security is simple and special coding might not be required. However, remember that malicious code can call your code. While code access security might stop malicious code from accessing resources, such code could still read values of your fields or properties that might contain sensitive information.

Additionally, if your code accepts user input from the Internet or other unreliable sources, you must be careful about malicious input.

[**Managed Wrapper to Native Code Implementation**](javascript:void(0))

Typically in this scenario, some useful functionality is implemented in native code that you want to make available to managed code. Managed wrappers are easy to write using either platform invoke or COM interop. However, if you do this, callers of your wrappers must have unmanaged code rights in order to succeed. Under default policy, this means that code downloaded from an intranet or the Internet will not work with the wrappers.

Instead of giving all applications that use these wrappers unmanaged code rights, it is better to give these rights only to the wrapper code. If the underlying functionality exposes no resources and the implementation is likewise safe, the wrapper only needs to assert its rights, which enables any code to call through it. When resources are involved, security coding should be the same as the library code case described in the next section. Because the wrapper is potentially exposing callers to these resources, careful verification of the safety of the native code is necessary and is the wrapper's responsibility.

[**Library Code That Exposes Protected Resources**](javascript:void(0))

This is the most powerful and hence potentially dangerous (if done incorrectly) approach for security coding: Your library serves as an interface for other code to access certain resources that are not otherwise available, just as the classes of the .NET Framework enforce permissions for the resources they use. Wherever you expose a resource, your code must first demand the permission appropriate to the resource (that is, it must perform a security check) and then typically assert its rights to perform the actual operation.

[**Related Topics**](javascript:void(0))

|  |  |
| --- | --- |
| **Title** | **Description** |
| [How to: Run Partially Trusted Code in a Sandbox](https://msdn.microsoft.com/en-us/library/bb763046(v=vs.100).aspx) | Explains how to run a partially trusted application in a restricted security environment, which limits the code access permissions granted to it. |
| [Permission Requests](https://msdn.microsoft.com/en-us/library/d17fa5e4(v=vs.100).aspx) | Describes how to interact with the .NET Framework security system using security requests. |
| [Securing State Data](https://msdn.microsoft.com/en-us/library/39ww3547(v=vs.100).aspx) | Describes how to protect private members. |
| [Securing Method Access](https://msdn.microsoft.com/en-us/library/c09d4x9t(v=vs.100).aspx) | Describes how to help protect methods from being called by partially trusted code. |
| [Securing Wrapper Code](https://msdn.microsoft.com/en-us/library/6f5fa4y4(v=vs.100).aspx) | Describes security concerns for code that wraps other code. |
| [Security and Public Read-only Array Fields](https://msdn.microsoft.com/en-us/library/ms172409(v=vs.100).aspx) | Describes security concerns for code that uses public read-only arrays found in .NET Framework libraries. |
| [Securing Exception Handling](https://msdn.microsoft.com/en-us/library/8cd7yaws(v=vs.100).aspx) | Describes security concerns for handling exceptions. |
| [Security and User Input](https://msdn.microsoft.com/en-us/library/sbfk95yb(v=vs.100).aspx) | Describes security concerns for applications that accept user input. |
| [Security and Remoting Considerations](https://msdn.microsoft.com/en-us/library/82wf1hcz(v=vs.100).aspx) | Describes security concerns for applications that communicate across application domains. |
| [Security and Serialization](https://msdn.microsoft.com/en-us/library/ek7af9ck(v=vs.100).aspx) | Describes security concerns when serializing objects. |
| [Security and Race Conditions](https://msdn.microsoft.com/en-us/library/1az4z7cb(v=vs.100).aspx) | Describes how to avoid race conditions in your code. |
| [Security and On-the-Fly Code Generation](https://msdn.microsoft.com/en-us/library/x222e4ce(v=vs.100).aspx) | Describes security concerns for applications that generate dynamic code. |
| [Dangerous Permissions and Policy Administration](https://msdn.microsoft.com/en-us/library/wybyf7a0(v=vs.100).aspx) | Describes permissions that can potentially allow security to be circumvented. |
| [Security and Setup Issues](https://msdn.microsoft.com/en-us/library/sw5swefy(v=vs.100).aspx) | Describes considerations for the testing and setup of your application. |
| [ASP.NET Web Application Security](https://msdn.microsoft.com/en-us/library/330a99hc(v=vs.100).aspx) | Describes ASP.NET security in detail and provides instructions for using it in your code. |
| [Code Access Security](https://msdn.microsoft.com/en-us/library/c5tk9z76(v=vs.100).aspx) | Describes .NET Framework code access security in detail and provides instructions for using it in your code. |
| [Role-Based Security](https://msdn.microsoft.com/en-us/library/shz8h065(v=vs.100).aspx) | Describes .NET Framework role-based security in detail and provides instructions for using it in your code. |

### Permission Requests

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4, 4.5**

Permission requests are the primary way to make your code security aware. You should include permission requests in applications that access protected resources. For more information, see [Code Access Security Basics](https://msdn.microsoft.com/en-us/library/33tceax8(v=vs.71).aspx). Requests allow you to do two things:

* Request the minimum permissions your code must receive to run.
* Ensure that your code receives only the permissions that it actually needs.

The following code example demonstrates a basic permission request.

VB

<assembly: FileIOPermissionAttribute(SecurityAction.RequestMinimum, Write := "C:\test.tmp"), \_

assembly: PermissionSet(SecurityAction.RequestOptional, Unrestricted := False)>

[C#]

[assembly:FileIOPermissionAttribute(SecurityAction.RequestMinimum, Write="C:\\test.tmp")]

[assembly:PermissionSet(SecurityAction.RequestOptional,Unrestricted=false)]

This example tells the .NET Framework security system that the code should not run unless it receives permission to write to C:\test.tmp. If the code ever encounters security policy that does not grant this permission, a [PolicyException](https://msdn.microsoft.com/en-us/library/system.security.policy.policyexception(v=vs.71).aspx) is raised and the code does not run. Using this request, you can be sure that your code will run only if it is granted this permission, and you do not have to worry about errors caused by having too few permissions.

This example also tells the system that no additional permissions are wanted. Absent this, your code will be granted whatever permissions policy chooses to give it. While extra permissions do not cause harm, having fewer permissions could prevent some unforeseen security problems. Carrying permissions that your code does not need can lead to security problems.

Another way to limit the permissions your code receives to the fewest privileges is to list specific permissions you want to refuse. Permissions are typically refused when you ask that all permissions be optional and exclude specific permissions from that request. For more information, see [Refusing Permissions](https://msdn.microsoft.com/en-us/library/4b7hy971(v=vs.71).aspx).

### Securing State Data

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4, 4.5, 4.6**

Applications that handle sensitive data or make any kind of security decisions need to keep that data under their own control and cannot allow other potentially malicious code to access the data directly. The best way to protect data in memory is to declare the data as private or internal (with scope limited to the same assembly) variables. However, even this data is subject to access you should be aware of:

* Using reflection mechanisms, highly trusted code that can reference your object can get and set private members.
* Using serialization, highly trusted code can effectively get and set private members if it can access the corresponding data in the serialized form of the object.
* Under debugging, this data can be read.

Make sure none of your own methods or properties exposes these values unintentionally.

In some cases, data can be declared as "protected," with access limited to the class and its derivatives. However, you should take the following additional precautions due to additional exposure:

* Control what code is allowed to derive from your class by restricting it to the same assembly or by using declarative security, described in [Securing Method Access](https://msdn.microsoft.com/en-us/library/c09d4x9t(v=vs.71).aspx), to require some identity or permissions in order for code to derive from your class.
* Ensure that all derived classes implement similar protection or are sealed.

#### Boxed Value Types

**.NET Framework 1.1, 2.0, 3.0**

Boxed value types can sometimes be modified in cases where you think you have distributed a copy of the type that cannot modify the original. When you return a boxed value type, you are returning a reference to the value type itself, rather than a reference to a copy of the value type. This allows the code that called your code to modify the value of your variable. For more information about boxed value types, see[Boxing and Unboxing (C# Programming Guide)](https://msdn.microsoft.com/en-us/library/yz2be5wk(v=vs.85).aspx).

The following example shows how boxed value types can be modified using a reference.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/f08s4k28(v=vs.85).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

using System;

using System.Reflection;

using System.Reflection.Emit;

using System.Threading;

using System.Collections;

class bug {

// Suppose you have an API element that exposes a

// field through a property with only a get accessor.

public object m\_Property;

public Object Property {

get { return m\_Property;}

set {m\_Property = value;} // (if applicable)

}

// You can modify the value of j by using

// the byref method with this signature.

public static void m1( ref int j ) {

j = Int32.MaxValue;

}

public static void m2( ref ArrayList j )

{

j = new ArrayList();

}

public static void Main(String[] args)

{

Console.WriteLine( "////// doing this with a value type" );

{

bug b = new bug();

b.m\_Property = 4;

Object[] objArr = new Object[]{b.Property};

Console.WriteLine( b.m\_Property );

typeof(bug).GetMethod( "m1" ).Invoke( null, objArr );

// Note that the property changes.

Console.WriteLine( b.m\_Property );

Console.WriteLine( objArr[0] );

}

Console.WriteLine( "////// doing this with a normal type" );

{

bug b = new bug();

ArrayList al = new ArrayList();

al.Add("elem");

b.m\_Property = al;

Object[] objArr = new Object[]{b.Property};

Console.WriteLine( ((ArrayList)(b.m\_Property)).Count );

typeof(bug).GetMethod( "m2" ).Invoke( null, objArr );

// Note that the property does not change.

Console.WriteLine( ((ArrayList)(b.m\_Property)).Count );

Console.WriteLine( ((ArrayList)(objArr[0])).Count );

}

}

}

The previous code displays the following to the console:

////// doing this with a value type

4

2147483647

2147483647

////// doing this with a normal type

1

1

0

### Securing Method Access

**.NET Framework 1.1**

[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635736413806036745

Some methods might not be suitable to allow arbitrary untrusted code to call. Such methods pose several risks: The method might provide some restricted information; it might believe any information passed to it; it might not do error checking on the parameters; or with the wrong parameters, it might malfunction or do something harmful. You should be aware of these cases and take action to help protect the method.

In some cases, you might need to restrict methods that are not intended for public use but still must be public. For example, you might have an interface that needs to be called across your own DLLs and hence must be public, but you do not want to expose it publicly to prevent customers from using it or to prevent malicious code from exploiting the entry point into your component. Another common reason to restrict a method not intended for public use (but that must be public) is to avoid having to document and support what might be a very internal interface.

Managed code offers several ways to restrict method access:

* Limit the scope of accessibility to the class, assembly, or derived classes, if they can be trusted. This is the simplest way to limit method access. Note that, in general, derived classes can be less trustworthy than the class they derive from, though in some cases they share the parent class's identity. In particular, do not infer trust from the keyword **protected**, which is not necessarily used in the security context.
* Limit the method access to callers of a specified identity--essentially, any particular [evidence](https://msdn.microsoft.com/en-us/library/7y5x1hcd(v=vs.71).aspx) (strong name, publisher, zone, and so on) you choose.
* Limit the method access to callers having whatever permissions you select.

Similarly, declarative security allows you to control inheritance of classes. You can use **InheritanceDemand** to do the following:

* Require derived classes to have a specified identity or permission.
* Require derived classes that override specific methods to have a specified identity or permission.

The following example shows how to help protect a public class for limited access by requiring that callers be signed with a particular strong name. This example uses the [StrongNameIdentityPermissionAttribute](https://msdn.microsoft.com/en-us/library/system.security.permissions.strongnameidentitypermissionattribute(v=vs.71).aspx) with a **Demand** for the strong name. For task-based information on how to sign an assembly with a strong name, see [Creating and Using Strong-Named Assemblies](https://msdn.microsoft.com/en-us/library/xwb8f617(v=vs.71).aspx).

VB

<StrongNameIdentityPermissionAttribute(SecurityAction.Demand, PublicKey := "...hex...", Name := "App1", Version := "0.0.0.0")> \_

Public Class Class1

End Class

[C#]

[StrongNameIdentityPermissionAttribute(SecurityAction.Demand, PublicKey="...hex...", Name="App1", Version="0.0.0.0")]

public class Class1

{

}

#### Excluding Classes and Members from Use by Untrusted Code

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4, 4.5**

Use the declarations shown in this section to prevent specific classes and methods, as well as properties and events, from being used by partially trusted code. By applying these declarations to a class, you apply the protection to all its methods, properties, and events; however, note that field access is not affected by declarative security. Note also that link demands help protect against only the immediate callers and might still be subject to luring attacks.

|  |
| --- |
| **Note Note** |
| A new transparency model has been introduced in the .NET Framework 4. The [Security-Transparent Code, Level 2](https://msdn.microsoft.com/en-us/library/dd233102(v=vs.110).aspx) model identifies secure code with the [SecurityCriticalAttribute](https://msdn.microsoft.com/en-us/library/system.security.securitycriticalattribute(v=vs.110).aspx) attribute. Security-critical code requires both callers and inheritors to be fully trusted. Assemblies that are running under the code access security rules from earlier .NET Framework versions can call level 2 assemblies. In this case, the security-critical attributes will be treated as link demands for full trust. |

In strong-named assemblies, a [LinkDemand](https://msdn.microsoft.com/en-us/library/hzsc022c(v=vs.71).aspx) is applied to all publicly accessible methods, properties, and events therein to restrict their use to fully trusted callers. To disable this feature, you must apply the [AllowPartiallyTrustedCallersAttribute](https://msdn.microsoft.com/en-us/library/system.security.allowpartiallytrustedcallersattribute(v=vs.71).aspx) attribute. Thus, explicitly marking classes to exclude untrusted callers is necessary only for unsigned assemblies or assemblies with this attribute; you can use these declarations to mark a subset of types therein that are not intended for untrusted callers.

The following examples show how to prevent classes and members from being used by untrusted code.

For public nonsealed classes:

VB

<System.Security.Permissions.PermissionSetAttribute(System.Security.Permissions.SecurityAction.InheritanceDemand, Name := "FullTrust"), \_

System.Security.Permissions.PermissionSetAttribute(System.Security.Permissions.SecurityAction.LinkDemand, Name := "FullTrust")> \_

Public Class CanDeriveFromMe

End Class

[C#]

[System.Security.Permissions.PermissionSetAttribute(System.Security.Permissions.SecurityAction.InheritanceDemand, Name="FullTrust")]

[System.Security.Permissions.PermissionSetAttribute(System.Security.Permissions.SecurityAction.LinkDemand, Name="FullTrust")]

public class CanDeriveFromMe

{

}

For public sealed classes:

VB

<System.Security.Permissions.PermissionSetAttribute(System.Security.Permissions.SecurityAction.LinkDemand, Name := "FullTrust")> \_

NotInheritable Public Class CannotDeriveFromMe

End Class

[C#]

[System.Security.Permissions.PermissionSetAttribute(System.Security.Permissions.SecurityAction.LinkDemand, Name="FullTrust")]

public sealed class CannotDeriveFromMe

{

}

For public abstract classes:

VB

<System.Security.Permissions.PermissionSetAttribute(System.Security.Permissions.SecurityAction.InheritanceDemand, Name := "FullTrust"), \_

System.Security.Permissions.PermissionSetAttribute(System.Security.Permissions.SecurityAction.LinkDemand, Name := "FullTrust")> \_

MustInherit Public Class CannotCreateInstanceOfMe\_CanCastToMe

End Class

[C#]

[System.Security.Permissions.PermissionSetAttribute(System.Security.Permissions.SecurityAction.InheritanceDemand, Name="FullTrust")]

[System.Security.Permissions.PermissionSetAttribute(System.Security.Permissions.SecurityAction.LinkDemand, Name="FullTrust")]

public abstract class CannotCreateInstanceOfMe\_CanCastToMe

{

}

For public virtual functions:

VB

Class Base

<System.Security.Permissions.PermissionSetAttribute(System.Security.Permissions.SecurityAction.InheritanceDemand, Name := "FullTrust"), \_

System.Security.Permissions.PermissionSetAttribute(System.Security.Permissions.SecurityAction.LinkDemand, Name := "FullTrust")> \_

Sub CanOverrideOrCallMe()

End Sub 'CanOverrideOrCallMe

End Class 'Base

[C#]

class Base {

[System.Security.Permissions.PermissionSetAttribute(System.Security.Permissions.SecurityAction.InheritanceDemand, Name="FullTrust")]

[System.Security.Permissions.PermissionSetAttribute(System.Security.Permissions.SecurityAction.LinkDemand, Name="FullTrust")]

public override void CanOverrideOrCallMe() { ... }

}

For public abstract functions:

VB

Class Base

<System.Security.Permissions.PermissionSetAttribute(System.Security.Permissions.SecurityAction.InheritanceDemand, Name := "FullTrust"), \_

System.Security.Permissions.PermissionSetAttribute(System.Security.Permissions.SecurityAction.LinkDemand, Name := "FullTrust")> \_

Sub CanOverrideMe()

End Sub

End Class

[C#]

class Base

{

[System.Security.Permissions.PermissionSetAttribute(System.Security.Permissions.SecurityAction.InheritanceDemand, Name="FullTrust")]

[System.Security.Permissions.PermissionSetAttribute(System.Security.Permissions.SecurityAction.LinkDemand, Name="FullTrust")]

public override void CanOverrideMe() { ... }

}

For public override functions where the base class does not demand full trust:

VB

Class Derived

<System.Security.Permissions.PermissionSetAttribute(System.Security.Permissions.SecurityAction.Demand, Name := "FullTrust")> \_

Sub CanOverrideOrCallMe()

End Sub

End Class

[C#]

class Derived

{

[System.Security.Permissions.PermissionSetAttribute(System.Security.Permissions.SecurityAction.Demand, Name="FullTrust")]

public override void CanOverrideOrCallMe() { ... }

}

For public override functions where the base class demands full trust:

VB

Class Derived

<System.Security.Permissions.PermissionSetAttribute(System.Security.Permissions.SecurityAction.LinkDemand,Name:="FullTrust")> \_

Sub CanOverrideOrCallMe()

End Sub

End Class

[C#]

class Derived

{

[System.Security.Permissions.PermissionSetAttribute(System.Security.Permissions.SecurityAction.LinkDemand,Name="FullTrust")]

public override void CanOverrideOrCallMe() { ... }

}

For public interfaces:

VB

<System.Security.Permissions.PermissionSetAttribute(System.Security.Permissions.SecurityAction.InheritanceDemand, Name:="FullTrust"), \_

System.Security.Permissions.PermissionSetAttribute(System.Security.Permissions.SecurityAction.LinkDemand, Name:="FullTrust")> \_

Public Interface CanCastToMe

End Interface

End Interface

[C#]

[System.Security.Permissions.PermissionSetAttribute(System.Security.Permissions.SecurityAction.InheritanceDemand, Name="FullTrust")]

[System.Security.Permissions.PermissionSetAttribute(System.Security.Permissions.SecurityAction.LinkDemand, Name="FullTrust")]

public interface CanCastToMe

#### Virtual Internal Overrides or Overloads Overridable Friend

**.NET Framework 1.1, 2.0**

You must be aware of a nuance of the type system accessibility when confirming that your code is unavailable to other assemblies. A method that is declared **virtual** and **internal** (**Overloads Overridable Friend** in Visual Basic) can override the parent class's vtable entry and can be used only from within the same assembly because it is internal. However, the accessibility for overriding is determined by the **virtual** keyword, and this can be overridden from another assembly as long as that code has access to the class itself. If the possibility of an override presents a problem, use declarative security to fix it, or remove the **virtual** keyword if it is not strictly required.

Note that even if a language compiler prevents these overrides with a compilation error, it is possible for code written with other compilers to override.

#### Virtual Internal Overrides or Overloads Overridable Friend

**.NET Framework 3.0, 3.5, 4, 4.5**

|  |
| --- |
| **Note Note** |
| This topic warns about a security issue when declaring a method as both **virtual** and **internal** (**OverloadsOverridableFriend** in Visual Basic). This warning applies only to the .NET Framework versions 1.0 and 1.1, it does not apply to version 2.0. |

In the .NET Framework versions 1.0 and 1.1 you must be aware of a nuance of the type system accessibility when confirming that your code is unavailable to other assemblies. A method that is declared **virtual** and **internal** (**Overloads Overridable Friend** in Visual Basic) can override the parent class's vtable entry and can be used only from within the same assembly because it is internal. However, the accessibility for overriding is determined by the **virtual** keyword, and this can be overridden from another assembly as long as that code has access to the class itself. If the possibility of an override presents a problem, use declarative security to fix it, or remove the **virtual** keyword if it is not strictly required.

Note that even if a language compiler prevents these overrides with a compilation error, it is possible for code written with other compilers to override.

### Securing Wrapper Code

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4**, **4.5, 4.6**

Wrapper code, especially where the wrapper has higher trust than code that uses it, can open a unique set of security weaknesses. Anything done on behalf of a caller, where the caller's limited permissions are not included in the appropriate security check, is a potential weakness to be exploited.

Never enable something through the wrapper that the caller could not do itself. This is a special danger when doing something that involves a limited security check, as opposed to a full stack walk demand. When single-level checks are involved, interposing the wrapper code between the real caller and the API element in question can easily cause the security check to succeed when it should not, thereby weakening security.

#### [**Delegates**](javascript:void(0))

Delegate security differs between versions of the .NET Framework. This section describes the different delegate behaviors and associated security considerations.

[**In version 1.0 and 1.1 of the .NET Framework**](javascript:void(0))

Version 1.0 and 1.1 of the .NET Framework perform the following security actions against a delegate creator and a delegate caller.

* When a delegate is created, security link demands on the delegate target method are performed against the grant set of the delegate creator. Failure to satisfy the security action results in a [SecurityException](https://msdn.microsoft.com/en-us/library/system.security.securityexception(v=vs.110).aspx).
* When the delegate is invoked, any existing security demands on the delegate caller are performed.

Whenever your code takes a [Delegate](https://msdn.microsoft.com/en-us/library/system.delegate(v=vs.110).aspx) from less-trusted code that might call it, make sure that you are not enabling the less-trusted code to escalate its permissions. If you take a delegate and use it later, the code that created the delegate is not on the call stack and its permissions will not be tested if code in or under the delegate attempts a protected operation. If your code and the caller code have higher privileges than the creator, the creator can orchestrate the call path without being part of the call stack.

[**In version 2.0 and later of the .NET Framework**](javascript:void(0))

Unlike previous versions, version 2.0 of the .NET Framework performs security action against the delegate creator when the delegate is created and called.

* When a delegate is created, security link demands on the delegate target method are performed against the grant set of the delegate creator. Failure to satisfy the security action results in a [SecurityException](https://msdn.microsoft.com/en-us/library/system.security.securityexception(v=vs.110).aspx).
* The delegate creator's grant set is also captured during delegate creation and stored with the delegate.
* When the delegate is invoked, the delegate creator's captured grant set is first evaluated against any demands in the current context if the delegate creator and caller belong to different assemblies. Next, any existing security demands on the delegate caller are performed.

#### [**Link demands and wrappers**](javascript:void(0))

A special protection case with link demands has been strengthened in the security infrastructure, but it is still a source of possible weakness in your code.

If fully trusted code calls a property, event, or method protected by a [LinkDemand](https://msdn.microsoft.com/en-us/library/hzsc022c(v=vs.110).aspx), the call succeeds if the **LinkDemand** permission check for the caller is satisfied. Additionally, if the fully trusted code exposes a class that takes the name of a property and calls its **get** accessor using reflection, that call to the **get** accessor succeeds even though the user code does not have the right to access this property. This is because the **LinkDemand** checks only the immediate caller, which is the fully trusted code. In essence, the fully trusted code is making a privileged call on behalf of user code without making sure that the user code has the right to make that call.

To help prevent such security holes, the common language runtime extends the check into a full stack-walking demand on any indirect call to a method, constructor, property, or event protected by a **LinkDemand**. This protection incurs some performance costs and changes the semantics of the security check; the full stack-walk demand might fail where the faster, one-level check would have passed.

|  |
| --- |
| **Note Note** |
| In the .NET Framework 4, partially trusted code has been redefined as transparent code. The transparency model draws a barrier between code that can do privileged things (critical code), such as calling native code, and code that cannot (transparent code). Transparency replaces the use of the [LinkDemand](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.110).aspx) for full trust to identify fully trusted code with the [SecurityCriticalAttribute](https://msdn.microsoft.com/en-us/library/system.security.securitycriticalattribute(v=vs.110).aspx). For more information about this and other changes, see [Security Changes in the .NET Framework](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.110).aspx). |

#### [**Assembly loading wrappers**](javascript:void(0))

Several methods used to load managed code, including [Assembly.Load](https://msdn.microsoft.com/en-us/library/system.reflection.assembly.load(v=vs.110).aspx), load assemblies with the evidence of the caller. If you wrap any of these methods, the security system could use your code's permission grant, instead of the permissions of the caller to your wrapper, to load the assemblies. You should not allow less-trusted code to load code that is granted higher permissions than those of the caller to your wrapper.

Any code that has full trust or significantly higher trust than a potential caller (including an Internet-permissions-level caller) could weaken security in this way. If your code has a public method that takes a byte array and passes it to **Assembly.Load**, thereby creating an assembly on the caller's behalf, it might break security.

This issue applies to the following API elements:

* [System.AppDomain.DefineDynamicAssembly](https://msdn.microsoft.com/en-us/library/system.appdomain.definedynamicassembly(v=vs.110).aspx)
* [System.Appdomain.Load](https://msdn.microsoft.com/en-us/library/system.appdomain.load(v=vs.110).aspx)
* [System.Reflection.Assembly.LoadFrom](https://msdn.microsoft.com/en-us/library/system.reflection.assembly.loadfrom(v=vs.110).aspx)
* [System.Reflection.Assembly.Load](https://msdn.microsoft.com/en-us/library/system.reflection.assembly.load(v=vs.110).aspx)

#### [**Demand vs. LinkDemand**](javascript:void(0))

Declarative security offers two kinds of security checks that are similar but perform very different checks. You should understand both forms because the wrong choice can result in weak security or performance loss.

Declarative security offers the following security checks:

* [Demand](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.110).aspx) specifies the code access security stack walk. All callers on the stack must have the specified permission or identity to pass.**Demand** occurs on every call because the stack might contain different callers. If you call a method repeatedly, this security check occurs each time. **Demand** is good protection against luring attacks; unauthorized code trying to get through it will be detected.
* [LinkDemand](https://msdn.microsoft.com/en-us/library/hzsc022c(v=vs.110).aspx) happens at just-in-time (JIT) compilation time and checks only the immediate caller. This security check does not check the caller's caller. Once this check passes, there is no additional security overhead no matter how many times the caller might call. However, there is also no protection from luring attacks. With **LinkDemand**, any code that passes the test and can reference your code can potentially break security by allowing malicious code to call using the authorized code. Therefore, do not use **LinkDemand**unless all the possible weaknesses can be thoroughly avoided.

|  |
| --- |
| **Note Note** |
| In the .NET Framework 4, link demands have been replaced by the [SecurityCriticalAttribute](https://msdn.microsoft.com/en-us/library/system.security.securitycriticalattribute(v=vs.110).aspx) attribute in [Level2](https://msdn.microsoft.com/en-us/library/system.security.securityruleset(v=vs.110).aspx) assemblies. The[SecurityCriticalAttribute](https://msdn.microsoft.com/en-us/library/system.security.securitycriticalattribute(v=vs.110).aspx) is equivalent to a link demand for full trust; however, it also affects inheritance rules. For more information about this change, see [Security-Transparent Code, Level 2](https://msdn.microsoft.com/en-us/library/dd233102(v=vs.110).aspx). |

The extra precautions required when using **LinkDemand** must be programmed individually; the security system can help with enforcement. Any mistake opens a security weakness. All authorized code that uses your code must be responsible for implementing additional security by doing the following:

* Restricting the calling code's access to the class or assembly.
* Placing the same security checks on the calling code that appear on the code being called and obligating its callers to do so. For example, if you write code that calls a method that is protected with a **LinkDemand** for the [SecurityPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermission(v=vs.110).aspx) with the[UnmanagedCode](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag(v=vs.110).aspx) flag specified, your method should also make a **LinkDemand** (or **Demand**, which is stronger) for this permission. The exception is if your code uses the **LinkDemand**-protected method in a limited way that you decide is safe, given other security protection mechanisms (such as demands) in your code. In this exceptional case, the caller takes responsibility in weakening the security protection on the underlying code.
* Ensuring that your code's callers cannot trick your code into calling the protected code on their behalf. In other words, callers cannot force the authorized code to pass specific parameters to the protected code, or to get results back from it.

[**Interfaces and Link Demands**](javascript:void(0))

If a virtual method, property, or event with **LinkDemand** overrides a base class method, the base class method must also have the same**LinkDemand** for the overridden method in order to be effective. It is possible for malicious code to cast back to the base type and call the base class method. Also note that link demands can be added implicitly to assemblies that do not have the[AllowPartiallyTrustedCallersAttribute](https://msdn.microsoft.com/en-us/library/system.security.allowpartiallytrustedcallersattribute(v=vs.110).aspx) assembly-level attribute.

It is a good practice to protect method implementations with link demands when interface methods also have link demands. Note the following about using link demands with interfaces:

* The **AllowPartiallyTrustedCallersAttribute** attribute also applies to interfaces.
* You can place link demands on interfaces to selectively protect certain interfaces from being used by partially trusted code, such as when using the **AllowPartiallyTrustedCallersAttribute**attribute.
* If you have an interface defined in an assembly that does not contain the **AllowPartiallyTrustedCallersAttribute**attribute, you can implement that interface on a partially trusted class.
* If you place a **LinkDemand** on a public method of a class that implements an interface method, the **LinkDemand** will not be enforced if you then cast to the interface and call the method. In this case, because you linked against the interface, only the**LinkDemand** on the interface is honored.

Review the following items for security issues:

* Explicit link demands on interface methods. Make sure these link demands offer the expected protection. Determine whether malicious code can use a cast to get around the link demands as described previously.
* Virtual methods with link demands applied.
* Types and the interfaces they implement. These should use link demands consistently.

#### Unmanaged Code

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4, 4.5, 4.6**

Some library code needs to call into unmanaged code (for example, native code APIs, such as Win32). Because this means going outside the security perimeter for managed code, due caution is required. If your code is security-neutral, both your code and any code that calls it must have unmanaged code permission ([SecurityPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermission(v=vs.71).aspx) with the [UnmanagedCode](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag(v=vs.71).aspx)flag specified).

However, it is often unreasonable for your caller to have such powerful permissions. In such cases, your trusted code can be the go-between, similar to the managed wrapper or library code described in [Securing Wrapper Code](https://msdn.microsoft.com/en-us/library/6f5fa4y4(v=vs.71).aspx). If the underlying unmanaged code functionality is totally safe, it can be directly exposed; otherwise, a suitable permission check (demand) is required first.

When your code calls into unmanaged code but you do not want to require your callers to have permission to access unmanaged code, you must assert that right. An assertion blocks the stack walk at your frame. You must be careful that you do not create a security hole in this process. Usually, this means that you must demand a suitable permission of your callers and then use unmanaged code to perform only what that permission allows and no more. In some cases (for example, a get time-of-day function), unmanaged code can be directly exposed to callers without any security checks. In any case, any code that asserts must take responsibility for security.

Because any managed code that provides a code path into native code is a potential target for malicious code, determining which unmanaged code can be safely used and how it must be used requires extreme care. Generally, unmanaged code should never be directly exposed to partially trusted callers. There are two primary considerations in evaluating the safety of unmanaged code use in libraries that are callable by partially trusted code:

* **Functionality**. Does the unmanaged API provide functionality that does not allow callers to perform potentially dangerous operations? Code access security uses permissions to enforce access to resources, so consider whether the API uses files, a user interface, or threading, or whether it exposes protected information. If it does, the managed code wrapping it must demand the necessary permissions before allowing it to be entered. Additionally, while not protected by a permission, memory access must be confined to strict type safety.
* **Parameter checking**. A common attack passes unexpected parameters to exposed unmanaged code API methods in an attempt to cause them to operate out of specification. Buffer overruns using out-of-range index or offset values are one common example of this type of attack, as are any parameters that might exploit a bug in the underlying code. Thus, even if the unmanaged code API is functionally safe (after necessary demands) for partially trusted callers, managed code must also check parameter validity exhaustively to ensure that no unintended calls are possible from malicious code using the managed code wrapper layer.

##### Using SuppressUnmanagedCodeSecurityAttribute

**.NET Framework 1.1, 2.0 ,3.0, 3.5, 4, 4.5**

There is a performance aspect to asserting and then calling unmanaged code. For every such call, the security system automatically demands unmanaged code permission, resulting in a stack walk each time. If you assert and immediately call unmanaged code, the stack walk can be meaningless: it consists of your assert and your unmanaged code call.

A custom attribute called [SuppressUnmanagedCodeSecurityAttribute](https://msdn.microsoft.com/en-us/library/system.security.suppressunmanagedcodesecurityattribute(v=vs.71).aspx) can be applied to unmanaged code entry points to disable the normal security check that demands [SecurityPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermission(v=vs.71).aspx)with[UnmanagedCode](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag(v=vs.71).aspx)permission specified. Extreme caution must always be taken when doing this, because this action creates an open door into unmanaged code with no runtime security checks. It should be noted that even with**SuppressUnmanagedCodeSecurityAttribute**applied, there is a one-time security check that happens at just-in-time (JIT) compilation to ensure that the immediate caller has permission to call unmanaged code.

If you use the **SuppressUnmanagedCodeSecurityAttribute**, check the following points:

* Make the unmanaged code entry point internal or otherwise inaccessible outside your code.
* Any call into unmanaged code is a potential security hole. Make sure your code is not a portal for malicious code to indirectly call into unmanaged code and avoid a security check. Demand permissions, if appropriate.
* Use a naming convention to explicitly identify when you are creating a dangerous path into unmanaged code, as described in [Naming Convention for Unmanaged Code Methods](https://msdn.microsoft.com/en-us/library/btadwd4w(v=vs.71).aspx).

##### Naming Convention for Unmanaged Code Methods

**.NET Framework 1.1, 2.0 ,3.0, 3.5, 4, 4.5**

A useful and highly recommended convention has been established for naming unmanaged code methods. All unmanaged code methods are separated into three categories: **safe**, **native**, and **unsafe**. These keywords can be used as class names within which the various kinds of unmanaged code entry points are defined. In source code, these keywords should be added to the class name, as in Safe.GetTimeOfDay,Native.Xyz, or Unsafe.DangerousAPI, for example. Each of these keywords provides useful security information for developers using that class, as described in the following table.

|  |  |
| --- | --- |
| **Keyword** | **Security considerations** |
| **safe** | Completely harmless for any code, even malicious code, to call. Can be used just like other managed code. For example, a function that gets the time of day is typically safe. |
| **native** | Security-neutral; that is, unmanaged code that requires unmanaged code permission to call. Security is checked, which stops an unauthorized caller. |
| **unsafe** | Potentially dangerous unmanaged code entry point with security suppressed. Developers should use the greatest caution when using such unmanaged code, making sure that other protections are in place to help prevent a security vulnerability. Developers must be responsible, as this keyword overrides the security system. |

### Security and Public Read-only Array Fields

**.NET Framework 2.0, 3.0, 3.5, 4. 4.5, 4.6**

Never use read-only public array fields from managed libraries to define the boundary behavior or security of your applications because read-only public array fields can be modified.

**Remarks**

Some .NET framework classes include read-only public fields that contain platform-specific boundary parameters. For example, the[InvalidPathChars](https://msdn.microsoft.com/en-US/library/system.io.path.invalidpathchars(v=vs.80).aspx) field is an array that describes the characters that are not allowed in a file path string. Many similar fields are present throughout the .NET Framework.

The values of public read-only fields like **InvalidPathChars** can be modified by your code or code that shares your code’s application domain. You should not use read-only public array fields like this to define the boundary behavior of your applications. If you do, malicious code can alter the boundary definitions and use your code in unexpected ways.

In version 2.0 and later of the .NET Framework, you should use methods that return a new array instead of using public array fields. For example, instead of using the **InvalidPathChars** field, you should use the [GetInvalidPathChars](https://msdn.microsoft.com/en-US/library/ms143439(v=vs.80).aspx) method.

Note that the .NET Framework types do not use the public fields to define boundary types internally. Instead, the .NET Framework uses separate private fields. Changing the values of these public fields does not alter the behavior of .NET Framework types.

### Securing Exception Handling

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4, 4.5, 46.**

In the Managed Extensions for C++ and in Visual Basic, a filter expression further up the stack runs before any **finally** statement. The **catch**block associated with that filter runs after the **finally** statement. For more information, see [Using User-Filtered Exceptions](https://msdn.microsoft.com/en-us/library/4dy8x9k9(v=vs.71).aspx). This section examines the security implications of this order. Consider the following pseudocode example that illustrates the order in which filter statements and **finally** statements run.

C++

void Main()

{

try

{

Sub();

}

except (Filter())

{

Console.WriteLine("catch");

}

}

bool Filter () {

Console.WriteLine("filter");

return true;

}

void Sub()

{

try

{

Console.WriteLine("throw");

throw new Exception();

}

finally

{

Console.WriteLine("finally");

}

}

This code prints the following.

Throw

Filter

Finally

Catch

The filter runs before the **finally** statement, so security issues can be introduced by anything that makes a state change where execution of other code could take advantage. For example:

C++

try

{

Alter\_Security\_State();

// This means changing anything (state variables,

// switching unmanaged context, impersonation, and

// so on) that could be exploited if malicious

// code ran before state is restored.

Do\_some\_work();

}

finally

{

Restore\_Security\_State();

// This simply restores the state change above.

}

This pseudocode allows a filter higher up the stack to run arbitrary code. Other examples of operations that would have a similar effect are temporary impersonation of another identity, setting an internal flag that bypasses some security check, or changing the culture associated with the thread. The recommended solution is to introduce an exception handler to isolate the code's changes to thread state from callers' filter blocks. However, it is important that the exception handler be properly introduced or this problem will not be fixed. The following example switches the UI culture, but any kind of thread state change could be similarly exposed.

C++

YourObject.YourMethod()

{

CultureInfo saveCulture = Thread.CurrentThread.CurrentUICulture;

try {

Thread.CurrentThread.CurrentUICulture = new CultureInfo("de-DE");

// Do something that throws an exception.

}

finally {

Thread.CurrentThread.CurrentUICulture = saveCulture;

}

}

[Visual Basic]

Public Class UserCode

Public Shared Sub Main()

Try

Dim obj As YourObject = new YourObject

obj.YourMethod()

Catch e As Exception When FilterFunc

Console.WriteLine("An error occurred: '{0}'", e)

Console.WriteLine("Current Culture: {0}",

Thread.CurrentThread.CurrentUICulture)

End Try

End Sub

Public Function FilterFunc As Boolean

Console.WriteLine("Current Culture: {0}", Thread.CurrentThread.CurrentUICulture)

Return True

End Sub

End Class

The correct fix in this case is to wrap the existing **try**/**finally** block in a **try**/**catch**block. Simply introducing a **catch-throw** clause into the existing **try**/**finally** block does not fix the problem, as shown in the following example.

C++

YourObject.YourMethod()

{

CultureInfo saveCulture = Thread.CurrentThread.CurrentUICulture;

try

{

Thread.CurrentThread.CurrentUICulture = new CultureInfo("de-DE");

// Do something that throws an exception.

}

catch { throw; }

finally

{

Thread.CurrentThread.CurrentUICulture = saveCulture;

}

}

This does not fix the problem because the **finally** statement has not run before the FilterFunc gets control.

The following example fixes the problem by ensuring that the **finally** clause has executed before offering an exception up the callers' exception filter blocks.

C++

YourObject.YourMethod()

{

CultureInfo saveCulture = Thread.CurrentThread.CurrentUICulture;

try

{

try

{

Thread.CurrentThread.CurrentUICulture = new CultureInfo("de-DE");

// Do something that throws an exception.

}

finally

{

Thread.CurrentThread.CurrentUICulture = saveCulture;

}

}

catch { throw; }

}

### Security and User Input

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4, 4.5, 4.6**

User data, which is any kind of input (data from a Web request or URL, input to controls of a Microsoft Windows Forms application, and so on), can adversely influence code because often that data is used directly as parameters to call other code. This situation is analogous to malicious code calling your code with strange parameters, and the same precautions should be taken. User input is actually harder to make safe because there is no stack frame to trace the presence of the potentially untrusted data.

These are among the subtlest and hardest security bugs to find because, although they can exist in code that is seemingly unrelated to security, they are a gateway to pass bad data through to other code. To look for these bugs, follow any kind of input data, imagine what the range of possible values might be, and consider whether the code seeing this data can handle all those cases. You can fix these bugs through range checking and rejecting any input the code cannot handle.

Some important considerations involving user data include the following:

* Any user data in a server response runs in the context of the server's site on the client. If your Web server takes user data and inserts it into the returned Web page, it might, for example, include a **<script>** tag and run as if from the server.
* Remember that the client can request any URL.
* Consider tricky or invalid paths:
  + ..\ , extremely long paths.
  + Use of wild card characters (\*).
  + Token expansion (%token%).
  + Strange forms of paths with special meaning.
  + Alternate file system stream names such as filename::$DATA.
  + Short versions of file names such as longfi~1 for longfilename.
* Remember that Eval(userdata) can do anything.
* Be wary of late binding to a name that includes some user data.
* If you are dealing with Web data, consider the various forms of escapes that are permissible, including:
  + Hexadecimal escapes (%nn).
  + Unicode escapes (%nnn).
  + Overlong UTF-8 escapes (%nn%nn).
  + Double escapes (%nn becomes %mmnn, where %mm is the escape for '%').
* Be wary of user names that might have more than one canonical format. For example, in Microsoft Windows 2000, you can often use either the MYDOMAIN\username form or the username@mydomain.example.com form.

### Security and Remoting Considerations

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4, 4.5, 4.6**

Remoting allows you to set up transparent calling between application domains, processes, or computers. However, the code access security stack walk cannot cross process or machine boundaries (it does apply between application domains of the same process).

Any class that is remotable (derived from a [MarshalByRefObject](https://msdn.microsoft.com/en-us/library/system.marshalbyrefobject(v=vs.71).aspx) class) needs to take responsibility for security. Either the code should be used only in closed environments where the calling code can be implicitly trusted, or remoting calls should be designed so that they do not subject protected code to outside entry that could be used maliciously.

Generally, you should never expose methods, properties, or events that are protected by declarative [LinkDemand](https://msdn.microsoft.com/en-us/library/hzsc022c(v=vs.71).aspx) and [InheritanceDemand](https://msdn.microsoft.com/en-us/library/x4yx82e6(v=vs.71).aspx)security checks. With remoting, these checks are not enforced. Other security checks, such as [Demand](https://msdn.microsoft.com/en-us/library/9kc0c6st(v=vs.71).aspx), [Assert](https://msdn.microsoft.com/en-us/library/91wteedy(v=vs.71).aspx), and so on, work between application domains within a process but do not work in cross-process or cross-machine scenarios.

#### [**Protected objects**](javascript:void(0))

Some objects hold security state in themselves. These objects should not be passed to untrusted code, which would then acquire security authorization beyond its own permissions.

One example is creating a [FileStream](https://msdn.microsoft.com/en-us/library/system.io.filestream(v=vs.110).aspx) object. The [FileIOPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.fileiopermission(v=vs.110).aspx) is demanded at the time of creation and, if it succeeds, the file object is returned. However, if this object reference is passed to code without file permissions, the object will be able to read and write to this particular file.

The simplest defense for such an object is to demand the same **FileIOPermission** of any code that seeks to get the object reference through a public API element.

#### [**Application domain crossing issues**](javascript:void(0))

To isolate code in managed hosting environments, it is common to generate multiple child application domains with explicit policy reducing the permission levels for various assemblies. However, the policy for those assemblies remains unchanged in the default application domain. If one of the child application domains can force the default application domain to load an assembly, the effect of code isolation is lost and types in the forcibly loaded assembly will be able to run code at a higher level of trust.

An application domain can force another application domain to load an assembly and run code contained therein by calling a proxy to an object hosted in the other application domain. To obtain a cross-application-domain proxy, the application domain hosting the object must distribute one through a method call parameter or return value. Or, if the application domain was just created, the creator has a proxy to the [AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.110).aspx) object by default. Thus, to avoid breaking code isolation, an application domain with a higher level of trust should not distribute references to marshaled-by-reference objects (instances of classes derived from [MarshalByRefObject](https://msdn.microsoft.com/en-us/library/system.marshalbyrefobject(v=vs.110).aspx)) in its domain to application domains with lower levels of trust.

Usually, the default application domain creates the child application domains with a control object in each one. The control object manages the new application domain and occasionally takes orders from the default application domain, but it cannot actually contact the domain directly. Occasionally, the default application domain calls its proxy to the control object. However, there might be cases in which it is necessary for the control object to call back to the default application domain. In these cases, the default application domain passes a marshal-by-reference callback object to the constructor of the control object. It is the responsibility of the control object to protect this proxy. If the control object were to place the proxy on a public static field of a public class, or otherwise publicly expose the proxy, this would open up a dangerous mechanism for other code to call back into the default application domain. For this reason, control objects are always implicitly trusted to keep the proxy private.

### Security and Serialization

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4, 4.5, 4.6**

Because serialization can allow other code to see or modify object instance data that would otherwise be inaccessible, a special permission is required of code performing serialization: [SecurityPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermission(v=vs.71).aspx) with the [SerializationFormatter](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag(v=vs.71).aspx) flag specified. Under default policy, this permission is not given to Internet-downloaded or intranet code; only code on the local computer is granted this permission.

Normally, all fields of an object instance are serialized, meaning that data is represented in the serialized data for the instance. It is possible for code that can interpret the format to determine what the data values are, independent of the accessibility of the member. Similarly, deserialization extracts data from the serialized representation and sets object state directly, again irrespective of accessibility rules.

Any object that could contain security-sensitive data should be made nonserializable, if possible. If it must be serializable, try to make specific fields that hold sensitive data nonserializable. If this cannot be done, be aware that this data will be exposed to any code that has permission to serialize, and make sure that no malicious code can get this permission.

The [ISerializable](https://msdn.microsoft.com/en-us/library/system.runtime.serialization.iserializable(v=vs.71).aspx) interface is intended for use only by the serialization infrastructure. However, if unprotected, it can potentially release sensitive information. If you provide custom serialization by implementing **ISerializable**, make sure you take the following precautions:

* The [GetObjectData](https://msdn.microsoft.com/en-us/library/system.runtime.serialization.iserializable.getobjectdata(v=vs.71).aspx) method should be explicitly secured either by demanding the **SecurityPermission**with**SerializationFormatter**permission specified or by making sure that no sensitive information is released with the method output. For example:

VB

Public Overrides<SecurityPermissionAttribute(SecurityAction.Demand, SerializationFormatter := True)> \_

Sub GetObjectData(info As SerializationInfo, context As StreamingContext)

End Sub

[C#]

[SecurityPermissionAttribute(SecurityAction.Demand,SerializationFormatter

=true)]

public override void GetObjectData(SerializationInfo info,

StreamingContext context)

{

}

* The special constructor used for serialization should also perform thorough input validation and should be either protected or private to help protect against misuse by malicious code. It should enforce the same security checks and permissions required to obtain an instance of such a class by any other means, such as explicitly creating the class or indirectly creating it through some kind of factory.

### Security and Race Conditions

**.NET Framework 4.6 and 4.5**

[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635736413806036745

Another area of concern is the potential for security holes exploited by race conditions. There are several ways in which this might happen. The subtopics that follow outline some of the major pitfalls that the developer must avoid.

[**Race Conditions in the Dispose Method**](javascript:void(0))

If a class's **Dispose** method (for more information, see [Garbage Collection](https://msdn.microsoft.com/en-us/library/0xy59wtx(v=vs.110).aspx)) is not synchronized, it is possible that cleanup code inside**Dispose** can be run more than once, as shown in the following example.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/1az4z7cb(v=vs.110).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

void Dispose()

{

if( myObj != null )

{

Cleanup(myObj);

myObj = null;

}

}

Because this **Dispose** implementation is not synchronized, it is possible for Cleanup to be called by first one thread and then a second thread before \_myObj is set to **null**. Whether this is a security concern depends on what happens when the Cleanup code runs. A major issue with unsynchronized **Dispose** implementations involves the use of resource handles such as files. Improper disposal can cause the wrong handle to be used, which often leads to security vulnerabilities.

[**Race Conditions in Constructors**](javascript:void(0))

In some applications, it might be possible for other threads to access class members before their class constructors have completely run. You should review all class constructors to make sure that there are no security issues if this should happen, or synchronize threads if necessary.

[**Race Conditions with Cached Objects**](javascript:void(0))

Code that caches security information or uses the code access security [Assert](https://msdn.microsoft.com/en-us/library/91wteedy(v=vs.110).aspx) operation might also be vulnerable to race conditions if other parts of the class are not appropriately synchronized, as shown in the following example.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/1az4z7cb(v=vs.110).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

void SomeSecureFunction()

{

if(SomeDemandPasses())

{

fCallersOk = true;

DoOtherWork();

fCallersOk = false();

}

}

void DoOtherWork()

{

if( fCallersOK )

{

DoSomethingTrusted();

}

else

{

DemandSomething();

DoSomethingTrusted();

}

}

If there are other paths to DoOtherWork that can be called from another thread with the same object, an untrusted caller can slip past a demand.

If your code caches security information, make sure that you review it for this vulnerability.

[**Race Conditions in Finalizers**](javascript:void(0))

Race conditions can also occur in an object that references a static or unmanaged resource that it then frees in its finalizer. If multiple objects share a resource that is manipulated in a class's finalizer, the objects must synchronize all access to that resource.

### Security and On-the-Fly Code Generation

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4, 4.5, 4.6**

Some libraries operate by generating code and running it to perform some operation for the caller. The basic problem is generating code on behalf of lesser-trust code and running it at a higher trust. The problem worsens when the caller can influence code generation, so you must ensure that only code you consider safe is generated.

You need to know exactly what code you are generating at all times. This means that you must have strict controls on any values that you get from a user, be they quote-enclosed strings (which should be escaped so they cannot include unexpected code elements), identifiers (which should be checked to verify that they are valid identifiers), or anything else. Identifiers can be dangerous because a compiled assembly can be modified so that its identifiers contain strange characters, which will probably break it (although this is rarely a security vulnerability).

It is recommended that you generate code with reflection emit, which often helps you avoid many of these problems.

When you compile the code, consider whether there is some way a malicious program could modify it. Is there a small window of time during which malicious code can change source code on disk before the compiler reads it or before your code loads the .dll file? If so, you must protect the directory containing these files, using code access security or an Access Control List in the file system, as appropriate.

If a caller can influence the generated code in a way that causes a compiler error, a security vulnerability might also exist there.

Run the generated code at the lowest possible permission settings, using [PermitOnly](https://msdn.microsoft.com/en-us/library/system.security.permissions.securityaction(v=vs.110).aspx) or [Deny](https://msdn.microsoft.com/en-us/library/hk3b9142(v=vs.110).aspx).

### Dangerous Permissions and Policy Administration

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4, 4.5, 4.6**

Several of the protected operations for which the .NET Framework provides permissions can potentially allow the security system to be circumvented. These dangerous permissions should be given only to trustworthy code, and then only as necessary. There is usually no defense against malicious code if it is granted these permissions.

|  |
| --- |
| **Note Note** |
| In the .NET Framework 4, there have been important changes to the .NET Framework security model and terminology. For more information about these changes, see [Security Changes in the .NET Framework](https://msdn.microsoft.com/en-us/library/dd233103(v=vs.110).aspx). |

The dangerous permissions are explained in the following table.

|  |  |
| --- | --- |
| **Permission** | **Potential risk** |
| [SecurityPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermission(v=vs.110).aspx) |  |
| [UnmanagedCode](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag(v=vs.110).aspx) | Allows managed code to call into unmanaged code, which is often dangerous. |
| [SkipVerification](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag(v=vs.110).aspx) | Without verification, the code can do anything. |
| [ControlEvidence](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag(v=vs.110).aspx) | Invalidated evidence can fool security policy. |
| [ControlPolicy](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag(v=vs.110).aspx) | The ability to modify security policy can disable security. |
| [SerializationFormatter](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag(v=vs.110).aspx) | The use of serialization can circumvent accessibility mechanisms. For details, see [Security and Serialization](https://msdn.microsoft.com/en-us/library/ek7af9ck(v=vs.110).aspx). |
| [ControlPrincipal](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag(v=vs.110).aspx) | The ability to set the current principal can trick role-based security. |
| [ControlThread](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag(v=vs.110).aspx) | Manipulation of threads is dangerous because of the security state associated with threads. |
| [ReflectionPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.reflectionpermission(v=vs.110).aspx) |  |
| [MemberAccess](https://msdn.microsoft.com/en-us/library/system.memberaccessexception(v=vs.110).aspx) | Can use private members to defeat accessibility mechanisms. |

### Security and Setup Issues

**.NET Framework 1.1, 2.0, 3.0, 3.5, 4, 4.5, 4.6**

The following steps are recommended when installing managed code or unmanaged code to ensure that the installation itself is secure. These steps should be performed for all platforms that support the NTFS file system:

* Set up a system with two partitions.
* Freshly format the second partition; do not change the default Access Control List on the root of the drive.
* Install the product, changing the install directory to point to a new directory on the second partition.

Ensure that none of the following is true:

* Is any code that executes as a service or that normally is run by administrator-level users now world-writable?
* If the code were installed on a terminal server system in application server mode, can your users now write binaries that other users might run?
* Is there anything that ends up in a system area or subdirectory of a system area that might be writable by non-administrators?

Additionally, if the product interacts with the Web, be aware that occasional Web server exploits allow users to run commands that are often executed in the context of the IUSR\_MACHINE account. Confirm that there are no files or configuration items that are world-writable that a guest account could leverage under these conditions.

## Secure Coding Guidelines for Unmanaged Code

**.NET Framework 4.6 and 4.5**

[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635736413806036745

Some library code needs to call into unmanaged code (for example, native code APIs, such as Win32). Because this means going outside the security perimeter for managed code, due caution is required. If your code is security-neutral, both your code and any code that calls it must have unmanaged code permission ([SecurityPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermission(v=vs.110).aspx) with the [UnmanagedCode](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag(v=vs.110).aspx) flag specified).

However, it is often unreasonable for your caller to have such powerful permissions. In such cases, your trusted code can be the go-between, similar to the managed wrapper or library code described in [Securing Wrapper Code](https://msdn.microsoft.com/en-us/library/6f5fa4y4(v=vs.110).aspx). If the underlying unmanaged code functionality is totally safe, it can be directly exposed; otherwise, a suitable permission check (demand) is required first.

When your code calls into unmanaged code but you do not want to require your callers to have permission to access unmanaged code, you must assert that right. An assertion blocks the stack walk at your frame. You must be careful that you do not create a security hole in this process. Usually, this means that you must demand a suitable permission of your callers and then use unmanaged code to perform only what that permission allows and no more. In some cases (for example, a get time-of-day function), unmanaged code can be directly exposed to callers without any security checks. In any case, any code that asserts must take responsibility for security.

Because any managed code that provides a code path into native code is a potential target for malicious code, determining which unmanaged code can be safely used and how it must be used requires extreme care. Generally, unmanaged code should never be directly exposed to partially trusted callers. There are two primary considerations in evaluating the safety of unmanaged code use in libraries that are callable by partially trusted code:

* **Functionality**. Does the unmanaged API provide functionality that does not allow callers to perform potentially dangerous operations? Code access security uses permissions to enforce access to resources, so consider whether the API uses files, a user interface, or threading, or whether it exposes protected information. If it does, the managed code wrapping it must demand the necessary permissions before allowing it to be entered. Additionally, while not protected by a permission, memory access must be confined to strict type safety.
* **Parameter checking**. A common attack passes unexpected parameters to exposed unmanaged code API methods in an attempt to cause them to operate out of specification. Buffer overruns using out-of-range index or offset values are one common example of this type of attack, as are any parameters that might exploit a bug in the underlying code. Thus, even if the unmanaged code API is functionally safe (after necessary demands) for partially trusted callers, managed code must also check parameter validity exhaustively to ensure that no unintended calls are possible from malicious code using the managed code wrapper layer.

[**Using SuppressUnmanagedCodeSecurityAttribute**](javascript:void(0))

There is a performance aspect to asserting and then calling unmanaged code. For every such call, the security system automatically demands unmanaged code permission, resulting in a stack walk each time. If you assert and immediately call unmanaged code, the stack walk can be meaningless: it consists of your assert and your unmanaged code call.

A custom attribute called [SuppressUnmanagedCodeSecurityAttribute](https://msdn.microsoft.com/en-us/library/system.security.suppressunmanagedcodesecurityattribute(v=vs.110).aspx) can be applied to unmanaged code entry points to disable the normal security check that demands [SecurityPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermission(v=vs.110).aspx) with [UnmanagedCode](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag(v=vs.110).aspx) permission specified. Extreme caution must always be taken when doing this, because this action creates an open door into unmanaged code with no runtime security checks. It should be noted that even with **SuppressUnmanagedCodeSecurityAttribute**applied, there is a one-time security check that happens at just-in-time (JIT) compilation to ensure that the immediate caller has permission to call unmanaged code.

If you use the **SuppressUnmanagedCodeSecurityAttribute**, check the following points:

* Make the unmanaged code entry point internal or otherwise inaccessible outside your code.
* Any call into unmanaged code is a potential security hole. Make sure your code is not a portal for malicious code to indirectly call into unmanaged code and avoid a security check. Demand permissions, if appropriate.
* Use a naming convention to explicitly identify when you are creating a dangerous path into unmanaged code, as described in the section below..

[**Naming convention for unmanaged code methods**](javascript:void(0))

A useful and highly recommended convention has been established for naming unmanaged code methods. All unmanaged code methods are separated into three categories: **safe**, **native**, and **unsafe**. These keywords can be used as class names within which the various kinds of unmanaged code entry points are defined. In source code, these keywords should be added to the class name, as in Safe.GetTimeOfDay,Native.Xyz, or Unsafe.DangerousAPI, for example. Each of these keywords provides useful security information for developers using that class, as described in the following table.

|  |  |
| --- | --- |
| **Keyword** | **Security considerations** |
| **safe** | Completely harmless for any code, even malicious code, to call. Can be used just like other managed code. For example, a function that gets the time of day is typically safe. |
| **native** | Security-neutral; that is, unmanaged code that requires unmanaged code permission to call. Security is checked, which stops an unauthorized caller. |
| **unsafe** | Potentially dangerous unmanaged code entry point with security suppressed. Developers should use the greatest caution when using such unmanaged code, making sure that other protections are in place to prevent a security vulnerability. Developers must be responsible, as this keyword overrides the security system. |

## How to: Run Partially Trusted Code in a Sandbox

**.NET Framework 4**

Sandboxing is the practice of running code in a restricted security environment, which limits the access permissions granted to the code. For example, if you have a managed library from a source you do not completely trust, you should not run it as fully trusted. Instead, you should place the code in a sandbox that limits its permissions to those that you expect it to need (for example, [Execution](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag(v=vs.100).aspx) permission).

You can also use sandboxing to test code you will be distributing that will run in partially trusted environments.

An [AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.100).aspx) is an effective way of providing a sandbox for managed applications. Application domains that are used for running partially trusted code have permissions that define the protected resources that are available when running within that [AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.100).aspx). Code that runs inside the [AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.100).aspx) is bound by the permissions associated with the [AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.100).aspx) and is allowed to access only the specified resources. The[AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.100).aspx) also includes a [StrongName](https://msdn.microsoft.com/en-us/library/system.security.policy.strongname(v=vs.100).aspx) array that is used to identify assemblies that are to be loaded as fully trusted. This enables the creator of an [AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.100).aspx) to start a new sandboxed domain that allows specific helper assemblies to be fully trusted. Another option for loading assemblies as fully trusted is to place them in the global assembly cache; however, that will load assemblies as fully trusted in all application domains created on that computer. The list of strong names supports a per-[AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.100).aspx) decision that provides more restrictive determination.

You can use the [AppDomain.CreateDomain(String, Evidence, AppDomainSetup, PermissionSet, StrongName[])](https://msdn.microsoft.com/en-us/library/ms130766(v=vs.100).aspx) method overload to specify the permission set for applications that run in a sandbox. This overload enables you to specify the exact level of code access security you want. Assemblies that are loaded into an [AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.100).aspx) by using this overload can either have the specified grant set only, or can be fully trusted. The assembly is granted full trust if it is in the global assembly cache or listed in the *fullTrustAssemblies* (the [StrongName](https://msdn.microsoft.com/en-us/library/system.security.policy.strongname(v=vs.100).aspx)) array parameter. Only assemblies known to be fully trusted should be added to the *fullTrustAssemblies* list.

The overload has the following signature:

AppDomain.CreateDomain( string friendlyName,

                        Evidence securityInfo,

                        AppDomainSetup info,

                        PermissionSet grantSet,

                        params StrongName[] fullTrustAssemblies);

The parameters for the [CreateDomain(String, Evidence, AppDomainSetup, PermissionSet, StrongName[])](https://msdn.microsoft.com/en-us/library/ms130766(v=vs.100).aspx) method overload specify the name of the [AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.100).aspx), the evidence for the [AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.100).aspx), the [AppDomainSetup](https://msdn.microsoft.com/en-us/library/system.appdomainsetup(v=vs.100).aspx) object that identifies the application base for the sandbox, the permission set to use, and the strong names for fully trusted assemblies.

For security reasons, the application base specified in the *info* parameter should not be the application base for the hosting application.

For the *grantSet* parameter, you can specify either a permission set you have explicitly created, or a standard permission set created by the[GetStandardSandbox](https://msdn.microsoft.com/en-us/library/dd414122(v=vs.100).aspx) method.

Unlike most [AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.100).aspx) loads, the evidence for the [AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.100).aspx) (which is provided by the *securityInfo* parameter) is not used to determine the grant set for the partially trusted assemblies. Instead, it is independently specified by the *grantSet* parameter. However, the evidence can be used for other purposes such as determining the isolated storage scope.

**To run an application in a sandbox**

1. Create the permission set to be granted to the untrusted application. The minimum permission you can grant is [Execution](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag(v=vs.100).aspx) permission. You can also grant additional permissions you think might be safe for untrusted code; for example, [IsolatedStorageFilePermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.isolatedstoragefilepermission(v=vs.100).aspx). The following code creates a new permission set with only [Execution](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag(v=vs.100).aspx) permission.
2. PermissionSet permSet = new PermissionSet(PermissionState.None);
3. permSet.AddPermission(new SecurityPermission(SecurityPermissionFlag.Execution));

Alternatively, you can use an existing named permission set, such as Internet.

Evidence ev = new Evidence();

ev.AddHostEvidence(new Zone(SecurityZone.Internet));

PermissionSet internetPS = SecurityManager.GetStandardSandbox(ev);

The [GetStandardSandbox](https://msdn.microsoft.com/en-us/library/dd414122(v=vs.100).aspx) method returns either an **Internet** permission set or a **LocalIntranet** permission set depending on the zone in the evidence. [GetStandardSandbox](https://msdn.microsoft.com/en-us/library/dd414122(v=vs.100).aspx) also constructs identity permissions for some of the evidence objects passed as references.

1. Sign the assembly that contains the hosting class (named Sandboxer in this example) that calls the untrusted code. Add the[StrongName](https://msdn.microsoft.com/en-us/library/system.security.policy.strongname(v=vs.100).aspx) used to sign the assembly to the [StrongName](https://msdn.microsoft.com/en-us/library/system.security.policy.strongname(v=vs.100).aspx) array of the *fullTrustAssemblies* parameter of the [CreateDomain](https://msdn.microsoft.com/en-us/library/system.appdomain.createdomain(v=vs.100).aspx) call. The hosting class must run as fully trusted to enable the execution of the partial-trust code or to offer services to the partial-trust application. This is how you read the [StrongName](https://msdn.microsoft.com/en-us/library/system.security.policy.strongname(v=vs.100).aspx) of an assembly:
2. StrongName fullTrustAssembly = typeof(Sandboxer).Assembly.Evidence.GetHostEvidence<StrongName>();

.NET Framework assemblies such as mscorlib and System.dll do not have to be added to the full-trust list because they are loaded as fully trusted from the global assembly cache.

1. Initialize the [AppDomainSetup](https://msdn.microsoft.com/en-us/library/system.appdomainsetup(v=vs.100).aspx) parameter of the [CreateDomain](https://msdn.microsoft.com/en-us/library/system.appdomain.createdomain(v=vs.100).aspx) method. With this parameter, you can control many of the settings of the new [AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.100).aspx). The [ApplicationBase](https://msdn.microsoft.com/en-us/library/system.appdomainsetup.applicationbase(v=vs.100).aspx) property is an important setting, and should be different from the [ApplicationBase](https://msdn.microsoft.com/en-us/library/system.appdomainsetup.applicationbase(v=vs.100).aspx) property for the [AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.100).aspx) of the hosting application. If the [ApplicationBase](https://msdn.microsoft.com/en-us/library/system.appdomainsetup.applicationbase(v=vs.100).aspx) settings are the same, the partial-trust application can get the hosting application to load (as fully trusted) an exception it defines, thus exploiting it. This is another reason why a catch (exception) is not recommended. Setting the application base of the host differently from the application base of the sandboxed application mitigates the risk of exploits.
2. AppDomainSetup adSetup = new AppDomainSetup();
3. adSetup.ApplicationBase = Path.GetFullPath(pathToUntrusted);
4. Call the [CreateDomain(String, Evidence, AppDomainSetup, PermissionSet, StrongName[])](https://msdn.microsoft.com/en-us/library/ms130766(v=vs.100).aspx) method overload to create the application domain using the parameters we have specified.

The signature for this method is:

public static AppDomain CreateDomain(string friendlyName,

Evidence securityInfo, AppDomainSetup info, PermissionSet grantSet,

params StrongName[] fullTrustAssemblies)

Additional information:

* + This is the only overload of the [CreateDomain](https://msdn.microsoft.com/en-us/library/system.appdomain.createdomain(v=vs.100).aspx) method that takes a [PermissionSet](https://msdn.microsoft.com/en-us/library/system.security.permissionset(v=vs.100).aspx) as a parameter, and thus the only overload that lets you load an application in a partial-trust setting.
  + The *evidence* parameter is not used to calculate a permission set; it is used for identification by other features of the .NET Framework.
  + Setting the [ApplicationBase](https://msdn.microsoft.com/en-us/library/system.appdomainsetup.applicationbase(v=vs.100).aspx) property of the *info* parameter is mandatory for this overload.
  + The *fullTrustAssemblies* parameter has the params keyword, which means that it is not necessary to create a [StrongName](https://msdn.microsoft.com/en-us/library/system.security.policy.strongname(v=vs.100).aspx) array. Passing 0, 1, or more strong names as parameters is allowed.
  + The code to create the application domain is:

AppDomain newDomain = AppDomain.CreateDomain("Sandbox", null, adSetup, permSet, fullTrustAssembly);

1. Load the code into the sandboxing [AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.100).aspx) that you created. This can be done in two ways:
   * Call the [ExecuteAssembly](https://msdn.microsoft.com/en-us/library/system.appdomain.executeassembly(v=vs.100).aspx) method for the assembly.
   * Use the [CreateInstanceFrom](https://msdn.microsoft.com/en-us/library/system.activator.createinstancefrom(v=vs.100).aspx) method to create an instance of a class derived from [MarshalByRefObject](https://msdn.microsoft.com/en-us/library/system.marshalbyrefobject(v=vs.100).aspx) in the new [AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.100).aspx).

The second method is preferable, because it makes it easier to pass parameters to the new [AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.100).aspx) instance. The[CreateInstanceFrom](https://msdn.microsoft.com/en-us/library/system.activator.createinstancefrom(v=vs.100).aspx) method provides two important features:

* + You can use a code base that points to a location that does not contain your assembly.
  + You can do the creation under an [Assert](https://msdn.microsoft.com/en-us/library/system.security.codeaccesspermission.assert(v=vs.100).aspx) for full-trust ([PermissionState.Unrestricted](https://msdn.microsoft.com/en-us/library/system.security.permissions.permissionstate(v=vs.100).aspx)), which enables you to create an instance of a critical class. (This happens whenever your assembly has no transparency markings and is loaded as fully trusted.) Therefore, you have to be careful to create only code that you trust with this function, and we recommend that you create only instances of fully trusted classes in the new application domain.

ObjectHandle handle = Activator.CreateInstanceFrom(

newDomain, typeof(Sandboxer).Assembly.ManifestModule.FullyQualifiedName,

typeof(Sandboxer).FullName );

Note that in order to create an instance of a class in a new domain, the class has to extend the [MarshalByRefObject](https://msdn.microsoft.com/en-us/library/system.marshalbyrefobject(v=vs.100).aspx) class

class Sandboxer:MarshalByRefObject

1. Unwrap the new domain instance into a reference in this domain. This reference is used to execute the untrusted code.
2. Sandboxer newDomainInstance = (Sandboxer) handle.Unwrap();
3. Call the ExecuteUntrustedCode method in the instance of the Sandboxer class you just created.
4. newDomainInstance.ExecuteUntrustedCode(untrustedAssembly, untrustedClass, entryPoint, parameters);

This call is executed in the sandboxed application domain, which has restricted permissions.

public void ExecuteUntrustedCode(string assemblyName, string typeName, string entryPoint, Object[] parameters)

{

//Load the MethodInfo for a method in the new assembly. This might be a method you know, or

//you can use Assembly.EntryPoint to get to the entry point in an executable.

MethodInfo target = Assembly.Load(assemblyName).GetType(typeName).GetMethod(entryPoint);

try

{

// Invoke the method.

target.Invoke(null, parameters);

}

catch (Exception ex)

{

//When information is obtained from a SecurityException extra information is provided if it is

//accessed in full-trust.

(new PermissionSet(PermissionState.Unrestricted)).Assert();

Console.WriteLine("SecurityException caught:\n{0}", ex.ToString());

CodeAccessPermission.RevertAssert();

Console.ReadLine();

}

}

[System.Reflection](https://msdn.microsoft.com/en-us/library/system.reflection(v=vs.100).aspx) is used to get a handle of a method in the partially trusted assembly. The handle can be used to execute code in a safe way with minimum permissions.

In the previous code, note the [Assert](https://msdn.microsoft.com/en-us/library/system.security.permissionset.assert(v=vs.100).aspx) for the full-trust permission before printing the [SecurityException](https://msdn.microsoft.com/en-us/library/system.security.securityexception(v=vs.100).aspx).

new PermissionSet(PermissionState.Unrestricted)).Assert()

The full-trust assert is used to obtain extended information from the [SecurityException](https://msdn.microsoft.com/en-us/library/system.security.securityexception(v=vs.100).aspx). Without the [Assert](https://msdn.microsoft.com/en-us/library/system.security.permissionset.assert(v=vs.100).aspx), the [ToString](https://msdn.microsoft.com/en-us/library/wykbfa67(v=vs.100).aspx) method of[SecurityException](https://msdn.microsoft.com/en-us/library/system.security.securityexception(v=vs.100).aspx) will discover that there is partially trusted code on the stack and will restrict the information returned. This could cause security issues if the partial-trust code could read that information, but the risk is mitigated by not granting [UIPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.uipermission(v=vs.100).aspx). The full-trust assert should be used sparingly and only when you are sure that you are not allowing partial-trust code to elevate to full trust. As a rule, do not call code you do not trust in the same function and after you called an assert for full trust. It is good practice to always revert the assert when you have finished using it.

[**Example**](javascript:void(0))

The following example implements the procedure in the previous section. In the example, a project named Sandboxer in a Visual Studio solution also contains a project named UntrustedCode, which implements the class UntrustedClass. This scenario assumes that you have downloaded a library assembly containing a method that is expected to return **true** or **false** to indicate whether the number you provided is a Fibonacci number. Instead, the method attempts to read a file from your computer. The following example shows the untrusted code.

using System;

using System.IO;

namespace UntrustedCode

{

public class UntrustedClass

{

// Pretend to be a method checking if a number is a Fibonacci

// but which actually attempts to read a file.

public static bool IsFibonacci(int number)

{

File.ReadAllText("C:\\Temp\\file.txt");

return false;

}

}

}

The following example shows the Sandboxer application code that executes the untrusted code.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.IO;

using System.Security;

using System.Security.Policy;

using System.Security.Permissions;

using System.Reflection;

using System.Runtime.Remoting;

//The Sandboxer class needs to derive from MarshalByRefObject so that we can create it in another

// AppDomain and refer to it from the default AppDomain.

class Sandboxer : MarshalByRefObject

{

const string pathToUntrusted = @"..\..\..\UntrustedCode\bin\Debug";

const string untrustedAssembly = "UntrustedCode";

const string untrustedClass = "UntrustedCode.UntrustedClass";

const string entryPoint = "IsFibonacci";

private static Object[] parameters = { 45 };

static void Main()

{

//Setting the AppDomainSetup. It is very important to set the ApplicationBase to a folder

//other than the one in which the sandboxer resides.

AppDomainSetup adSetup = new AppDomainSetup();

adSetup.ApplicationBase = Path.GetFullPath(pathToUntrusted);

//Setting the permissions for the AppDomain. We give the permission to execute and to

//read/discover the location where the untrusted code is loaded.

PermissionSet permSet = new PermissionSet(PermissionState.None);

permSet.AddPermission(new SecurityPermission(SecurityPermissionFlag.Execution));

//We want the sandboxer assembly's strong name, so that we can add it to the full trust list.

StrongName fullTrustAssembly = typeof(Sandboxer).Assembly.Evidence.GetHostEvidence<StrongName>();

//Now we have everything we need to create the AppDomain, so let's create it.

AppDomain newDomain = AppDomain.CreateDomain("Sandbox", null, adSetup, permSet, fullTrustAssembly);

//Use CreateInstanceFrom to load an instance of the Sandboxer class into the

//new AppDomain.

ObjectHandle handle = Activator.CreateInstanceFrom(

newDomain, typeof(Sandboxer).Assembly.ManifestModule.FullyQualifiedName,

typeof(Sandboxer).FullName

);

//Unwrap the new domain instance into a reference in this domain and use it to execute the

//untrusted code.

Sandboxer newDomainInstance = (Sandboxer) handle.Unwrap();

newDomainInstance.ExecuteUntrustedCode(untrustedAssembly, untrustedClass, entryPoint, parameters);

}

public void ExecuteUntrustedCode(string assemblyName, string typeName, string entryPoint, Object[] parameters)

{

//Load the MethodInfo for a method in the new Assembly. This might be a method you know, or

//you can use Assembly.EntryPoint to get to the main function in an executable.

MethodInfo target = Assembly.Load(assemblyName).GetType(typeName).GetMethod(entryPoint);

try

{

//Now invoke the method.

bool retVal = (bool)target.Invoke(null, parameters);

}

catch (Exception ex)

{

// When we print informations from a SecurityException extra information can be printed if we are

//calling it with a full-trust stack.

(new PermissionSet(PermissionState.Unrestricted)).Assert();

Console.WriteLine("SecurityException caught:\n{0}", ex.ToString());

CodeAccessPermission.RevertAssert();

Console.ReadLine();

}

}

}

## How to: Run Partially Trusted Code in a Sandbox

**.NET Framework 4.6 and 4.5**

[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635736413806036745

Sandboxing is the practice of running code in a restricted security environment, which limits the access permissions granted to the code. For example, if you have a managed library from a source you do not completely trust, you should not run it as fully trusted. Instead, you should place the code in a sandbox that limits its permissions to those that you expect it to need (for example, [Execution](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag(v=vs.110).aspx) permission).

You can also use sandboxing to test code you will be distributing that will run in partially trusted environments.

|  |
| --- |
| **Caution note Caution** |
| Code Access Security and Partially Trusted Code  The .NET Framework provides a mechanism for the enforcement of varying levels of trust on different code running in the same application called Code Access Security (CAS).  Code Access Security in .NET Framework should not  be used as a security boundary with partially trusted code, especially code of unknown origin. We advise against loading and executing code of unknown origins without putting alternative security measures in place.  This policy applies to all versions of .NET Framework, but does not apply to the .NET Framework included in Silverlight. |

An [AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.110).aspx) is an effective way of providing a sandbox for managed applications. Application domains that are used for running partially trusted code have permissions that define the protected resources that are available when running within that [AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.110).aspx). Code that runs inside the [AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.110).aspx) is bound by the permissions associated with the [AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.110).aspx) and is allowed to access only the specified resources. The[AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.110).aspx) also includes a [StrongName](https://msdn.microsoft.com/en-us/library/system.security.policy.strongname(v=vs.110).aspx) array that is used to identify assemblies that are to be loaded as fully trusted. This enables the creator of an [AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.110).aspx) to start a new sandboxed domain that allows specific helper assemblies to be fully trusted. Another option for loading assemblies as fully trusted is to place them in the global assembly cache; however, that will load assemblies as fully trusted in all application domains created on that computer. The list of strong names supports a per-[AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.110).aspx) decision that provides more restrictive determination.

You can use the [AppDomain.CreateDomain(String, Evidence, AppDomainSetup, PermissionSet, StrongName[])](https://msdn.microsoft.com/en-us/library/ms130766(v=vs.110).aspx) method overload to specify the permission set for applications that run in a sandbox. This overload enables you to specify the exact level of code access security you want. Assemblies that are loaded into an [AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.110).aspx) by using this overload can either have the specified grant set only, or can be fully trusted. The assembly is granted full trust if it is in the global assembly cache or listed in the *fullTrustAssemblies* (the [StrongName](https://msdn.microsoft.com/en-us/library/system.security.policy.strongname(v=vs.110).aspx)) array parameter. Only assemblies known to be fully trusted should be added to the *fullTrustAssemblies* list.

The overload has the following signature:

AppDomain.CreateDomain( string friendlyName,

                        Evidence securityInfo,

                        AppDomainSetup info,

                        PermissionSet grantSet,

                        params StrongName[] fullTrustAssemblies);

The parameters for the [CreateDomain(String, Evidence, AppDomainSetup, PermissionSet, StrongName[])](https://msdn.microsoft.com/en-us/library/ms130766(v=vs.110).aspx) method overload specify the name of the [AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.110).aspx), the evidence for the [AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.110).aspx), the [AppDomainSetup](https://msdn.microsoft.com/en-us/library/system.appdomainsetup(v=vs.110).aspx) object that identifies the application base for the sandbox, the permission set to use, and the strong names for fully trusted assemblies.

For security reasons, the application base specified in the *info* parameter should not be the application base for the hosting application.

For the *grantSet* parameter, you can specify either a permission set you have explicitly created, or a standard permission set created by the[GetStandardSandbox](https://msdn.microsoft.com/en-us/library/dd414122(v=vs.110).aspx) method.

Unlike most [AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.110).aspx) loads, the evidence for the [AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.110).aspx) (which is provided by the *securityInfo* parameter) is not used to determine the grant set for the partially trusted assemblies. Instead, it is independently specified by the *grantSet* parameter. However, the evidence can be used for other purposes such as determining the isolated storage scope.

**To run an application in a sandbox**

1. Create the permission set to be granted to the untrusted application. The minimum permission you can grant is [Execution](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag(v=vs.110).aspx) permission. You can also grant additional permissions you think might be safe for untrusted code; for example, [IsolatedStorageFilePermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.isolatedstoragefilepermission(v=vs.110).aspx). The following code creates a new permission set with only [Execution](https://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag(v=vs.110).aspx) permission.
2. PermissionSet permSet = new PermissionSet(PermissionState.None);
3. permSet.AddPermission(new SecurityPermission(SecurityPermissionFlag.Execution));

Alternatively, you can use an existing named permission set, such as Internet.

Evidence ev = new Evidence();

ev.AddHostEvidence(new Zone(SecurityZone.Internet));

PermissionSet internetPS = SecurityManager.GetStandardSandbox(ev);

The [GetStandardSandbox](https://msdn.microsoft.com/en-us/library/dd414122(v=vs.110).aspx) method returns either an **Internet** permission set or a **LocalIntranet** permission set depending on the zone in the evidence. [GetStandardSandbox](https://msdn.microsoft.com/en-us/library/dd414122(v=vs.110).aspx) also constructs identity permissions for some of the evidence objects passed as references.

1. Sign the assembly that contains the hosting class (named Sandboxer in this example) that calls the untrusted code. Add the[StrongName](https://msdn.microsoft.com/en-us/library/system.security.policy.strongname(v=vs.110).aspx) used to sign the assembly to the [StrongName](https://msdn.microsoft.com/en-us/library/system.security.policy.strongname(v=vs.110).aspx) array of the *fullTrustAssemblies* parameter of the [CreateDomain](https://msdn.microsoft.com/en-us/library/system.appdomain.createdomain(v=vs.110).aspx) call. The hosting class must run as fully trusted to enable the execution of the partial-trust code or to offer services to the partial-trust application. This is how you read the [StrongName](https://msdn.microsoft.com/en-us/library/system.security.policy.strongname(v=vs.110).aspx) of an assembly:
2. StrongName fullTrustAssembly = typeof(Sandboxer).Assembly.Evidence.GetHostEvidence<StrongName>();

.NET Framework assemblies such as mscorlib and System.dll do not have to be added to the full-trust list because they are loaded as fully trusted from the global assembly cache.

1. Initialize the [AppDomainSetup](https://msdn.microsoft.com/en-us/library/system.appdomainsetup(v=vs.110).aspx) parameter of the [CreateDomain](https://msdn.microsoft.com/en-us/library/system.appdomain.createdomain(v=vs.110).aspx) method. With this parameter, you can control many of the settings of the new [AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.110).aspx). The [ApplicationBase](https://msdn.microsoft.com/en-us/library/system.appdomainsetup.applicationbase(v=vs.110).aspx) property is an important setting, and should be different from the [ApplicationBase](https://msdn.microsoft.com/en-us/library/system.appdomainsetup.applicationbase(v=vs.110).aspx) property for the [AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.110).aspx) of the hosting application. If the [ApplicationBase](https://msdn.microsoft.com/en-us/library/system.appdomainsetup.applicationbase(v=vs.110).aspx) settings are the same, the partial-trust application can get the hosting application to load (as fully trusted) an exception it defines, thus exploiting it. This is another reason why a catch (exception) is not recommended. Setting the application base of the host differently from the application base of the sandboxed application mitigates the risk of exploits.
2. AppDomainSetup adSetup = new AppDomainSetup();
3. adSetup.ApplicationBase = Path.GetFullPath(pathToUntrusted);
4. Call the [CreateDomain(String, Evidence, AppDomainSetup, PermissionSet, StrongName[])](https://msdn.microsoft.com/en-us/library/ms130766(v=vs.110).aspx) method overload to create the application domain using the parameters we have specified.

The signature for this method is:

public static AppDomain CreateDomain(string friendlyName,

Evidence securityInfo, AppDomainSetup info, PermissionSet grantSet,

params StrongName[] fullTrustAssemblies)

Additional information:

* + This is the only overload of the [CreateDomain](https://msdn.microsoft.com/en-us/library/system.appdomain.createdomain(v=vs.110).aspx) method that takes a [PermissionSet](https://msdn.microsoft.com/en-us/library/system.security.permissionset(v=vs.110).aspx) as a parameter, and thus the only overload that lets you load an application in a partial-trust setting.
  + The *evidence* parameter is not used to calculate a permission set; it is used for identification by other features of the .NET Framework.
  + Setting the [ApplicationBase](https://msdn.microsoft.com/en-us/library/system.appdomainsetup.applicationbase(v=vs.110).aspx) property of the *info* parameter is mandatory for this overload.
  + The *fullTrustAssemblies* parameter has the params keyword, which means that it is not necessary to create a [StrongName](https://msdn.microsoft.com/en-us/library/system.security.policy.strongname(v=vs.110).aspx) array. Passing 0, 1, or more strong names as parameters is allowed.
  + The code to create the application domain is:

AppDomain newDomain = AppDomain.CreateDomain("Sandbox", null, adSetup, permSet, fullTrustAssembly);

1. Load the code into the sandboxing [AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.110).aspx) that you created. This can be done in two ways:
   * Call the [ExecuteAssembly](https://msdn.microsoft.com/en-us/library/system.appdomain.executeassembly(v=vs.110).aspx) method for the assembly.
   * Use the [CreateInstanceFrom](https://msdn.microsoft.com/en-us/library/system.activator.createinstancefrom(v=vs.110).aspx) method to create an instance of a class derived from [MarshalByRefObject](https://msdn.microsoft.com/en-us/library/system.marshalbyrefobject(v=vs.110).aspx) in the new [AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.110).aspx).

The second method is preferable, because it makes it easier to pass parameters to the new [AppDomain](https://msdn.microsoft.com/en-us/library/system.appdomain(v=vs.110).aspx) instance. The[CreateInstanceFrom](https://msdn.microsoft.com/en-us/library/system.activator.createinstancefrom(v=vs.110).aspx) method provides two important features:

* + You can use a code base that points to a location that does not contain your assembly.
  + You can do the creation under an [Assert](https://msdn.microsoft.com/en-us/library/system.security.codeaccesspermission.assert(v=vs.110).aspx) for full-trust ([PermissionState.Unrestricted](https://msdn.microsoft.com/en-us/library/system.security.permissions.permissionstate(v=vs.110).aspx)), which enables you to create an instance of a critical class. (This happens whenever your assembly has no transparency markings and is loaded as fully trusted.) Therefore, you have to be careful to create only code that you trust with this function, and we recommend that you create only instances of fully trusted classes in the new application domain.

ObjectHandle handle = Activator.CreateInstanceFrom(

newDomain, typeof(Sandboxer).Assembly.ManifestModule.FullyQualifiedName,

typeof(Sandboxer).FullName );

Note that in order to create an instance of a class in a new domain, the class has to extend the [MarshalByRefObject](https://msdn.microsoft.com/en-us/library/system.marshalbyrefobject(v=vs.110).aspx) class

class Sandboxer:MarshalByRefObject

1. Unwrap the new domain instance into a reference in this domain. This reference is used to execute the untrusted code.
2. Sandboxer newDomainInstance = (Sandboxer) handle.Unwrap();
3. Call the ExecuteUntrustedCode method in the instance of the Sandboxer class you just created.
4. newDomainInstance.ExecuteUntrustedCode(untrustedAssembly, untrustedClass, entryPoint, parameters);

This call is executed in the sandboxed application domain, which has restricted permissions.

public void ExecuteUntrustedCode(string assemblyName, string typeName, string entryPoint, Object[] parameters)

{

//Load the MethodInfo for a method in the new assembly. This might be a method you know, or

//you can use Assembly.EntryPoint to get to the entry point in an executable.

MethodInfo target = Assembly.Load(assemblyName).GetType(typeName).GetMethod(entryPoint);

try

{

// Invoke the method.

target.Invoke(null, parameters);

}

catch (Exception ex)

{

//When information is obtained from a SecurityException extra information is provided if it is

//accessed in full-trust.

(new PermissionSet(PermissionState.Unrestricted)).Assert();

Console.WriteLine("SecurityException caught:\n{0}", ex.ToString());

CodeAccessPermission.RevertAssert();

Console.ReadLine();

}

}

[System.Reflection](https://msdn.microsoft.com/en-us/library/system.reflection(v=vs.110).aspx) is used to get a handle of a method in the partially trusted assembly. The handle can be used to execute code in a safe way with minimum permissions.

In the previous code, note the [Assert](https://msdn.microsoft.com/en-us/library/system.security.permissionset.assert(v=vs.110).aspx) for the full-trust permission before printing the [SecurityException](https://msdn.microsoft.com/en-us/library/system.security.securityexception(v=vs.110).aspx).

new PermissionSet(PermissionState.Unrestricted)).Assert()

The full-trust assert is used to obtain extended information from the [SecurityException](https://msdn.microsoft.com/en-us/library/system.security.securityexception(v=vs.110).aspx). Without the [Assert](https://msdn.microsoft.com/en-us/library/system.security.permissionset.assert(v=vs.110).aspx), the [ToString](https://msdn.microsoft.com/en-us/library/wykbfa67(v=vs.110).aspx) method of[SecurityException](https://msdn.microsoft.com/en-us/library/system.security.securityexception(v=vs.110).aspx) will discover that there is partially trusted code on the stack and will restrict the information returned. This could cause security issues if the partial-trust code could read that information, but the risk is mitigated by not granting [UIPermission](https://msdn.microsoft.com/en-us/library/system.security.permissions.uipermission(v=vs.110).aspx). The full-trust assert should be used sparingly and only when you are sure that you are not allowing partial-trust code to elevate to full trust. As a rule, do not call code you do not trust in the same function and after you called an assert for full trust. It is good practice to always revert the assert when you have finished using it.

[**Example**](javascript:void(0))

The following example implements the procedure in the previous section. In the example, a project named Sandboxer in a Visual Studio solution also contains a project named UntrustedCode, which implements the class UntrustedClass. This scenario assumes that you have downloaded a library assembly containing a method that is expected to return **true** or **false** to indicate whether the number you provided is a Fibonacci number. Instead, the method attempts to read a file from your computer. The following example shows the untrusted code.

using System;

using System.IO;

namespace UntrustedCode

{

public class UntrustedClass

{

// Pretend to be a method checking if a number is a Fibonacci

// but which actually attempts to read a file.

public static bool IsFibonacci(int number)

{

File.ReadAllText("C:\\Temp\\file.txt");

return false;

}

}

}

The following example shows the Sandboxer application code that executes the untrusted code.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.IO;

using System.Security;

using System.Security.Policy;

using System.Security.Permissions;

using System.Reflection;

using System.Runtime.Remoting;

//The Sandboxer class needs to derive from MarshalByRefObject so that we can create it in another

// AppDomain and refer to it from the default AppDomain.

class Sandboxer : MarshalByRefObject

{

const string pathToUntrusted = @"..\..\..\UntrustedCode\bin\Debug";

const string untrustedAssembly = "UntrustedCode";

const string untrustedClass = "UntrustedCode.UntrustedClass";

const string entryPoint = "IsFibonacci";

private static Object[] parameters = { 45 };

static void Main()

{

//Setting the AppDomainSetup. It is very important to set the ApplicationBase to a folder

//other than the one in which the sandboxer resides.

AppDomainSetup adSetup = new AppDomainSetup();

adSetup.ApplicationBase = Path.GetFullPath(pathToUntrusted);

//Setting the permissions for the AppDomain. We give the permission to execute and to

//read/discover the location where the untrusted code is loaded.

PermissionSet permSet = new PermissionSet(PermissionState.None);

permSet.AddPermission(new SecurityPermission(SecurityPermissionFlag.Execution));

//We want the sandboxer assembly's strong name, so that we can add it to the full trust list.

StrongName fullTrustAssembly = typeof(Sandboxer).Assembly.Evidence.GetHostEvidence<StrongName>();

//Now we have everything we need to create the AppDomain, so let's create it.

AppDomain newDomain = AppDomain.CreateDomain("Sandbox", null, adSetup, permSet, fullTrustAssembly);

//Use CreateInstanceFrom to load an instance of the Sandboxer class into the

//new AppDomain.

ObjectHandle handle = Activator.CreateInstanceFrom(

newDomain, typeof(Sandboxer).Assembly.ManifestModule.FullyQualifiedName,

typeof(Sandboxer).FullName

);

//Unwrap the new domain instance into a reference in this domain and use it to execute the

//untrusted code.

Sandboxer newDomainInstance = (Sandboxer) handle.Unwrap();

newDomainInstance.ExecuteUntrustedCode(untrustedAssembly, untrustedClass, entryPoint, parameters);

}

public void ExecuteUntrustedCode(string assemblyName, string typeName, string entryPoint, Object[] parameters)

{

//Load the MethodInfo for a method in the new Assembly. This might be a method you know, or

//you can use Assembly.EntryPoint to get to the main function in an executable.

MethodInfo target = Assembly.Load(assemblyName).GetType(typeName).GetMethod(entryPoint);

try

{

//Now invoke the method.

bool retVal = (bool)target.Invoke(null, parameters);

}

catch (Exception ex)

{

// When we print informations from a SecurityException extra information can be printed if we are

//calling it with a full-trust stack.

(new PermissionSet(PermissionState.Unrestricted)).Assert();

Console.WriteLine("SecurityException caught:\n{0}", ex.ToString());

CodeAccessPermission.RevertAssert();

Console.ReadLine();

}

}

}

## Security Tools

**.NET Framework 1.1, 2.0, 3.0, 3.5**

The .NET Framework SDK supplies command-line tools that help you perform security-related tasks and test your components and applications before you deploy them. The following table briefly describes each of these tools.

|  |  |
| --- | --- |
| **Tool name** | **Description** |
| [Certificate Creation Tool (Makecert.exe)](https://msdn.microsoft.com/en-us/library/bfsktky3(v=vs.71).aspx) | Generates X.509 certificates for testing purposes only. |
| [Certificate Manager Tool (Certmgr.exe)](https://msdn.microsoft.com/en-us/library/e78byta0(v=vs.71).aspx) | Manages certificates, certificate trust lists (CTLs), and certificate revocation lists (CRLs). |
| [Certificate Verification Tool (Chktrust.exe)](https://msdn.microsoft.com/en-us/library/z045761b(v=vs.71).aspx) | Checks the validity of a file signed with an Authenticode™ certificate. |
| [Code Access Security Policy Tool (Caspol.exe)](https://msdn.microsoft.com/en-us/library/cb6t8dtz(v=vs.71).aspx) | Enables you to view and configure security policy. You can see the permissions that are granted to a specified assembly and the code groups that the assembly belongs to. |
| [File Signing Tool (Signcode.exe)](https://msdn.microsoft.com/en-us/library/9sh96ycy(v=vs.71).aspx) | Signs a portable executable (PE) file with requested permissions, giving you more control over the security restrictions placed on your components. |
| [Isolated Storage Tool (Storeadm.exe)](https://msdn.microsoft.com/en-us/library/ezabwsbk(v=vs.71).aspx) | Manages isolated storage, providing options to list the user's stores and delete them. |
| [Permissions View Tool (Permview.exe)](https://msdn.microsoft.com/en-us/library/06251f13(v=vs.71).aspx) | Allows you to view an assembly's requested permissions. |
| [PEVerify Tool (Peverify.exe)](https://msdn.microsoft.com/en-us/library/62bwd2yd(v=vs.71).aspx) | Determines whether the JIT compilation process can verify the type safety of the assembly. |
| [Secutil Tool (Secutil.exe)](https://msdn.microsoft.com/en-us/library/akt2ytd6(v=vs.71).aspx) | Extracts strong name public key information or Authenticode™ publisher certificates from an assembly, in a format that can be incorporated into code. |
| [Set Registry Tool (Setreg.exe)](https://msdn.microsoft.com/en-us/library/7zaf80x7(v=vs.71).aspx) | Changes the registry settings that pertain to certificates and digital signatures. |
| [Software Publisher Certificate Test Tool (Cert2spc.exe)](https://msdn.microsoft.com/en-us/library/f657tk8f(v=vs.71).aspx) | Creates a Software Publisher's Certificate (SPC) from one or more X.509 certificates. This tool is for testing purposes only. |
| [Strong Name Tool (Sn.exe)](https://msdn.microsoft.com/en-us/library/k5b5tt23(v=vs.71).aspx) | Helps create assemblies with strong names. Sn.exe provides options for key management, signature generation, and signature verification |

## ACL Technology Overview

**.NET Framework 2.0, 3.0, 3.5, 4, 4.5**

The classes in the [System.Security.AccessControl](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol(v=vs.80).aspx) namespace allow you to programmatically create or modify discretionary access control lists (DACLs) and system access control lists (SACLs) for a number of protected resources such as files, folders, and so on. DACLs allow you to programmatically control access to protected resources, while SACLs allow you to programmatically control system auditing policies of protected resources. For example, you can use the DACL classes to make sure that only an administrator can read a file; you can use the SACL classes to make sure that all successful attempts to open the file are logged.

The topics in this section describe the concepts and techniques that allow you to build ACL functionality into your applications.

In This Section

[Technology Summary for ACLs](https://msdn.microsoft.com/en-US/library/ms229936(v=vs.80).aspx)

Summarizes ACL concepts and the use of classes in the **System.Security.AccessControl** namespace.

[ACL Technology Architecture](https://msdn.microsoft.com/en-US/library/ms229748(v=vs.80).aspx)

Explains the architecture of ACL classes that are used to enforce access control and audit rules.

[ACL Technology Scenarios](https://msdn.microsoft.com/en-US/library/ms229925(v=vs.80).aspx)

Describes the most common scenarios in which ACL classes are used

[ACL Propagation Rules](https://msdn.microsoft.com/en-US/library/ms229747(v=vs.80).aspx)

Describes the rules used to propagate ACLs to folders and files contained within a target folder.

Reference

**System.Security.AccessControl**

Provides reference documentation for the **System.Security.AccessControl** namespace, which provides a managed implementation of the Windows ACL interface.

### Technology Summary for ACLs

**.NET Framework 2.0, 3.0, 3.5, 4, 4.5**

This topic summarizes information about the managed API for discretionary access control lists (DACLs), system access control lists (SACLs), and access control entries (ACEs). You can use the classes in the [System.Security.AccessControl](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol(v=vs.80).aspx) namespace to programmatically access DACLs, SACLs, and ACEs for several types of protected resources in the .NET Framework version 2.0 and later.

The managed ACL API makes working with ACLs easier than the unmanaged implementation by providing several classes that take care of many details for you. For example, it is not possible to create **null** DACLs in the managed implementation and you no longer have to worry about accidentally creating an insecure DACL because of incorrectly ordering your ACEs.

**Keywords**

Access control list (ACL), discretionary access control list (DACL), system access control list (SACL), access control entry (ACE), audit ACE

**Namespaces**

**System.Security.AccessControl**

**Related Technologies**

[Security in the .NET Framework](https://msdn.microsoft.com/en-US/library/fkytk30f(v=vs.80).aspx)

**Background**

A discretionary access control list (DACL), which is sometimes abbreviated to ACL, is a mechanism used by Microsoft Windows NT and later to protect resources such as files and folders. DACLs contain multiple access control entries (ACEs) that associate a principal (usually a user account or group of accounts) with a rule that governs the use of the resource. DACLs and ACEs let you allow or deny rights to resources based on permissions that you can associate with user accounts. For example, you can create an ACE and apply it to the DACL of a file to bar anyone but an administrator from reading the file.

A system access control list (SACL), which is sometimes referred to as an audit ACE, is a mechanism that controls the audit messages associated with a resource. Similar to DACLs, SACLs contain ACEs that define the audit rules for a given resource. Audit ACEs allow you to record successful or failed attempts to access a resource, but differ from access ACEs because they do not govern which accounts can use a resource. For example, you can create an ACE and apply it to the SACL of a file to log all successful attempts to open the file.

**ACL Classes at a Glance**

The following table lists the main classes that you can use to easily create and modify ACLs for several technology areas. This is not a comprehensive list of the **System.Security.AccessControl** namespace, but rather a list of the primary classes that you should use to work with ACLs.

|  |  |
| --- | --- |
| **Technology area** | **Classes** |
| Cryptographic keys | [CryptoKeySecurity](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol.cryptokeysecurity(v=vs.80).aspx)  [CryptoKeyAccessRule](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol.cryptokeyaccessrule(v=vs.80).aspx)  [CryptoKeyAuditRule](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol.cryptokeyauditrule(v=vs.80).aspx) |
| Directories | [DirectorySecurity](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol.directorysecurity(v=vs.80).aspx)  [FileSystemAccessRule](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol.filesystemaccessrule(v=vs.80).aspx)  [FileSystemAuditRule](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol.filesystemauditrule(v=vs.80).aspx) |
| Event Wait handles | [EventWaitHandleSecurity](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol.eventwaithandlesecurity(v=vs.80).aspx)  [EventWaitHandleAccessRule](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol.eventwaithandleaccessrule(v=vs.80).aspx)  [EventWaitHandleAuditRule](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol.eventwaithandleauditrule(v=vs.80).aspx) |
| Files | [FileSecurity](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol.filesecurity(v=vs.80).aspx)  **FileSystemAccessRule**  **FileSystemAuditRule** |
| Mutexes | [MutexSecurity](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol.mutexsecurity(v=vs.80).aspx)  [MutexAccessRule](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol.mutexaccessrule(v=vs.80).aspx)  [MutexAuditRule](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol.mutexauditrule(v=vs.80).aspx) |
| Registry keys | [RegistrySecurity](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol.registrysecurity(v=vs.80).aspx)  [RegistryAccessRule](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol.registryaccessrule(v=vs.80).aspx)  [RegistryAuditRule](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol.registryauditrule(v=vs.80).aspx) |
| Semaphores | [SemaphoreSecurity](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol.semaphoresecurity(v=vs.80).aspx)  [SemaphoreAccessRule](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol.semaphoreaccessrule(v=vs.80).aspx)  [SemaphoreAuditRule](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol.semaphoreauditrule(v=vs.80).aspx) |

To query the existing ACL information for a resource or to apply modified ACL information to a resource, you must use one of several methods that provide access to the ACLs of an existing resource. The classes in the previous table provide methods to construct and edit ACLs but they do not provide ways to query or apply ACLs.

Every resource has associated methods that apply ACLs during resource creation (usually a constructor overload), that retrieve the ACLs for an existing resource, and that apply ACLs to an existing resource.

The following table lists the methods used to get and set ACLs for each technology area. These methods are located in several namespaces outside the **System.Security.AccessControl** namespace and are part of the high-level classes that represent a particular resource. For example, to query the ACLs for a specified directory, you use the [System.IO.Directory.GetAccessControl(System.String)](https://msdn.microsoft.com/en-US/library/c1f66bc2(v=vs.80).aspx) method to get the**DirectorySecurity** object that encapsulates the DACLs and SACLs for the directory. Note that some resources, such as files and directories, have several equivalent methods in different classes that provide access to ACLs.

|  |  |
| --- | --- |
| **Technology area** | **Methods to get and set ACLs** |
| Cryptographic keys | [System.Security.Cryptography.CspParameters.CryptoKeySecurity](https://msdn.microsoft.com/en-US/library/ms148036(v=vs.80).aspx)  [System.Security.Cryptography.CspParameters.#ctor(System.Int32,System.String,System.String,System.Security.AccessControl.CryptoKeySecurity,System.Security.SecureString)](https://msdn.microsoft.com/en-US/library/ms148034(v=vs.80).aspx)  [System.Security.Cryptography.CspParameters.#ctor(System.Int32,System.String,System.String,System.Security.AccessControl.CryptoKeySecurity,System.IntPtr)](https://msdn.microsoft.com/en-US/library/ms148033(v=vs.80).aspx)  [System.Security.Cryptography.CspKeyContainerInfo.CryptoKeySecurity](https://msdn.microsoft.com/en-US/library/ms148029(v=vs.80).aspx) |
| Directories | [System.IO.Directory.GetAccessControl](https://msdn.microsoft.com/en-US/library/system.io.directory.getaccesscontrol(v=vs.80).aspx)  [System.IO.Directory.SetAccessControl(System.String,System.Security.AccessControl.DirectorySecurity)](https://msdn.microsoft.com/en-US/library/system.io.directory.setaccesscontrol(v=vs.80).aspx)  [System.IO.DirectoryInfo.GetAccessControl](https://msdn.microsoft.com/en-US/library/system.io.directoryinfo.getaccesscontrol(v=vs.80).aspx)  [System.IO.DirectoryInfo.SetAccessControl(System.Security.AccessControl.DirectorySecurity)](https://msdn.microsoft.com/en-US/library/system.io.directoryinfo.setaccesscontrol(v=vs.80).aspx)  [System.IO.DirectoryInfo.Create(System.Security.AccessControl.DirectorySecurity)](https://msdn.microsoft.com/en-US/library/w6esbt79(v=vs.80).aspx) |
| Event Wait handles | [System.Threading.EventWaitHandle.GetAccessControl](https://msdn.microsoft.com/en-US/library/system.threading.eventwaithandle.getaccesscontrol(v=vs.80).aspx)  [System.Threading.EventWaitHandle.SetAccessControl(System.Security.AccessControl.EventWaitHandleSecurity)](https://msdn.microsoft.com/en-US/library/system.threading.eventwaithandle.setaccesscontrol(v=vs.80).aspx)  [System.Threading.EventWaitHandle.#ctor(System.Boolean,System.Threading.EventResetMode,System.String,System.Boolean,System.Security.AccessControl.EventWaitHandleSecurity)](https://msdn.microsoft.com/en-US/library/z4c9z2kt(v=vs.80).aspx) |
| Files | [System.IO.FileStream.GetAccessControl](https://msdn.microsoft.com/en-US/library/system.io.filestream.getaccesscontrol(v=vs.80).aspx)  [System.IO.FileStream.SetAccessControl(System.Security.AccessControl.FileSecurity)](https://msdn.microsoft.com/en-US/library/system.io.filestream.setaccesscontrol(v=vs.80).aspx)  [System.IO.FileStream.#ctor(System.String,System.IO.FileMode,System.Security.AccessControl.FileSystemRights,System.IO.FileShare,System.Int32,System.IO.FileOptions,System.Security.AccessControl.FileSecurity)](https://msdn.microsoft.com/en-US/library/ms143397(v=vs.80).aspx)  [System.IO.File.GetAccessControl](https://msdn.microsoft.com/en-US/library/system.io.file.getaccesscontrol(v=vs.80).aspx)  [System.IO.File.SetAccessControl(System.String,System.Security.AccessControl.FileSecurity)](https://msdn.microsoft.com/en-US/library/system.io.file.setaccesscontrol(v=vs.80).aspx)  [System.IO.File.Create(System.String,System.Int32,System.IO.FileOptions,System.Security.AccessControl.FileSecurity)](https://msdn.microsoft.com/en-US/library/ms143361(v=vs.80).aspx)  [System.IO.FileInfo.GetAccessControl](https://msdn.microsoft.com/en-US/library/system.io.fileinfo.getaccesscontrol(v=vs.80).aspx)  [System.IO.FileInfo.SetAccessControl(System.Security.AccessControl.FileSecurity)](https://msdn.microsoft.com/en-US/library/system.io.fileinfo.setaccesscontrol(v=vs.80).aspx) |
| Mutexes | [System.Threading.Mutex.GetAccessControl](https://msdn.microsoft.com/en-US/library/system.threading.mutex.getaccesscontrol(v=vs.80).aspx)  [System.Threading.Mutex.SetAccessControl(System.Security.AccessControl.MutexSecurity)](https://msdn.microsoft.com/en-US/library/system.threading.mutex.setaccesscontrol(v=vs.80).aspx)  [System.Threading.Mutex.#ctor(System.Boolean,System.String,System.Boolean,System.Security.AccessControl.MutexSecurity)](https://msdn.microsoft.com/en-US/library/9zf2f5bz(v=vs.80).aspx) |
| Registry keys | [Microsoft.Win32.RegistryKey.GetAccessControl](https://msdn.microsoft.com/en-US/library/microsoft.win32.registrykey.getaccesscontrol(v=vs.80).aspx)  [Microsoft.Win32.RegistryKey.SetAccessControl(System.Security.AccessControl.RegistrySecurity)](https://msdn.microsoft.com/en-US/library/microsoft.win32.registrykey.setaccesscontrol(v=vs.80).aspx)  [Microsoft.Win32.RegistryKey.CreateSubKey(System.String,Microsoft.Win32.RegistryKeyPermissionCheck,System.Security.AccessControl.RegistrySecurity)](https://msdn.microsoft.com/en-US/library/ms128495(v=vs.80).aspx) |
| Semaphores | [System.Threading.Semaphore.GetAccessControl](https://msdn.microsoft.com/en-US/library/system.threading.semaphore.getaccesscontrol(v=vs.80).aspx)  [System.Threading.Semaphore.SetAccessControl(System.Security.AccessControl.SemaphoreSecurity)](https://msdn.microsoft.com/en-US/library/system.threading.semaphore.setaccesscontrol(v=vs.80).aspx)  [System.Threading.Semaphore.#ctor(System.Int32,System.Int32,System.String,System.Boolean,System.Security.AccessControl.SemaphoreSecurity)](https://msdn.microsoft.com/en-US/library/kbk057cx(v=vs.80).aspx) |

### ACL Technology Architecture

**.NET Framework 2.0, 3.0, 3.5, 4, 4.5**

The [System.Security.AccessControl](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol(v=vs.80).aspx) namespace provides access to access control lists (ACLs) through convenient classes that abstract much of the complexity of the Windows ACL security system. Additionally, the **System.Security.AccessControl** namespace contains several classes that provide advanced access to the Windows ACL security system.

The .NET Framework provides access to ACLs for the following resources:

* Cryptographic keys
* Directories
* Event Wait handles
* Files
* Mutexes
* Registry Keys
* Semaphores

Each of these resources has several classes that you can use to create and modify ACLs.

**Hierarchy of ACL classes**

For most scenarios, you can use the higher level, abstracted classes instead of the advanced classes to create and modify ACLs. For each resource, the higher level classes take the following form:

* A class that encapsulates the discretionary access control list (DACL) and the system access control list (SACL). This class takes the name **<***Resource Name***>Security**. For example, the [FileSecurity](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol.filesecurity(v=vs.80).aspx) and [DirectorySecurity](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol.directorysecurity(v=vs.80).aspx) classes encapsulate DACLs and SACLs for files and folders.
* A class that encapsulates an access control entry (ACE). This class takes the name **<***Resource Name***>AccessRule**.
* A class that encapsulates an audit ACE. This class takes the name **<***Resource Name***>AuditRule**.
* Several enumerations that allow you to create specific access and audit rules.
* For a complete list of all high-level ACL classes, see [Technology Summary for ACLs](https://msdn.microsoft.com/en-US/library/ms229936(v=vs.80).aspx).

**Adding ACEs to ACLs**

After you create an ACE using one of the access rule or audit rule classes, you can add the rule to a resource or use it to remove an existing rule from a resource. For example, you might create a rule using the [FileSystemAccessRule](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol.filesystemaccessrule(v=vs.80).aspx) class specifying that only administrators can open a file. You could then add that rule to a **FileSecurity** object or remove a similar rule from a **FileSecurity** object.

Adding an ACE that grants access does not guarantee that a principal will receive access because a deny rule always supersedes an allow rule. For example, if you add an allow access rule for a system account to a file, this does not mean the person will have access because they may also be denied access to the file by another rule.

Each **<***Resource Name***>Security** object associated with a resource provides the following methods to add or remove access rules and audit rules.

|  |  |
| --- | --- |
| **Method** | **Description** |
| **AddAccessRule**  -and-  **AddAuditRule** | Searches for an access or audit rule that can be merged with the new rule. If none are found, adds the new rule. |
| **SetAccessRule** | Removes all access control rules with the same user and [AccessControlType](https://msdn.microsoft.com/en-US/library/w4ds5h86(v=vs.80).aspx) value (**Allow** or **Deny**) as the specified rule, then adds the specified rule. |
| **SetAuditRule** | Removes all audit rules with the same user as the specified rule, regardless of the [AuditFlags](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol.auditflags(v=vs.80).aspx) value, then adds the specified rule. |
| **ResetAccessRule** | Removes all access control rules with the same user as the specified rule, regardless of the**AccessControlType** value, then adds the specified rule. |
| **RemoveAccessRule** | Searches for an access control rule with the same user and **AccessControlType** value (**Allow** or **Deny**) as the specified rule, and with compatible inheritance and propagation flags. If found, the rights contained in the specified access rule are removed from the rule. |
| **RemoveAuditRule** | Searches for an audit control rule with the same user as the specified rule, and with compatible inheritance and propagation flags. If found, the rights contained in the specified rule are removed from the rule. |
| **RemoveAccessRuleAll** | Searches for all access rules with the same user and **AccessControlType** value (**Allow** or **Deny**) as the specified rule and, if found, removes them. |
| **RemoveAuditRuleAll** | Searches for all audit rules with the same user as the specified rule and, if found, removes them. |
| **RemoveAccessRuleSpecific**  -and-  **RemoveAuditRuleSpecific** | Searches for an access or audit rule that exactly matches the specified rule and, if found, removes the rule. |
| **AddAccessRule**  -and-  **AddAuditRule** | Searches for an access or audit rule that can be merged with the new rule. If none are found, adds the new rule. |

**Getting and Setting ACLs**

Each protected resource has methods that get and set the **<***Resource Name***>Security** object associated with a resource. To retrieve the existing ACLs for a specific resource, use one of the **GetAccessControl** methods associated with the resource. To propagate changes back to a resource, use one of the **SetAccesscontrol** methods associated with the resource. Note that changes are not propagated back to a resource until you explicitly re-apply them with one of the set methods.

For a complete list of all of the get and set methods for each protected resource, see [Technology Summary for ACLs](https://msdn.microsoft.com/en-US/library/ms229936(v=vs.80).aspx).

### ACL Technology Scenarios

**.NET Framework 2.0, 3.0, 3.5, 4, 4.5**

The classes in the [System.Security.AccessControl](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol(v=vs.80).aspx) namespace allow you to programmatically create or change discretionary access control lists (DACLs) and system access control lists (SACLs) associated with a protected resource.

This section describes the two most common access control and audit scenarios: setting rules at resource creation and programmatically modifying the rules of an existing resource.

**Scenario 1: Create an access or audit rule for a new file or directory**

Sometimes your application needs to create a new file or folder. This scenario describes how to specify the ACLs for a new file or folder using the managed ACL classes.

**Scenario Key Points**

Create an application that performs the following tasks:

1. Creates one or more [FileSystemAccessRule](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol.filesystemaccessrule(v=vs.80).aspx) or [FileSystemAuditRule](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol.filesystemauditrule(v=vs.80).aspx) objects to represent the rules you want to apply.
2. Adds **FileSystemAccessRule** or **FileSystemAuditRule** objects to a new [FileSecurity](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol.filesecurity(v=vs.80).aspx) or [DirectorySecurity](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol.directorysecurity(v=vs.80).aspx) object.
3. Creates a new file or folder by passing the **FileSecurity** or **DirectorySecurity** object to the appropriate method or constructor. Every file or directory creation method or constructor has an overload that accepts a file security parameter. For example, you can use the[System.IO.File.Create(System.String,System.Int32,System.IO.FileOptions,System.Security.AccessControl.FileSecurity)](https://msdn.microsoft.com/en-US/library/ms143361(v=vs.80).aspx) method, the[System.IO.Directory.CreateDirectory(System.String,System.Security.AccessControl.DirectorySecurity)](https://msdn.microsoft.com/en-US/library/9h4z99zb(v=vs.80).aspx) method, and the[System.IO.FileStream.#ctor(System.String,System.IO.FileMode,System.IO.FileAccess)](https://msdn.microsoft.com/en-US/library/tyhc0kft(v=vs.80).aspx) method in addition to several other methods to set ACLs at creation time.

**Scenario 2: Modify an access or audit rule for an existing file or directory**

Sometimes you need to programmatically change the ACLs of an existing file or folder. This scenario describes how to modify the ACLs for an existing file or folder using the managed ACL classes.

**Scenario Key Points**

Create an application that performs the following tasks:

1. Retrieves the **FileSecurity** or **DirectorySecurity** object from an existing file or folder using the [GetAccessControl](https://msdn.microsoft.com/en-US/library/system.io.file.getaccesscontrol(v=vs.80).aspx) method or the[GetAccessControl](https://msdn.microsoft.com/en-US/library/system.io.directory.getaccesscontrol(v=vs.80).aspx) method.
2. Creates one or more **FileSystemAccessRule** or **FileSystemAuditRule** objects to represent the rules you want to apply.
3. Adds **FileSystemAccessRule** or **FileSystemAuditRule** objects to the **FileSecurity** or **DirectorySecurity** object.
4. Persists the **FileSecurity** or **DirectorySecurity** object using the [SetAccessControl](https://msdn.microsoft.com/en-US/library/system.io.file.setaccesscontrol(v=vs.80).aspx) method or the **GetAccessControl** method.

### ACL Propagation Rules

**.NET Framework 2.0, 3.0, 3.5, 4,4.5**

When you create or modify access control entries (ACEs) for container objects such as folders, you can specify how to propagate the ACEs to objects within the container. For example, you might apply ACEs to all subfolders but not the files within those folders.

The rules of ACE propagation are controlled by different combinations of the [InheritanceFlags](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol.inheritanceflags(v=vs.80).aspx) enumeration and the [PropagationFlags](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol.propagationflags(v=vs.80).aspx)enumeration. You can pass both enumerations to constructors of the [FileSystemAuditRule](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol.filesystemauditrule(v=vs.80).aspx) class or the [FileSystemAccessRule](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol.filesystemaccessrule(v=vs.80).aspx) class.

The following table shows all combinations of the two enumerations and describes how each combination affects the rules of propagation.

|  |  |
| --- | --- |
| **Flag combinations** | **Propagation results** |
| No Flags | Target folder. |
| [ObjectInherit](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol.inheritanceflags(v=vs.80).aspx) | Target folder, child object (file), grandchild object (file). |
| **ObjectInherit** and [NoPropagateInherit](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol.propagationflags(v=vs.80).aspx) | Target folder, child object (file). |
| **ObjectInherit** and [InheritOnly](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol.propagationflags(v=vs.80).aspx) | Child object (file), grandchild object (file). |
| **ObjectInherit**, **InheritOnly**, and **NoPropagateInherit** | Child object (file). |
| [ContainerInherit](https://msdn.microsoft.com/en-US/library/system.security.accesscontrol.inheritanceflags(v=vs.80).aspx) | Target folder, child folder, grandchild folder. |
| **ContainerInherit**, and **NoPropagateInherit** | Target folder, child folder. |
| **ContainerInherit**, and **InheritOnly** | Child folder, grandchild folder. |
| **ContainerInherit**, **InheritOnly**, and **NoPropagateInherit** | Child folder. |
| **ContainerInherit**, and **ObjectInherit** | Target folder, child folder, child object (file), grandchild folder, grandchild object (file). |
| **ContainerInherit**, **ObjectInherit**, and **NoPropagateInherit** | Target folder, child folder, child object (file). |
| **ContainerInherit**, **ObjectInherit**, and **InheritOnly** | Child folder, child object (file), grandchild folder, grandchild object (file). |
| **ContainerInherit**, **ObjectInherit**, **NoPropagateInherit**,**InheritOnly** | Child folder, child object (file). |

**Note**   To change the access rules of only certain child files or folders, you must break your operation into several different calls.

## About System.Security.Cryptography.Pkcs

**.NET Framework 2.0, 3.0**

[System.Security.Cryptography.Pkcs](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs(v=vs.85).aspx) is the namespace that contains the managed code implementation of the Cryptographic Message Syntax (CMS) and Public-Key Cryptography Standards #7 (PKCS #7) standards. CMS is a superset of PKCS #7. This documentation refers to the Microsoft implementation of this combination of technologies as *CMS/PKCS #7* to emphasize that there are dual standards implemented in this namespace.

For more information about CMS, see Cryptographic Message Syntax (CMS) (<http://www.ietf.org/rfc/rfc3369.txt>).

For more information about PKCS #7, see PKCS #7: Cryptographic Message Syntax Standard (<http://www.rsasecurity.com/rsalabs/node.asp?id=2129>).

**In This Section**

The following topics provide information about managed CMS/PKCS #7.

[Security Services Provided by CMS/PKCS #7](https://msdn.microsoft.com/en-us/library/ms180946(v=vs.85).aspx)

Provides the ability to implement several security services into an application.

[Algorithm Support in CMS/PKCS #7](https://msdn.microsoft.com/en-us/library/ms180947(v=vs.85).aspx)

Supports many cryptographic algorithms.

[Components of a CMS/PKCS #7 Message](https://msdn.microsoft.com/en-us/library/ms180949(v=vs.85).aspx)

Facilitates the implementation of multiple security services into a message. Attributes can also be associated with the CMS/PKCS #7 message, which yield additional information about the message.

[Types of CMS/PKCS #7 Messages](https://msdn.microsoft.com/en-us/library/ms180948(v=vs.85).aspx)

Provides implementations of several protected message types.

### Security Services Provided by CMS/PKCS #7

**.NET Framework 2.0,3.0**

CMS/PKCS #7 provides the ability to implement several *security services* into an application. A security service is some type of data protection, and is independent of the mechanism or cryptographic algorithm by which it is implemented.

Data encryption is provided through *digital envelopes*. This security service provides a high probability that only an intended recipient can read that message. The digital envelope *encrypts* a message for a set of recipients. The main class that provides digital enveloping is the[EnvelopedCms](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.envelopedcms(v=vs.85).aspx) class. For more information, see [EnvelopedCms Message](https://msdn.microsoft.com/en-us/library/ms180951(v=vs.85).aspx).

Entity *authentication* and data *integrity* are provided through *digital signatures*. These security services provide a high probability that an entity claiming to be the author or sender of a message is indeed that entity, and that the data has not been modified since it was signed. The entity can be a person, a software application, or any other source with a unique identity.

A set of signers can digitally sign a message. In addition, each signature can have a set of *countersignatures*. The main class that provides digital signing is the [SignedCms](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.signedcms(v=vs.85).aspx) class. For more information, see [SignedCms Message](https://msdn.microsoft.com/en-us/library/ms180950(v=vs.85).aspx).

The previously mentioned security services can be combined to give entity authentication, data integrity, and data confidentiality. This is done by both digitally signing and digitally enveloping a message. Although these mechanisms may be applied in either order, it might be advantageous to first sign a message and then envelope it. In that case, you can justify that the signed data was intelligible because it was not yet encrypted. For more information, see [Enveloped and Signed CMS/PKCS #7 Message](https://msdn.microsoft.com/en-us/library/ms180952(v=vs.85).aspx).

Both digitally signed and digitally enveloped messages can also carry *attributes*. Attributes effectively extend the security services that CMS/PKCS #7 provides. For example, the time that a message was digitally signed can be included in a **SignedCms** message. Including signing time might help to satisfy a requirement to implement *nonrepudiation*. It is a security service that helps prevent the author or sender of a message from later denying writing or sending it. For more background information about attributes, see [Components of a CMS/PKCS #7 Message](https://msdn.microsoft.com/en-us/library/ms180949(v=vs.85).aspx) .

### Algorithm Support in CMS/PKCS #7

**.NET Framework 2.0, 3.0**

CMS/PKCS #7 supports many cryptographic algorithms.

CMS/PKCS #7 specifies the implementation of security services. A security service can almost always be implemented by any of several different cryptographic algorithms. For example, data confidentiality is a security service that can be implemented by the Triple DES encryption algorithm, the AES encryption algorithm, or by a number of other encryption algorithms.

The algorithms available to you depend on the *cryptographic service providers* (CSP) that are installed on your computer. Each CSP contains the implementation of a specific set of cryptographic algorithms. For applications that transmit data, the systems that receive messages that are protected by CMS/PKCS #7 need to have CSPs installed that support the same algorithms that the sender used. For more information, see "Cryptographic Service Providers" on the MSDN Library site ([http://msdn.microsoft.com/library](https://msdn.microsoft.com/library)).

### Types of CMS/PKCS #7 Messages

**.NET Framework 2.0, 3.0**

CMS/PKCS #7 provides implementations of several protected message types. These message types are known as *content types* in the standards that define CMS/PKCS #7.

**In This Section**

The following topics provide information about CMS/PKCS #7 protected message types.

[SignedCms Message](https://msdn.microsoft.com/en-us/library/ms180950(v=vs.85).aspx)

Provides the [SignedCms](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.signedcms(v=vs.85).aspx) class to digitally sign messages.

[EnvelopedCms Message](https://msdn.microsoft.com/en-us/library/ms180951(v=vs.85).aspx)

Provides the [EnvelopedCms](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.envelopedcms(v=vs.85).aspx) class to digitally envelope messages.

[Enveloped and Signed CMS/PKCS #7 Message](https://msdn.microsoft.com/en-us/library/ms180952(v=vs.85).aspx)

Provides the capability of both signing and enveloping a message because protected messages can be nested.

#### SignedCms Message

**.NET Framework 2.0, 3.0**

CMS/PKCS #7 provides the [SignedCms](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.signedcms(v=vs.85).aspx) class to digitally sign messages.

Digitally signing a message protects it by applying security services known as entity *authentication* and data *integrity*. Entity authentication gives a high probability that an entity claiming to be the author or sender of a message is indeed that entity. The entity can be a person, a software application, or any other source with a unique identity.

Data integrity gives a high probability that the message itself was not altered in any way, including by deletion or duplication.

Use the **SignedCms** class to apply either or both of these security services to a message. Use this class if the goal is to verify with high probability the authenticity of a message sender or author, or that the message has not been modified while in storage or transit, or both.

E-mail, using the S/MIME security standard, is an example of how you can use the **SignedCms** class to provide security. In addition to other security services, S/MIME specifies the ability to verify the authenticity of the sender of an e-mail message as well as the checking the integrity of the message itself.

Multiple signatures can be applied to a message. An example of an application that may need this capability is a document authoring and reading application. With the support of multiple signatures by CMS/PKCS #7, the application can allow multiple document authors to each sign the document they took part in writing. It can also allow the document reader to verify that the claimed authors are the actual authors of a document, and that the document has not been modified since it was signed.

Use one of the ComputeSignature methods of the **SignedCms** class to compute a message signature. The[System.Security.Cryptography.Pkcs.SignedCms.ComputeSignature(System.Security.Cryptography.Pkcs.CmsSigner)](https://msdn.microsoft.com/en-us/library/8412wc31(v=vs.85).aspx) method requires the application to first set up the characteristics of the message signer by constructing a [CmsSigner](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.cmssigner(v=vs.85).aspx) object. The **CmsSigner** class stores the signer's X509 certificate in addition to other properties. The [System.Security.Cryptography.Pkcs.SignedCms.ComputeSignature](https://msdn.microsoft.com/en-us/library/1wd3e26b(v=vs.85).aspx) method presents a dialog box so the user can select the appropriate signer's certificate.

A signer's certificate chain can be included in whole or in part within the [System.Security.Cryptography.Pkcs.CmsSigner.Certificates](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.cmssigner.certificates(v=vs.85).aspx) property. Set the [System.Security.Cryptography.Pkcs.CmsSigner.IncludeOption](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.cmssigner.includeoption(v=vs.85).aspx) property to affect how much of the certificate chain is included.

A **SignedCms** message can either be *nondetached* or *detached*. The Boolean property[System.Security.Cryptography.Pkcs.SignedCms.Detached](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.signedcms.detached(v=vs.85).aspx) determines whether the message is detached. A nondetached **SignedCms** message contains the message that was signed. A detached **SignedCms** message does not contain the message that was signed, but contains all other properties of the message, such as signatures and attributes. An example of an application in which a detached **SignedCms** message might be used is a document-reading application. In this case, it might be undesirable to duplicate the storage of a large signed document by having it stored in the SignedCms message in addition to a primary storage location. Applications that sign large contents are most likely to use detached **SignedCms** messages.

CMS/PKCS #7 supports *countersignatures*. A countersignature is a digital signature of another digital signature. As such, it only provides authenticity of the signature, and not of the message's content. A digital signature can be countersigned by multiple countersigners. However, CMS/PKCS #7 supports only one level of countersignature. A countersignature itself cannot be countersigned. An example of an application that might use countersignatures is a digital notary service.

Use one of the ComputeCounterSignature methods of the [SignerInfo](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.signerinfo(v=vs.85).aspx) class to compute a countersignature. The[System.Security.Cryptography.Pkcs.SignedCms.SignerInfos](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.signedcms.signerinfos(v=vs.85).aspx) property is a collection of **SignerInfo** objects that result from signing a message. The [ComputeCounterSignature](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.signerinfo.computecountersignature(v=vs.85).aspx) methods work analogous to that of the [ComputeSignature](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.signedcms.computesignature(v=vs.85).aspx) methods of the **SignedCms** class.

A **SignedCms** message can have associated signature-specific attributes. These attributes can be signed or unsigned.

Signed attributes are placed in the message by setting them in the appropriate signer's[System.Security.Cryptography.Pkcs.CmsSigner.SignedAttributes](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.cmssigner.signedattributes(v=vs.85).aspx) property. When a signature is computed for that signer, the signed attributes are signed along with the inner content in the [System.Security.Cryptography.Pkcs.SignedCms.ContentInfo](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.signedcms.contentinfo(v=vs.85).aspx) property. The signed attributes are available after the message has been signed in the [System.Security.Cryptography.Pkcs.SignerInfo.SignedAttributes](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.signerinfo.signedattributes(v=vs.85).aspx) property for the applicable signer from the **System.Security.Cryptography.Pkcs.SignedCms.SignerInfos** property. An example of a signed attribute that might be useful is the [Pkcs9SigningTime](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.pkcs9signingtime(v=vs.85).aspx) signing-time attribute. This attribute contains the time that the message was signed.

Unsigned attributes are placed in the message by setting them in the appropriate signer's[System.Security.Cryptography.Pkcs.CmsSigner.UnsignedAttributes](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.cmssigner.unsignedattributes(v=vs.85).aspx) property. These attributes are not signed—it cannot be assumed that they have authenticity or integrity. The unsigned attributes are available after the message has been signed in the[System.Security.Cryptography.Pkcs.SignerInfo.UnsignedAttributes](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.signerinfo.unsignedattributes(v=vs.85).aspx) property for the applicable signer from the**System.Security.Cryptography.Pkcs.SignedCms.SignerInfos** property. An example of an unsigned attribute that is used internally by CMS/PKCS #7 is a countersignature. Because it is already a type of signature, it does not need to be signed again. Another example of an attribute that might remain unsigned is a document description, available in the [Pkcs9DocumentDescription](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.pkcs9documentdescription(v=vs.85).aspx) class.

Use one of the CheckSignature methods of the **SignedCms** class to verify the message signatures, countersignatures, and signed attributes. The **SignedCms** message contains the signers' certificates that are necessary for the verification. The signature verification can either validate the signers' certificates or not, as determined by the value of the *verifySignatureOnly* parameter to these methods.

To verify a detached **SignedCms** message, first associate the content of the message with the **SignedCms** message. Do so by constructing a[ContentInfo](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.contentinfo(v=vs.85).aspx) object with the message content. Use that to construct a **SignedCms** object by using, for example, the [SignedCms](https://msdn.microsoft.com/en-us/library/ca076kek(v=vs.85).aspx) constructor. Set the second parameter to **true** to indicate that the message is detached. Decode the encoded **SignedCms** message to be verified by using the [Decode](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.signedcms.decode(v=vs.85).aspx) method. Finally, check the signature as previously described.

For several code examples that use **SignedCms** messages, see [Using System.Security.Cryptography.Pkcs](https://msdn.microsoft.com/en-us/library/ms180955(v=vs.85).aspx).

#### EnvelopedCms Message

**.NET Framework 2.0,3.0**

CMS/PKCS #7 provides the [EnvelopedCms](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.envelopedcms(v=vs.85).aspx) class to digitally envelope messages.

*Digitally enveloping* a message protects it by applying a security service known as *data confidentiality*.

The digital envelope uses *encryption* to help keep the message confidential. A *session key* is generated and used to encrypt the message. The session key is itself then encrypted by using the public key of the recipient. The combination of the encrypted message and the encrypted session key constitutes the digital envelope. Upon receipt, the session key is decrypted by using the private key of the recipient. The message is then decrypted by using the session key. The session key can be used to both encrypt and decrypt the message because it is a *symmetric key*.

Use the **EnvelopedCms** class in any application that has the requirement to maximize the confidentiality of the data. E-mail, using the S/MIME security standard, is an example of how you can use the **EnvelopedCms** class to provide security. In addition to other security services, S/MIME specifies the ability to encrypt an e-mail message for multiple recipients. Digital envelopes are tailored to meet that need.

Use one of the Encrypt methods of the **EnvelopedCms** class to encrypt a message. The[System.Security.Cryptography.Pkcs.EnvelopedCms.Encrypt(System.Security.Cryptography.Pkcs.CmsRecipient)](https://msdn.microsoft.com/en-us/library/882kkwk6(v=vs.85).aspx) method requires the application to first set up the characteristics of the message recipient by constructing a [CmsRecipient](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.cmsrecipient(v=vs.85).aspx) object. The **CmsRecipient** class stores the recipient's X509 certificate and the technique by which a session key is established between the message sender and recipient. The[System.Security.Cryptography.Pkcs.EnvelopedCms.Encrypt](https://msdn.microsoft.com/en-us/library/7z52d2za(v=vs.85).aspx) method presents a dialog box so the user can select the appropriate recipient's certificate.

The **EnvelopedCms** class supports enveloping a message for multiple recipients. Set up the group of recipients in a [CmsRecipientCollection](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.cmsrecipientcollection(v=vs.85).aspx)collection to use as input to the[System.Security.Cryptography.Pkcs.EnvelopedCms.Encrypt(System.Security.Cryptography.Pkcs.CmsRecipientCollection)](https://msdn.microsoft.com/en-us/library/3hc1y2b3(v=vs.85).aspx) method.

There are two techniques for establishing the session key between the sender and the recipient of a message. *Key transport* algorithms typically use the RSA (<http://www.rsasecurity.com/rsalabs/node.asp?id=2125>) algorithm, in which an originator establishes a shared cryptographic key with a recipient by generating that key and then transporting it to the recipient. *Key agreement* algorithms typically use the Diffie-Hellman key agreement (<http://www.rsasecurity.com/rsalabs/node.asp?id=2126>) algorithm. In it, two parties establish a shared cryptographic key by both taking part in its generation and, by definition, agree on that key.

Information for each recipient of an **EnvelopedCms** message is available in the[System.Security.Cryptography.Pkcs.EnvelopedCms.RecipientInfos](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.envelopedcms.recipientinfos(v=vs.85).aspx) property. Information about a recipient that establishes the session key with the sender by means of a key transport algorithm is found in a [KeyTransRecipientInfo](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.keytransrecipientinfo(v=vs.85).aspx) object. Information about a recipient that establishes the session key with the sender by means of a key agreement algorithm is found in a [KeyAgreeRecipientInfo](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.keyagreerecipientinfo(v=vs.85).aspx) object.

An **EnvelopedCms** message can include unprotected attributes. These are attributes that are not encrypted; they do not have data confidentiality. These attributes are stored in the [System.Security.Cryptography.Pkcs.EnvelopedCms.UnprotectedAttributes](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.envelopedcms.unprotectedattributes(v=vs.85).aspx) property.

Use one of the Decrypt methods of the **EnvelopedCms** class to decrypt the enveloped message. The **EnvelopedCms** message contains the recipients' identifying information that is necessary for the decryption. That information is contained in the **RecipientInfos** property.

For several code examples that use **EnvelopedCms** messages, see [Using System.Security.Cryptography.Pkcs](https://msdn.microsoft.com/en-us/library/ms180955(v=vs.85).aspx).

#### Enveloped and Signed CMS/PKCS #7 Message

**.NET Framework 2.0, 3.0**

CMS/PKCS #7 provides the capability to both sign and envelope a message. This is possible because protected messages can be nested. Digitally signing and enveloping a message protects it by applying all the security services separately described in the [SignedCms Message](https://msdn.microsoft.com/en-us/library/ms180950(v=vs.85).aspx)and [EnvelopedCms Message](https://msdn.microsoft.com/en-us/library/ms180951(v=vs.85).aspx) topics.

E-mail, using the S/MIME security standard, is an example of an application in which both signing and enveloping a message is useful. S/MIME specifies the ability to both sign and encrypt an e-mail message. If your application requires a high probability of both data authentication and data confidentiality, use a combination of digital signing and digital enveloping.

Although these security services can be applied in either order, it might be advantageous to first sign a message and then envelope it. As such, the data that was signed was intelligible because it was not yet encrypted.

Sign and envelope a message by using the [SignedCms](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.signedcms(v=vs.85).aspx) and [EnvelopedCms](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.envelopedcms(v=vs.85).aspx) classes in conjunction with one another. For example, application requirements dictate first signing a message and then enveloping it. Sign the message using the **SignedCms** class as discussed in the topic[SignedCms Message](https://msdn.microsoft.com/en-us/library/ms180950(v=vs.85).aspx). Encode the **SignedCms** message by invoking the [System.Security.Cryptography.Pkcs.SignedCms.Encode](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.signedcms.encode(v=vs.85).aspx) method. This yields the encoding as a byte array. Use that byte array as the input to construct a [ContentInfo](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.contentinfo(v=vs.85).aspx) object. Use the **ContentInfo** object as input to construct an **EnvelopedCms** object.

The **EnvelopedCms** object now has a **SignedCms** object nested inside it as its inner content. Now, envelope the message as described in[EnvelopedCms Message](https://msdn.microsoft.com/en-us/library/ms180951(v=vs.85).aspx).

For a code example that signs and envelopes a message, see the [How to: Sign and Envelop a Message](https://msdn.microsoft.com/en-us/library/ms180961(v=vs.85).aspx) topic.

#### Components of a CMS/PKCS #7 Message

**.NET Framework 2.0, 3.0**

The CMS/PKCS #7 message structure facilitates the implementation of one or multiple security services into a message. Attributes can also be associated with the CMS/PKCS #7 message, which yield additional information about the message.

**Data to Be Protected**

CMS/PKCS #7 gains the capability of applying multiple security services to data by storing it in a [ContentInfo](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.contentinfo(v=vs.85).aspx) object. This object is a property of the two primary classes that protect data in the [System.Security.Cryptography.Pkcs](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs(v=vs.85).aspx) namespace, [SignedCms](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.signedcms(v=vs.85).aspx) and [EnvelopedCms](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.envelopedcms(v=vs.85).aspx). The**ContentInfo** object might store unprotected data, but it might also encapsulate data that was previously protected by a CMS/PKCS #7 security service. This encapsulation allows nested layers of security to be applied to data. For example, data that has already been digitally signed can then be digitally enveloped.

The data being supplied with a new layer of protection in a CMS/PKCS #7 message, which is in the **ContentInfo** object, is referred to as the*inner content* of the CMS/PKCS #7 message. The data that results from applying that protection is referred to as the *outer content* of the CMS/PKCS #7 message. The *outer content* is obtained by invoking the Encode method of the applicable CMS/PKCS #7 message object. In practice, this means invoking [System.Security.Cryptography.Pkcs.SignedCms.Encode](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.signedcms.encode(v=vs.85).aspx) or[System.Security.Cryptography.Pkcs.EnvelopedCms.Encode](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.envelopedcms.encode(v=vs.85).aspx).

For more information about data that is protected by multiple security services, see [Enveloped and Signed CMS/PKCS #7 Message](https://msdn.microsoft.com/en-us/library/ms180952(v=vs.85).aspx). For a code example of data protected by multiple security services, see [How to: Sign and Envelop a Message](https://msdn.microsoft.com/en-us/library/ms180961(v=vs.85).aspx).

**Attributes**

CMS/PKCS #7 enables you to place *attributes* in protected messages. An attribute consists of an object identifier ([Oid](https://msdn.microsoft.com/en-us/library/system.security.cryptography.oid(v=vs.85).aspx)) that identifies what kind of attribute it is, and data that holds the value of the attribute. The choice, location, and characteristics of these attributes depend on the type of CMS/PKCS #7 message.

A **SignedCms** message can carry two types of signature-specific attributes: those that are signed along with the message content, and those that remain unsigned. For more information, see [SignedCms Message](https://msdn.microsoft.com/en-us/library/ms180950(v=vs.85).aspx). An **EnvelopedCms** message can carry attributes that remain unencrypted. For more information, see [EnvelopedCms Message](https://msdn.microsoft.com/en-us/library/ms180951(v=vs.85).aspx).

The following attribute classes are included in the **System.Security.Cryptography.Pkcs** namespace. Some are required by the CMS/PKCS #7 standards, and others are attributes useful in many application scenarios:

* [Pkcs9ContentType](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.pkcs9contenttype(v=vs.85).aspx)
* [Pkcs9DocumentDescription](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.pkcs9documentdescription(v=vs.85).aspx)
* [Pkcs9DocumentName](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.pkcs9documentname(v=vs.85).aspx)
* [Pkcs9MessageDigest](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.pkcs9messagedigest(v=vs.85).aspx)
* [Pkcs9SigningTime](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.pkcs9signingtime(v=vs.85).aspx)

Custom attributes can also be fashioned to suit the needs of an application. Derive a custom attribute class from [Pkcs9AttributeObject](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.pkcs9attributeobject(v=vs.85).aspx).

**Subjects**

A *subject* in a CMS/PKCS #7 message is an entity involved in producing or consuming the message. Depending on the behavior of an application, a subject may play any of a number of different roles. Examples of such roles include sender, author, signer, countersigner, recipient, or reader of a message, or some combination thereof. The subject can be a person, a software application, or any other entity that has a unique identity. The two main classes that embody subjects in **System.Security.Cryptography.Pkcs** are [CmsSigner](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.cmssigner(v=vs.85).aspx) and [CmsRecipient](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.cmsrecipient(v=vs.85).aspx).

Subjects in CMS/PKCS #7 use cryptographic credentials to perform their operations. These are usually in the form of a public key certificate with an associated private key. A subject is identified and located by its public key certificate. That certificate can be uniquely identified by one of two ways.

* The distinguished name of the issuer along with an issuer-specific certificate serial number
* A subject key identifier

These mechanisms of identification are represented in the [SubjectIdentifierType](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.subjectidentifiertype(v=vs.85).aspx) enumeration. Some of the constructors used in the**CmsSigner** and **CmsRecipient** classes take a **SubjectIdentifierType** member as input.

## Using System.Security.Cryptography.Pkcs

**.NET Framework 2.0, 3.0**

You can use [System.Security.Cryptography.Pkcs](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs(v=vs.85).aspx) to program cryptography into your application. This namespace implements the Cryptographic Message Syntax standard and PKCS #7. Using this standard, you can implement signed messages, enveloped messages, or signed and enveloped messages.

**In This Section**

The following topics provide information about **System.Security.Cryptography.Pkcs** namespace.

[How to: Sign Messages by One Signer](https://msdn.microsoft.com/en-us/library/ms180956(v=vs.85).aspx)

Creates a CMS/PKCS #7 signed message. The message is signed by a single signer.

[How to: Sign a Message by Multiple Signers](https://msdn.microsoft.com/en-us/library/ms180957(v=vs.85).aspx)

Creates a CMS/PKCS #7 signed message. The message is signed by multiple signers.

[How to: Countersign a Message](https://msdn.microsoft.com/en-us/library/ms180958(v=vs.85).aspx)

Creates a CMS/PKCS #7 signed message. The message is signed by a single signer. That signature is then countersigned by two other signers.

[How to: Envelope a Message for One Recipient](https://msdn.microsoft.com/en-us/library/ms180959(v=vs.85).aspx)

Creates a CMS/PKCS #7 enveloped message. The message is encrypted for a single recipient.

[How to: Envelope a Message for Multiple Recipients](https://msdn.microsoft.com/en-us/library/ms180960(v=vs.85).aspx)

Creates a CMS/PKCS #7 enveloped message. The message is encrypted for multiple recipients.

[How to: Sign and Envelop a Message](https://msdn.microsoft.com/en-us/library/ms180961(v=vs.85).aspx)

Creates a CMS/PKCS #7 enveloped signed message. The message is first signed by a single signer and is then encrypted for a single recipient.

[Supporting Tasks for Using System.Security.Cryptography.Pkcs](https://msdn.microsoft.com/en-us/library/ms180962(v=vs.85).aspx)

Contains additional programming tasks.

### How to: Sign Messages by One Signer

**.NET Framework 2.0, 3.0**

This example creates a CMS/PKCS #7 signed message by using [System.Security.Cryptography.Pkcs](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs(v=vs.85).aspx). The message is signed by a single signer. The signature on that message is then verified.

**Example**

This example uses the following classes:

* [CmsSigner](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.cmssigner(v=vs.85).aspx)
* [ContentInfo](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.contentinfo(v=vs.85).aspx)
* [SignedCms](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.signedcms(v=vs.85).aspx)
* [X509Certificate2](https://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.x509certificate2(v=vs.85).aspx)
* [X509Certificate2Collection](https://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.x509certificate2collection(v=vs.85).aspx)
* [X509Store](https://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.x509store(v=vs.85).aspx)

The following example requires that a public key certificate with the subject name "MessageSigner1" be contained in the My certificate store, and that the associated private key exists.

|  |
| --- |
| **NoteNote** |
| This example is only for illustrative purposes. Production environments might use a different model in which the sender and the recipient of the message execute in different processes with their unique public key credentials. |

Set up this example by using the Makecert.exe utility, one of several ways to do so. [Certificate Creation Tool (Makecert.exe)](https://msdn.microsoft.com/en-us/library/bfsktky3(v=vs.85).aspx) is a convenient utility for generating test certificates. In a production environment, certificates are generated by a certification authority.

The following **Makecert** command generates the required public key certificates and private keys.

**Makecert -n "CN=MessageSigner1" -ss My**

C#

// Copyright (c) Microsoft Corporation. All rights reserved.

#region Using directives

using System;

using System.Security.Cryptography.Pkcs;

using System.Security.Cryptography.X509Certificates;

using System.Text;

#endregion

namespace SigningAMessageByOneSigner

{

class SignedCmsSingleSigner

{

const String signerName = "MessageSigner1";

static void Main(string[] args)

{

Console.WriteLine("System.Security.Cryptography.Pkcs " +

"Sample: Single-signer signed and verified message");

// Original message.

const String msg = "This is the message to be signed.";

Console.WriteLine("\nOriginal message (len {0}): {1} ",

msg.Length, msg);

// Convert message to array of bytes for signing.

Encoding unicode = Encoding.Unicode;

byte[] msgBytes = unicode.GetBytes(msg);

Console.WriteLine("\n\n------------------------------");

Console.WriteLine(" SETUP OF CREDENTIALS ");

Console.WriteLine("------------------------------\n");

X509Certificate2 signerCert = GetSignerCert();

Console.WriteLine("\n\n----------------------");

Console.WriteLine(" SENDER SIDE ");

Console.WriteLine("----------------------\n");

byte[] encodedSignedCms = SignMsg(msgBytes, signerCert);

Console.WriteLine("\n\n------------------------");

Console.WriteLine(" RECIPIENT SIDE ");

Console.WriteLine("------------------------\n");

if (VerifyMsg(encodedSignedCms))

{

Console.WriteLine("\nMessage verified");

}

else

{

Console.WriteLine("\nMessage failed to verify");

}

}

// Open the My (or Personal) certificate store and search for

// credentials to sign the message with. The certificate

// must have the subject name "MessageSigner1".

static public X509Certificate2 GetSignerCert()

{

// Open the My certificate store.

X509Store storeMy = new X509Store(StoreName.My,

StoreLocation.CurrentUser);

storeMy.Open(OpenFlags.ReadOnly);

// Display certificates to help troubleshoot

// the example's setup.

Console.WriteLine("Found certs with the following subject " +

"names in the {0} store:", storeMy.Name);

foreach (X509Certificate2 cert in storeMy.Certificates)

{

Console.WriteLine("\t{0}", cert.SubjectName.Name);

}

// Find the signer's certificate.

X509Certificate2Collection certColl =

storeMy.Certificates.Find(X509FindType.FindBySubjectName,

signerName, false);

Console.WriteLine(

"Found {0} certificates in the {1} store with name {2}",

certColl.Count, storeMy.Name, signerName);

// Check to see if the certificate suggested by the example

// requirements is not present.

if (certColl.Count == 0)

{

Console.WriteLine(

"A suggested certificate to use for this example " +

"is not in the certificate store. Select " +

"an alternate certificate to use for " +

"signing the message.");

}

storeMy.Close();

// If more than one matching cert, return the first one.

return certColl[0];

}

// Sign the message with the private key of the signer.

static public byte[] SignMsg(

Byte[] msg,

X509Certificate2 signerCert)

{

// Place message in a ContentInfo object.

// This is required to build a SignedCms object.

ContentInfo contentInfo = new ContentInfo(msg);

// Instantiate SignedCms object with the ContentInfo above.

// Has default SubjectIdentifierType IssuerAndSerialNumber.

// Has default Detached property value false, so message is

// included in the encoded SignedCms.

SignedCms signedCms = new SignedCms(contentInfo);

// Formulate a CmsSigner object for the signer.

CmsSigner cmsSigner = new CmsSigner(signerCert);

// Sign the CMS/PKCS #7 message.

Console.Write("Computing signature with signer subject " +

"name {0} ... ", signerCert.SubjectName.Name);

signedCms.ComputeSignature(cmsSigner);

Console.WriteLine("Done.");

// Encode the CMS/PKCS #7 message.

return signedCms.Encode();

}

// Verify the encoded SignedCms message and return a Boolean

// value that specifies whether the verification was successful.

static public bool VerifyMsg(byte[] encodedSignedCms)

{

// Prepare an object in which to decode and verify.

SignedCms signedCms = new SignedCms();

signedCms.Decode(encodedSignedCms);

// Catch a verification exception if you want to

// advise the message recipient that

// security actions might be appropriate.

try

{

// Verify signature. Do not validate signer

// certificate for the purposes of this example.

// Note that in a production environment, validating

// the signer certificate chain will probably

// be necessary.

Console.Write("Checking signature on message ... ");

signedCms.CheckSignature(true);

Console.WriteLine("Done.");

}

catch (System.Security.Cryptography.CryptographicException e)

{

Console.WriteLine("VerifyMsg caught exception: {0}",

e.Message);

Console.WriteLine("Verification of the signed PKCS #7 " +

"failed. The message, signatures, or " +

"countersignatures may have been modified " +

"in transit or storage. The message signers or " +

"countersigners may not be who they claim to be. " +

"The message's authenticity or integrity, " +

"or both, are not guaranteed.");

return false;

}

return true;

}

}

}

### How to: Sign a Message by Multiple Signers

**.NET Framework 2.0, 3.0**

This example creates a CMS/PKCS #7 signed message by using [System.Security.Cryptography.Pkcs](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs(v=vs.85).aspx). The message is signed by multiple signers. The signatures on that message are then verified.

**Example**

This example uses the following classes:

* [CmsSigner](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.cmssigner(v=vs.85).aspx)
* [ContentInfo](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.contentinfo(v=vs.85).aspx)
* [SignedCms](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.signedcms(v=vs.85).aspx)
* [X509Certificate2](https://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.x509certificate2(v=vs.85).aspx)
* [X509Certificate2Collection](https://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.x509certificate2collection(v=vs.85).aspx)
* [X509Store](https://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.x509store(v=vs.85).aspx)

The following example begins by using a previously generated encoded **SignedCms** message (signed by a signer with the subject name "MessageSigner1"), and then signs it a second time by a signer with the subject name "MessageSigner2". This example requires that a public key certificate with the subject name "MessageSigner2" be contained in the My certificate store, and that is has an associated private key.

|  |
| --- |
| **NoteNote** |
| This example is only for illustrative purposes. Production environments might use a different model in which the sender and the recipient of the message execute in different processes with their unique public key credentials. |

Set up this example by using the Makecert.exe utility, one of several ways to do so. [Certificate Creation Tool (Makecert.exe)](https://msdn.microsoft.com/en-us/library/bfsktky3(v=vs.85).aspx) is a convenient utility for generating test certificates. In a production environment, certificates are generated by a certification authority.

The following **Makecert** command generates the required public key certificates and private keys.

**Makecert -n "CN=MessageSigner2" -ss My**

C#

// Copyright (c) Microsoft Corporation. All rights reserved.

#region Using directives

using System;

using System.Security.Cryptography.Pkcs;

using System.Security.Cryptography.X509Certificates;

using System.Text;

#endregion

namespace SigningAMessageByMultipleSigners

{

class SignedCmsMultipleSigners

{

// Subject name of the second signer of the message.

const String signerName = "MessageSigner2";

static void Main(string[] args)

{

// An encoded SignedCms message that was generated

// previously. It is the message "The Board of Directors

// hereby adopts the resolution." signed by one signer

// with the subject name "MessageSigner1". This example

// adds a second signature to this encoded

// SignedCms message.

Byte[] signedMessage = {

0x30, 0x82, 0x03, 0x34, 0x06, 0x09, 0x2a, 0x86, 0x48, 0x86,

0xf7, 0x0d, 0x01, 0x07, 0x02, 0xa0, 0x82, 0x03, 0x25, 0x30,

0x82, 0x03, 0x21, 0x02, 0x01, 0x01, 0x31, 0x0b, 0x30, 0x09,

0x06, 0x05, 0x2b, 0x0e, 0x03, 0x02, 0x1a, 0x05, 0x00, 0x30,

0x77, 0x06, 0x09, 0x2a, 0x86, 0x48, 0x86, 0xf7, 0x0d, 0x01,

0x07, 0x01, 0xa0, 0x6a, 0x04, 0x68, 0x54, 0x00, 0x68, 0x00,

0x65, 0x00, 0x20, 0x00, 0x42, 0x00, 0x6f, 0x00, 0x61, 0x00,

0x72, 0x00, 0x64, 0x00, 0x20, 0x00, 0x6f, 0x00, 0x66, 0x00,

0x20, 0x00, 0x44, 0x00, 0x69, 0x00, 0x72, 0x00, 0x65, 0x00,

0x63, 0x00, 0x74, 0x00, 0x6f, 0x00, 0x72, 0x00, 0x73, 0x00,

0x20, 0x00, 0x68, 0x00, 0x65, 0x00, 0x72, 0x00, 0x65, 0x00,

0x62, 0x00, 0x79, 0x00, 0x20, 0x00, 0x61, 0x00, 0x64, 0x00,

0x6f, 0x00, 0x70, 0x00, 0x74, 0x00, 0x73, 0x00, 0x20, 0x00,

0x74, 0x00, 0x68, 0x00, 0x65, 0x00, 0x20, 0x00, 0x72, 0x00,

0x65, 0x00, 0x73, 0x00, 0x6f, 0x00, 0x6c, 0x00, 0x75, 0x00,

0x74, 0x00, 0x69, 0x00, 0x6f, 0x00, 0x6e, 0x00, 0x2e, 0x00,

0xa0, 0x82, 0x01, 0xc2, 0x30, 0x82, 0x01, 0xbe, 0x30, 0x82,

0x01, 0x68, 0xa0, 0x03, 0x02, 0x01, 0x02, 0x02, 0x10, 0x04,

0xd3, 0xc9, 0xdd, 0xb4, 0x38, 0x2a, 0xb5, 0x4d, 0x43, 0x4b,

0x99, 0x65, 0x19, 0xab, 0xeb, 0x30, 0x0d, 0x06, 0x09, 0x2a,

0x86, 0x48, 0x86, 0xf7, 0x0d, 0x01, 0x01, 0x04, 0x05, 0x00,

0x30, 0x16, 0x31, 0x14, 0x30, 0x12, 0x06, 0x03, 0x55, 0x04,

0x03, 0x13, 0x0b, 0x52, 0x6f, 0x6f, 0x74, 0x20, 0x41, 0x67,

0x65, 0x6e, 0x63, 0x79, 0x30, 0x1e, 0x17, 0x0d, 0x30, 0x33,

0x31, 0x32, 0x32, 0x36, 0x32, 0x32, 0x34, 0x35, 0x33, 0x37,

0x5a, 0x17, 0x0d, 0x33, 0x39, 0x31, 0x32, 0x33, 0x31, 0x32,

0x33, 0x35, 0x39, 0x35, 0x39, 0x5a, 0x30, 0x19, 0x31, 0x17,

0x30, 0x15, 0x06, 0x03, 0x55, 0x04, 0x03, 0x13, 0x0e, 0x4d,

0x65, 0x73, 0x73, 0x61, 0x67, 0x65, 0x53, 0x69, 0x67, 0x6e,

0x65, 0x72, 0x31, 0x30, 0x81, 0x9f, 0x30, 0x0d, 0x06, 0x09,

0x2a, 0x86, 0x48, 0x86, 0xf7, 0x0d, 0x01, 0x01, 0x01, 0x05,

0x00, 0x03, 0x81, 0x8d, 0x00, 0x30, 0x81, 0x89, 0x02, 0x81,

0x81, 0x00, 0x9b, 0xb4, 0x92, 0x23, 0x35, 0x3f, 0x23, 0xec,

0x4b, 0xcf, 0x4d, 0x5b, 0xed, 0x81, 0x22, 0x45, 0x62, 0x97,

0xea, 0x38, 0xff, 0x32, 0xc6, 0xa0, 0xdd, 0xeb, 0xd1, 0x18,

0x6a, 0x30, 0xec, 0x6e, 0x4b, 0x4f, 0xab, 0x2a, 0x41, 0xc7,

0x0d, 0xbb, 0xcd, 0x80, 0xdc, 0xef, 0xf2, 0xd0, 0x00, 0xd6,

0x82, 0x81, 0x7f, 0x9a, 0x9c, 0xc9, 0x41, 0xf3, 0xa8, 0x0b,

0xa3, 0x9d, 0xed, 0x9a, 0xee, 0x23, 0xb8, 0xf0, 0xe6, 0x27,

0x65, 0x30, 0x10, 0x13, 0x65, 0x75, 0x33, 0x64, 0x0b, 0x0b,

0xea, 0x7f, 0xf8, 0x3b, 0x49, 0xa7, 0xea, 0xd0, 0x2d, 0xc1,

0xf8, 0xa1, 0x66, 0xb9, 0x6d, 0xa2, 0x8d, 0x36, 0x43, 0x2e,

0xe1, 0x91, 0xe2, 0x41, 0xa1, 0xe6, 0x80, 0xc4, 0xa5, 0xf6,

0x1a, 0xa4, 0x1e, 0x1a, 0x47, 0x3e, 0x5e, 0xf1, 0x97, 0xc9,

0x26, 0x6a, 0x0c, 0xf1, 0x0f, 0xcb, 0x55, 0x03, 0xb2, 0xb7,

0x02, 0x03, 0x01, 0x00, 0x01, 0xa3, 0x4b, 0x30, 0x49, 0x30,

0x47, 0x06, 0x03, 0x55, 0x1d, 0x01, 0x04, 0x40, 0x30, 0x3e,

0x80, 0x10, 0x12, 0xe4, 0x09, 0x2d, 0x06, 0x1d, 0x1d, 0x4f,

0x00, 0x8d, 0x61, 0x21, 0xdc, 0x16, 0x64, 0x63, 0xa1, 0x18,

0x30, 0x16, 0x31, 0x14, 0x30, 0x12, 0x06, 0x03, 0x55, 0x04,

0x03, 0x13, 0x0b, 0x52, 0x6f, 0x6f, 0x74, 0x20, 0x41, 0x67,

0x65, 0x6e, 0x63, 0x79, 0x82, 0x10, 0x06, 0x37, 0x6c, 0x00,

0xaa, 0x00, 0x64, 0x8a, 0x11, 0xcf, 0xb8, 0xd4, 0xaa, 0x5c,

0x35, 0xf4, 0x30, 0x0d, 0x06, 0x09, 0x2a, 0x86, 0x48, 0x86,

0xf7, 0x0d, 0x01, 0x01, 0x04, 0x05, 0x00, 0x03, 0x41, 0x00,

0x1c, 0x97, 0xe5, 0x69, 0xac, 0x34, 0xa5, 0xa0, 0xbb, 0xc5,

0x65, 0x2e, 0xdf, 0x14, 0xa8, 0x8d, 0x4e, 0xf1, 0x86, 0x6c,

0x05, 0x5f, 0x51, 0xf3, 0xcc, 0x09, 0x8e, 0xaa, 0xa9, 0x43,

0x85, 0x11, 0x3b, 0xa9, 0xc3, 0x7d, 0x46, 0x58, 0x6b, 0xae,

0xf5, 0x6b, 0xd4, 0xef, 0xdf, 0xa5, 0x0f, 0xdb, 0x37, 0x78,

0xfd, 0x79, 0xf3, 0x31, 0x61, 0x26, 0x44, 0x98, 0x8b, 0xa4,

0xab, 0x3a, 0x89, 0x6e, 0x31, 0x81, 0xcf, 0x30, 0x81, 0xcc,

0x02, 0x01, 0x01, 0x30, 0x2a, 0x30, 0x16, 0x31, 0x14, 0x30,

0x12, 0x06, 0x03, 0x55, 0x04, 0x03, 0x13, 0x0b, 0x52, 0x6f,

0x6f, 0x74, 0x20, 0x41, 0x67, 0x65, 0x6e, 0x63, 0x79, 0x02,

0x10, 0x04, 0xd3, 0xc9, 0xdd, 0xb4, 0x38, 0x2a, 0xb5, 0x4d,

0x43, 0x4b, 0x99, 0x65, 0x19, 0xab, 0xeb, 0x30, 0x09, 0x06,

0x05, 0x2b, 0x0e, 0x03, 0x02, 0x1a, 0x05, 0x00, 0x30, 0x0d,

0x06, 0x09, 0x2a, 0x86, 0x48, 0x86, 0xf7, 0x0d, 0x01, 0x01,

0x01, 0x05, 0x00, 0x04, 0x81, 0x80, 0x4f, 0x71, 0x32, 0xc8,

0x5f, 0x2f, 0xe3, 0x8e, 0xc7, 0x8d, 0x85, 0x96, 0x28, 0x1d,

0x6e, 0xa6, 0x6e, 0x76, 0x63, 0x64, 0xae, 0x8d, 0xdc, 0x06,

0x64, 0x14, 0xeb, 0xcf, 0xcb, 0x2a, 0xd2, 0x17, 0xaa, 0xa5,

0x24, 0xd9, 0x17, 0x05, 0x07, 0x35, 0x8e, 0xa0, 0xce, 0x48,

0xed, 0x4b, 0x9d, 0xe4, 0x6c, 0xfa, 0xdc, 0x00, 0x82, 0x15,

0x6b, 0xde, 0x29, 0x90, 0x28, 0xe9, 0x53, 0xcb, 0x6b, 0xb7,

0xac, 0xa0, 0xf5, 0xd7, 0x75, 0x92, 0x89, 0x5e, 0x3d, 0x71,

0x46, 0x0c, 0x38, 0xee, 0xfb, 0x51, 0xfc, 0x4b, 0x7e, 0x70,

0x87, 0xde, 0x09, 0x8f, 0x0d, 0x63, 0x27, 0x26, 0x81, 0x43,

0xce, 0xcb, 0x03, 0x44, 0xe9, 0x70, 0xf5, 0xf0, 0x80, 0xcd,

0xe1, 0x89, 0x2b, 0xd8, 0x84, 0xd2, 0xc4, 0x90, 0xef, 0xbc,

0xad, 0x31, 0x2a, 0x3a, 0x29, 0xa0, 0xb0, 0x84, 0x9f, 0x65,

0x5b, 0x0d, 0x6f, 0x61};

Console.WriteLine("System.Security.Cryptography.Pkcs " +

"Sample: Multiple-signer signed " +

"and verified message");

Console.WriteLine("\n\n------------------------------");

Console.WriteLine(" SETUP OF CREDENTIALS ");

Console.WriteLine("------------------------------\n");

// Get the certificate of the additional message signer.

X509Certificate2Collection signerCerts = GetSignerCerts();

Console.WriteLine("\n\n----------------------");

Console.WriteLine(" SENDER SIDE ");

Console.WriteLine("----------------------\n");

// Decode the existing SignedCms message and add a signature

// to it.

byte[] encodedSignedCms = SignMsg(signedMessage, signerCerts);

Console.WriteLine("\n\n------------------------");

Console.WriteLine(" RECIPIENT SIDE ");

Console.WriteLine("------------------------\n");

// Verify all signatures in the message.

if (VerifyMsg(encodedSignedCms))

{

Console.WriteLine("\nMessage verified");

}

else

{

Console.WriteLine("\nMessage failed to verify");

}

}

// Open the My (or Personal) certificate store and search for

// credentials to sign the message with. There must be

// a certificate with subject name "MessageSigner2" in that

// certificate store.

static public X509Certificate2Collection GetSignerCerts()

{

// Open the My certificate store.

X509Store storeMy = new X509Store(StoreName.My,

StoreLocation.CurrentUser);

storeMy.Open(OpenFlags.ReadOnly);

// Display certificates to help troubleshoot

// the example's setup.

Console.WriteLine("Found certs with the following subject " +

"names in the {0} store:", storeMy.Name.ToString());

foreach (X509Certificate2 cert in storeMy.Certificates)

{

Console.WriteLine("\t{0}", cert.SubjectName.Name);

}

// Find the signer's certificate.

// Add to the signers' certificate collection.

X509Certificate2Collection signerCertsColl =

storeMy.Certificates.Find(X509FindType.FindBySubjectName,

signerName, false);

Console.WriteLine(

"Found {0} certificates in the {1} store with name {2}",

signerCertsColl.Count, storeMy.Name, signerName);

// Check to see if the certificate suggested by the example

// requirements is not present.

if (signerCertsColl.Count == 0)

{

Console.WriteLine(

"A suggested certificate to use for this example " +

"is not in the certificate store. Select " +

"an alternate certificate to use for " +

"signing the message.");

}

storeMy.Close();

return signerCertsColl;

}

// Sign the message with the private key of the signer.

static public byte[] SignMsg(

Byte[] signedCmsMsg,

X509Certificate2Collection signerCerts)

{

// Instantiate a SignedCms object, and then decode the

// input encoded SignedCms message.

// Has default SubjectIdentifierType IssuerAndSerialNumber.

// Has default Detached property value false, so message is

// included in the encoded SignedCms.

SignedCms signedCms = new SignedCms();

signedCms.Decode(signedCmsMsg);

DisplaySignedCmsProperties("Original message", signedCms);

// Sign the PKCS #7/CMS message once for each

// signer certificate. In this example, one additional

// signer is added to the SignedCms object.

foreach (X509Certificate2 cert in signerCerts)

{

Console.Write("Computing signature with signer subject "

+ "name {0} ... ", cert.SubjectName.Name);

signedCms.ComputeSignature(new CmsSigner(cert));

Console.WriteLine("Done.");

}

DisplaySignedCmsProperties("Message with additional " +

"signer", signedCms);

// Encode the PKCS #7/CMS message.

return signedCms.Encode();

}

// Verify the encoded SignedCms message and return a Boolean

// value that specifies whether the verification was successful.

static public bool VerifyMsg(byte[] encodedSignedCms)

{

// Prepare an object in which to decode and verify.

SignedCms signedCms = new SignedCms();

signedCms.Decode(encodedSignedCms);

// Catch a verification exception if you want to

// advise the message recipient that security actions might

// be appropriate.

try

{

// Verify signature. Do not validate signer

// certificate chain for the purposes of this example.

// Note that in a production environment, validating

// the signer certificate chain will probably

// be necessary.

Console.Write("Checking signatures on message ... ");

signedCms.CheckSignature(true);

Console.WriteLine("Done.");

}

catch (System.Security.Cryptography.CryptographicException e)

{

Console.WriteLine("VerifyMsg caught exception: {0}",

e.Message);

Console.WriteLine("Verification of the signed PKCS #7 " +

"failed. The message, signatures, or " +

"countersignatures might have been modified " +

"in transit or storage. The message signers or " +

"countersigners might not be who they claim to be. " +

"The message's authenticity or integrity, or both, " +

"are not guaranteed.");

return false;

}

return true;

}

// This method displays some properties of a signed

// CMS/PKCS #7 message. These properties include the number of

// signers and countersigners for each signer, whether the

// message is detached, and the version of the message.

static void DisplaySignedCmsProperties(String info, SignedCms s)

{

Console.WriteLine();

Console.WriteLine("\n>>>>> SignedCms Signer Info: {0}", info);

Console.WriteLine("\tNumber of signers:\t\t\t{0}",

s.SignerInfos.Count);

for (int i = 0; i < s.SignerInfos.Count; i++)

{

Console.WriteLine("\tSubject name of signer #{0}:\t\t{1}",

i + 1, s.SignerInfos[i].Certificate.SubjectName.Name);

Console.WriteLine("\tNumber of countersigners for " +

"signer #{0}:\t{1}",

i + 1, s.SignerInfos[i].CounterSignerInfos.Count);

}

Console.WriteLine("\tMessage detached state:\t\t\t{0}",

s.Detached);

Console.WriteLine("\tMessage version:\t\t\t{0}", s.Version);

Console.WriteLine();

}

}

}

### How to: Countersign a Message

**.NET Framework 2.0, 3.0**

This example creates a CMS/PKCS #7 signed message uses [System.Security.Cryptography.Pkcs](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs(v=vs.85).aspx). The message is signed by a single signer, and then that signature is countersigned by two other signers. The signature and countersignatures on that message are then verified.

**Example**

This example uses the following classes:

* [CmsSigner](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.cmssigner(v=vs.85).aspx)
* [ContentInfo](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.contentinfo(v=vs.85).aspx)
* [SignedCms](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.signedcms(v=vs.85).aspx)
* [SignerInfo](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.signerinfo(v=vs.85).aspx)
* [SignerInfoCollection](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.signerinfocollection(v=vs.85).aspx)
* [X509Certificate2](https://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.x509certificate2(v=vs.85).aspx)
* [X509Certificate2Collection](https://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.x509certificate2collection(v=vs.85).aspx)
* [X509Store](https://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.x509store(v=vs.85).aspx)

The following example requires that three public key certificates with the subject names "MessageSigner1", "CounterSigner1", and "CounterSigner2" be contained in the My certificate store, and that they have associated private keys.

|  |
| --- |
| **NoteNote** |
| This example is only for illustrative purposes. Production environments might use a different model in which the sender and the recipient of the message execute in different processes with their unique public key credentials. |

Set up this example by using the Makecert.exe utility, one of several ways to do so. [Certificate Creation Tool (Makecert.exe)](https://msdn.microsoft.com/en-us/library/bfsktky3(v=vs.85).aspx) is a convenient utility for generating test certificates. In a production environment, certificates are generated by a certification authority.

The following **Makecert** commands generate the required public key certificates and private keys.

**Makecert -n "CN=MessageSigner1" -ss My**

**Makecert -n "CN=CounterSigner1" -ss My**

**Makecert -n "CN=CounterSigner2" -ss My**

C#

// Copyright (c) Microsoft Corporation. All rights reserved.

#region Using directives

using System;

using System.Security.Cryptography.Pkcs;

using System.Security.Cryptography.X509Certificates;

using System.Text;

#endregion

namespace CountersignAMessage

{

class SignedCmsCountersigned

{

const String signerName = "MessageSigner1";

const String countersignerName1 = "CounterSigner1";

const String countersignerName2 = "CounterSigner2";

static void Main(string[] args)

{

Console.WriteLine("System.Security.Cryptography.Pkcs " +

"Sample: Single-signer multiple " +

"countersigner signed and verified message.");

// The original message.

const String msg = "The following is the approved corporate" +

" reorganization plan.";

Console.WriteLine("\nOriginal message (len {0}): {1} ",

msg.Length, msg);

// Convert the message to an array of bytes for signing.

Encoding unicode = Encoding.Unicode;

byte[] msgBytes = unicode.GetBytes(msg);

Console.WriteLine("\n\n------------------------------");

Console.WriteLine(" SETUP OF CREDENTIALS ");

Console.WriteLine("------------------------------\n");

X509Certificate2Collection signerCerts = GetSignerCerts();

X509Certificate2Collection countersignerCerts =

GetCountersignerCerts();

Console.WriteLine("\n\n----------------------");

Console.WriteLine(" SENDER SIDE ");

Console.WriteLine("----------------------\n");

byte[] encodedSignedCms = null;

try

{

encodedSignedCms = SignMsg(msgBytes, signerCerts,

countersignerCerts);

}

catch (ArgumentOutOfRangeException e)

{

Console.WriteLine(e.Message);

Console.WriteLine("The proper sample certificates are " +

"not in the My store. Refer to the documentation " +

"about this sample for instructions to correct this.");

return;

}

Console.WriteLine("\n\n------------------------");

Console.WriteLine(" RECIPIENT SIDE ");

Console.WriteLine("------------------------\n");

if (VerifyMsg(encodedSignedCms))

{

Console.WriteLine("\nMessage verified");

}

else

{

Console.WriteLine("\nMessage failed to verify");

}

}

// Open the My (or Personal) certificate store and search for

// credentials with which to sign the message. There must be

// a certificate with subject name "MessageSigner1".

static public X509Certificate2Collection GetSignerCerts()

{

// Open the My certificate store.

X509Store storeMy = new X509Store(StoreName.My,

StoreLocation.CurrentUser);

storeMy.Open(OpenFlags.ReadOnly);

// Display certificates to help troubleshoot

// the example's setup.

Console.WriteLine("Found certs with the following subject " +

"names in the {0} store:",

storeMy.Name);

foreach (X509Certificate2 cert in storeMy.Certificates)

Console.WriteLine("\t{0}", cert.SubjectName.Name);

// Start the collection of signer certificates to be returned.

X509Certificate2Collection signerCertsColl = new

X509Certificate2Collection();

// Find the signer's certificate.

// Add it to the signers' certificates collection.

X509Certificate2Collection foundCertColl = storeMy.

Certificates.Find(X509FindType.FindBySubjectName,

signerName, false);

Console.WriteLine(

"Found {0} certificates in the {1} store with name {2}",

foundCertColl.Count, storeMy.Name, signerName);

// Check to see if the certificate suggested by the example

// requirements is not present.

if (foundCertColl.Count == 0)

{

Console.WriteLine(

"A suggested certificate to use for this example " +

"is not in the certificate store. Select " +

"an alternate certificate to use for " +

"signing the message.");

}

signerCertsColl.Add(foundCertColl[0]);

storeMy.Close();

return signerCertsColl;

}

// Open the My (or Personal) certificate store and search for

// credentials with which to countersign the message. There must

// be two certificates: one with the subject name

// "Countersigner1", and the other with the subject name

// "Countersigner2".

static public X509Certificate2Collection GetCountersignerCerts()

{

// Open the My certificate store.

X509Store storeMy = new X509Store(StoreName.My,

StoreLocation.CurrentUser);

storeMy.Open(OpenFlags.ReadOnly);

// Start the collection of signer certificates to be returned.

X509Certificate2Collection countersignerCertsColl = new

X509Certificate2Collection();

// Find the first countersigner's certificate.

// Add it to the countersigners' certificates collection.

X509Certificate2Collection foundCertColl = storeMy.

Certificates.Find(X509FindType.FindBySubjectName,

countersignerName1, false);

Console.WriteLine(

"Found {0} certificates in the {1} store with name {2}",

foundCertColl.Count, storeMy.Name, countersignerName1);

// Check to see if the certificate suggested by the example

// requirements is not present.

if (foundCertColl.Count == 0)

{

Console.WriteLine(

"A suggested certificate to use for this example " +

"is not in the certificate store. Select " +

"an alternate certificate to use for " +

"signing the message.");

}

countersignerCertsColl.Add(foundCertColl[0]);

// Find the second countersigner's certificate.

// Add it to the countersigners' certificates collection.

foundCertColl = storeMy.

Certificates.Find(X509FindType.FindBySubjectName,

countersignerName2, false);

Console.WriteLine(

"Found {0} certificates in the {1} store with name {2}",

foundCertColl.Count, storeMy.Name, countersignerName2);

// Check to see if the certificate suggested by the example

// requirements is not present.

if (foundCertColl.Count == 0)

{

Console.WriteLine(

"A suggested certificate to use for this example " +

"is not in the certificate store. Select " +

"an alternate certificate to use for " +

"signing the message.");

}

countersignerCertsColl.Add(foundCertColl[0]);

storeMy.Close();

return countersignerCertsColl;

}

// Sign the message with the private key of the signer,

// and countersign that signature with the private key

// of each of the countersigners.

static public byte[] SignMsg(

Byte[] msg,

X509Certificate2Collection signerCerts,

X509Certificate2Collection countersignerCerts)

{

// There must be at least one certificate in the collection.

if (signerCerts.Count == 0 || countersignerCerts.Count == 0)

{

throw new ArgumentOutOfRangeException("Empty certificate" +

" collection passed to SignMsg.");

}

// Place the message in a ContentInfo object.

// This is required to build a SignedCms object.

ContentInfo contentInfo = new ContentInfo(msg);

// Instantiate a SignedCms object with the ContentInfo above.

// Use the default SubjectIdentifierType

// IssuerAndSerialNumber.

// Use the default Detached property value FALSE so that the

// message is included in the encoded SignedCms.

SignedCms signedCms = new SignedCms(contentInfo);

// Sign the CMS/PKCS #7 message once for each

// signer certificate.

foreach (X509Certificate2 cert in signerCerts)

{

Console.Write("Computing signature with signer subject " +

"name {0} ... ", cert.SubjectName.Name);

signedCms.ComputeSignature(new CmsSigner(cert));

Console.WriteLine("Done.");

}

// Countersign the first signature in the CMS/PKCS #7 message

// once for each countersigner certificate.

foreach (X509Certificate2 cert in countersignerCerts)

{

Console.Write("Computing countersignature with signer " +

"subject name {0} ... ",

cert.SubjectName.Name);

signedCms.SignerInfos[0].ComputeCounterSignature(new

CmsSigner(cert));

Console.WriteLine("Done.");

}

// Encode the CMS/PKCS #7 message.

return signedCms.Encode();

}

// Verify the encoded SignedCms message and return a Boolean

// value that specifies whether the verification was successful.

static public bool VerifyMsg(byte[] encodedSignedCms)

{

// Prepare an object in which to decode and verify.

SignedCms signedCms = new SignedCms();

signedCms.Decode(encodedSignedCms);

// Check the number of signers and countersigners.

DisplaySignedCmsProperties("In VerifyMsg method", signedCms);

// Catch a verification exception if you want to

// advise the message recipient that security actions

// might be appropriate.

try

{

// Verify signature. Do not validate the signer

// certificate chain for the purposes of this example.

// Note that in a production environment, validating

// the signer certificate chain will probably

// be necessary.

Console.Write("Checking signature and countersignatures" +

" on message ... ");

signedCms.CheckSignature(true);

Console.WriteLine("Done.");

}

catch (System.Security.Cryptography.CryptographicException e)

{

Console.WriteLine("VerifyMsg caught exception: {0}",

e.Message);

Console.WriteLine("Verification of the signed " +

"CMS/PKCS #7 failed. The message, signatures, or " +

"countersignatures might have been modified " +

"in transit or storage. The message signers or " +

"countersigners might not be who they claim to be. " +

"The message's authenticity or integrity, or both, " +

"are not guaranteed.");

return false;

}

return true;

}

// This method displays some properties of a signed

// CMS/PKCS #7 message. These properties include the number of

// signers and countersigners for each signer, whether the

// message is detached, and the version of the message.

static void DisplaySignedCmsProperties(String info, SignedCms s)

{

Console.WriteLine();

Console.WriteLine("\n>>>>> SignedCms Signer Info: {0}", info);

Console.WriteLine("\tNumber of signers:\t\t\t{0}",

s.SignerInfos.Count);

for (int i = 0; i < s.SignerInfos.Count; i++)

{

Console.WriteLine("\tSubject name of signer #{0}:\t\t{1}",

i + 1, s.SignerInfos[i].Certificate.SubjectName.Name);

Console.WriteLine("\tNumber of countersigners for " +

"signer #{0}:\t{1}",

i + 1, s.SignerInfos[i].CounterSignerInfos.Count);

}

Console.WriteLine("\tMessage detached state:\t\t\t{0}",

s.Detached);

Console.WriteLine("\tMessage version:\t\t\t{0}", s.Version);

Console.WriteLine();

}

}

}

### How to: Envelope a Message for One Recipient

**.NET Framework 2.0, 3.0**

This example creates a CMS/PKCS #7 enveloped message by using [System.Security.Cryptography.Pkcs](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs(v=vs.85).aspx). The message is encrypted for a single recipient. The message is then decrypted using that recipient's private key. The example uses the [EnvelopedCms](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.envelopedcms(v=vs.85).aspx) object, which allows messages to be encrypted for one or more recipients, or *enveloped*.

**Example**

This example uses the following classes:

* [CmsRecipient](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.cmsrecipient(v=vs.85).aspx)
* [ContentInfo](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.contentinfo(v=vs.85).aspx)
* **EnvelopedCms**
* [X509Certificate2](https://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.x509certificate2(v=vs.85).aspx)
* [X509Certificate2Collection](https://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.x509certificate2collection(v=vs.85).aspx)
* [X509Store](https://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.x509store(v=vs.85).aspx)

To run on a single computer, the following example requires that a public key certificate with the subject name "Recipient1" be contained in both the AddressBook and My certificate stores. It also requires that the associated private key be stored on that computer. This example first acts as the message sender, and then as the message recipient, by using the same public key credentials in each role. As the sender, it searches the AddressBook certificate store for the recipient's certificate, and uses that certificate to encrypt the message. As the recipient, it searches the My certificate store for the certificate, and uses the associated private key to decrypt the message.

|  |
| --- |
| **NoteNote** |
| This example is only for illustrative purposes. Production environments might use a different model in which the sender and the recipient of the message execute in different processes with their unique public key credentials. |

Set up this example by using the Makecert.exe utility, one of several ways to do so. [Certificate Creation Tool (Makecert.exe)](https://msdn.microsoft.com/en-us/library/bfsktky3(v=vs.85).aspx) is a convenient utility for generating test certificates. In a production environment, certificates are generated by a certification authority.

The following **Makecert** command generates the required public key certificates and private keys.

**Makecert -n "CN=Recipient1" -ss My**

This command places the appropriate public key certificate in the My certificate store and generates the private key. At this point, you need to get a copy of the public key certificate into the AddressBook certificate store. To do so, export the public key certificate, and then import it into the AddressBook certificate store by following the procedure in [How to: Export and Import a Public Key Certificate](https://msdn.microsoft.com/en-us/library/ms180963(v=vs.85).aspx).

C#

// Copyright (c) Microsoft Corporation. All rights reserved.

#region Using directives

using System;

using System.Security.Cryptography;

using System.Security.Cryptography.Pkcs;

using System.Security.Cryptography.X509Certificates;

using System.Text;

#endregion

namespace EnvelopAMessageForOneRecipient

{

class EnvelopedCmsSingleRecipient

{

const String recipientName = "Recipient1";

static void Main(string[] args)

{

Console.WriteLine("System.Security.Cryptography.Pkcs " +

"Sample: Single-recipient encrypted and decrypted message");

// Original message.

const String msg = "Here is your personal identification number:";

Console.WriteLine("\nOriginal message (len {0}): {1} ",

msg.Length, msg);

// Convert message to an array of Unicode bytes for signing.

UnicodeEncoding unicode = new UnicodeEncoding();

byte[] msgBytes = unicode.GetBytes(msg);

Console.WriteLine("\n\n------------------------------");

Console.WriteLine(" SETUP OF CREDENTIALS ");

Console.WriteLine("------------------------------\n");

// The recipient's certificate is necessary to encrypt

// the message for that recipient.

X509Certificate2 recipientCert = GetRecipientCert();

Console.WriteLine("\n\n----------------------");

Console.WriteLine(" SENDER SIDE ");

Console.WriteLine("----------------------\n");

byte[] encodedEnvelopedCms = EncryptMsg(msgBytes,

recipientCert);

Console.Write("\nMessage after encryption (len {0}): ",

encodedEnvelopedCms.Length);

foreach (byte b in encodedEnvelopedCms)

{

Console.Write("{0:x}", b);

}

Console.WriteLine();

Console.WriteLine("\n\n------------------------");

Console.WriteLine(" RECIPIENT SIDE ");

Console.WriteLine("------------------------\n");

Byte[] decryptedMsg = DecryptMsg(encodedEnvelopedCms);

// Convert Unicode bytes to the original message string.

Console.WriteLine("\nDecrypted Message: {0}",

unicode.GetString(decryptedMsg));

}

// Open the AddressBook (also known as Other in

// Internet Explorer) certificate store and search for

// a recipient certificate with which to encrypt the

// message. The certificate must have a subject name

// of "Recipient1".

static public X509Certificate2 GetRecipientCert()

{

// Open the AddressBook local user X509 certificate store.

X509Store storeAddressBook = new X509Store(StoreName.

AddressBook, StoreLocation.CurrentUser);

storeAddressBook.Open(OpenFlags.ReadOnly);

// Display certificates to help troubleshoot

// the example's setup.

Console.WriteLine(

"Found certs with the following subject names in the " +

"{0} store:", storeAddressBook.Name);

foreach (X509Certificate2 cert in storeAddressBook.Certificates)

{

Console.WriteLine("\t{0}", cert.SubjectName.Name);

}

// Get recipient certificate.

// For purposes of this sample, do not validate the

// certificate. Note that in a production environment,

// validating the certificate will probably be necessary.

X509Certificate2Collection certColl = storeAddressBook.

Certificates.Find(X509FindType.FindBySubjectName,

recipientName, false);

Console.WriteLine(

"Found {0} certificates in the {1} store with name {2}",

certColl.Count, storeAddressBook.Name, recipientName);

// Check to see if the certificate suggested by the example

// requirements is not present.

if (certColl.Count == 0)

{

Console.WriteLine(

"A suggested certificate to use for this example " +

"is not in the certificate store. Select " +

"an alternate certificate to use for " +

"signing the message.");

}

storeAddressBook.Close();

return certColl[0];

}

// Encrypt the message with the public key of

// the recipient. This is done by enveloping the message by

// using an EnvelopedCms object.

static public byte[] EncryptMsg(

Byte[] msg,

X509Certificate2 recipientCert)

{

// Place the message in a ContentInfo object.

// This is required to build an EnvelopedCms object.

ContentInfo contentInfo = new ContentInfo(msg);

// Instantiate an EnvelopedCms object with the ContentInfo

// above.

// Has default SubjectIdentifierType IssuerAndSerialNumber.

// Has default ContentEncryptionAlgorithm property value

// RSA\_DES\_EDE3\_CBC.

EnvelopedCms envelopedCms = new EnvelopedCms(contentInfo);

// Formulate a CmsRecipient object that

// represents information about the recipient

// to encrypt the message for.

CmsRecipient recip1 = new CmsRecipient(

SubjectIdentifierType.IssuerAndSerialNumber,

recipientCert);

Console.Write(

"Encrypting data for a single recipient of " +

"subject name {0} ... ",

recip1.Certificate.SubjectName.Name);

// Encrypt the message for the recipient.

envelopedCms.Encrypt(recip1);

Console.WriteLine("Done.");

// The encoded EnvelopedCms message contains the message

// ciphertext and the information about each recipient

// that the message was enveloped for.

return envelopedCms.Encode();

}

// Decrypt the encoded EnvelopedCms message.

static public Byte[] DecryptMsg(byte[] encodedEnvelopedCms)

{

// Prepare object in which to decode and decrypt.

EnvelopedCms envelopedCms = new EnvelopedCms();

// Decode the message.

envelopedCms.Decode(encodedEnvelopedCms);

// Display the number of recipients the message is

// enveloped for; it should be 1 for this example.

DisplayEnvelopedCms(envelopedCms, false);

// Decrypt the message for the single recipient.

Console.Write("Decrypting Data ... ");

envelopedCms.Decrypt(envelopedCms.RecipientInfos[0]);

Console.WriteLine("Done.");

// The decrypted message occupies the ContentInfo property

// after the Decrypt method is invoked.

return envelopedCms.ContentInfo.Content;

}

// Display the ContentInfo property of an EnvelopedCms object.

static private void DisplayEnvelopedCmsContent(String desc,

EnvelopedCms envelopedCms)

{

Console.WriteLine(desc + " (length {0}): ",

envelopedCms.ContentInfo.Content.Length);

foreach (byte b in envelopedCms.ContentInfo.Content)

{

Console.Write(b.ToString() + " ");

}

Console.WriteLine();

}

// Display some properties of an EnvelopedCms object.

static private void DisplayEnvelopedCms(EnvelopedCms e,

Boolean displayContent)

{

Console.WriteLine("\nEnveloped CMS/PKCS #7 Message " +

"Information:");

Console.WriteLine(

"\tThe number of recipients for the Enveloped CMS/PKCS " +

"#7 is: {0}", e.RecipientInfos.Count);

for (int i = 0; i < e.RecipientInfos.Count; i++)

{

Console.WriteLine(

"\tRecipient #{0} has type {1}.",

i + 1,

e.RecipientInfos[i].RecipientIdentifier.Type);

}

if (displayContent)

{

DisplayEnvelopedCmsContent("Enveloped CMS/PKCS " +

"#7 Content", e);

}

Console.WriteLine();

}

}

}

### How to: Envelope a Message for Multiple Recipients

**.NET Framework 2.0, 3.0**

This example creates a CMS/PKCS #7 enveloped message by using [System.Security.Cryptography.Pkcs](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs(v=vs.85).aspx). The message is encrypted for multiple recipients. The message is then decrypted for each recipient using that recipient's private key. This example uses the [EnvelopedCms](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.envelopedcms(v=vs.85).aspx) object, which allows messages to be encrypted for one or more recipients, or *enveloped*.

**Example**

This example uses the following classes:

* [CmsRecipient](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.cmsrecipient(v=vs.85).aspx)
* [CmsRecipientCollection](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.cmsrecipientcollection(v=vs.85).aspx)
* [ContentInfo](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.contentinfo(v=vs.85).aspx)
* **EnvelopedCms**
* [X509Certificate2](https://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.x509certificate2(v=vs.85).aspx)
* [X509Certificate2Collection](https://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.x509certificate2collection(v=vs.85).aspx)
* [X509Store](https://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.x509store(v=vs.85).aspx)

To run on a single computer, the following example requires that two public key certificates with the subject names "Recipient1" and "Recipient2" be contained in both the AddressBook and My certificate stores. This example also requires that the associated private keys be stored on that computer. The example code first acts as the message sender, and then as the message recipient. The same public key credentials are used in each role. As such, the example requires that the public key certificate be in two stores. As the sender, the example searches the AddressBook certificate store for the recipient's certificate and uses it to encrypt the message. As the recipient, it searches the My certificate store for the certificate, and uses the associated private key to decrypt the message.

|  |
| --- |
| **NoteNote** |
| This example is only for illustrative purposes. Production environments might use a different model in which the sender and the recipient of the message execute in different processes with their unique public key credentials. |

Set up this example by using the Makecert.exe utility, one of several ways to do so. [Certificate Creation Tool (Makecert.exe)](https://msdn.microsoft.com/en-us/library/bfsktky3(v=vs.85).aspx) is a convenient utility for testing certificates. In a production environment, certificates are generated by a certification authority.

The following **Makecert** commands generate the required public key certificates and private keys.

**Makecert -n "CN=Recipient1" -ss My**

**Makecert -n "CN=Recipient2" -ss My**

These commands place the appropriate public key certificates in the My certificate store. At this point, you need to export the public key certificates, and then import them into the AddressBook certificate store by following the procedure in [How to: Export and Import a Public Key Certificate](https://msdn.microsoft.com/en-us/library/ms180963(v=vs.85).aspx) for each recipient.

C#

// Copyright (c) Microsoft Corporation. All rights reserved.

#region Using directives

using System;

using System.Security.Cryptography;

using System.Security.Cryptography.Pkcs;

using System.Security.Cryptography.X509Certificates;

using System.Text;

#endregion

namespace EnvelopAMessageForMultipleRecipients

{

class EnvelopedCmsMultipleRecipients

{

const String recipientName1 = "Recipient1";

const String recipientName2 = "Recipient2";

static void Main(string[] args)

{

Console.WriteLine("System.Security.Cryptography.Pkcs " +

"Sample: Multiple-recipient " +

"encrypted and decrypted message");

// Original message.

const String msg = "To all department heads: The following " +

"organizational changes will be announced next week:";

Console.WriteLine("\nOriginal message (len {0}): {1} ",

msg.Length, msg);

// Convert message to array of Unicode bytes for signing.

UnicodeEncoding unicode = new UnicodeEncoding();

byte[] msgBytes = unicode.GetBytes(msg);

Console.WriteLine("\n\n------------------------------");

Console.WriteLine(" SETUP OF CREDENTIALS ");

Console.WriteLine("------------------------------\n");

// The recipients' certificates are necessary to encrypt

// the message for those recipients.

X509Certificate2Collection recipientCerts = GetRecipientCerts();

Console.WriteLine("\n\n----------------------");

Console.WriteLine(" SENDER SIDE ");

Console.WriteLine("----------------------\n");

byte[] encodedEnvelopedCms = EncryptMsg(msgBytes,

recipientCerts);

Console.Write("\nMessage after encryption (len {0}): ",

encodedEnvelopedCms.Length);

foreach (byte b in encodedEnvelopedCms)

{

Console.Write("{0:x}", b);

}

Console.WriteLine();

Console.WriteLine("\n\n------------------------");

Console.WriteLine(" RECIPIENT SIDE ");

Console.WriteLine("------------------------\n");

// Decrypts the message for one of the recipients.

// Return the decrypted message to display it.

Byte[] decryptedMsg = DecryptMsg(encodedEnvelopedCms);

// Convert Unicode bytes to the original message string.

Console.WriteLine("\nDecrypted Message: {0}",

unicode.GetString(decryptedMsg));

}

// Open the AddressBook (called Other in Internet Explorer)

// certificate store and search for recipient

// certificates with which to encrypt the message. It must

// include two certificates: one with the subject name

// "Recipient1", and one with the subject name "Recipient2".

static public X509Certificate2Collection GetRecipientCerts()

{

// Open the AddressBook local user X509 certificate store.

X509Store storeAddressBook = new X509Store(StoreName.

AddressBook, StoreLocation.CurrentUser);

storeAddressBook.Open(OpenFlags.ReadOnly);

// Display certificates to help troubleshoot the

// example's setup.

Console.WriteLine(

"Found certs with the following subject names in the " +

"{0} store:", storeAddressBook.Name);

foreach (X509Certificate2 cert in storeAddressBook.Certificates)

{

Console.WriteLine("\t{0}", cert.SubjectName.Name);

}

// Get recipient certificates.

// For purposes of this sample, do not validate the

// certificates. Note that in a production environment,

// validating the certificates will probably be necessary.

// Get first recipient certificate.

X509Certificate2Collection certColl = storeAddressBook.

Certificates.Find(X509FindType.FindBySubjectName,

recipientName1, false);

Console.WriteLine(

"Found {0} certificates in the {1} store with name {2}",

certColl.Count, storeAddressBook.Name, recipientName1);

X509Certificate2Collection recipientCerts =

new X509Certificate2Collection();

// Check to see if the certificate suggested by the example

// requirements is not present.

if (certColl.Count == 0)

{

Console.WriteLine(

"A suggested certificate to use for this example " +

"is not in the certificate store. Select " +

"an alternate certificate to use for " +

"signing the message.");

}

recipientCerts.Add(certColl[0]);

// Get second recipient certificate.

certColl = storeAddressBook.

Certificates.Find(X509FindType.FindBySubjectName,

recipientName2, false);

Console.WriteLine(

"Found {0} certificates in the {1} store with name {2}",

certColl.Count, storeAddressBook.Name, recipientName2);

// Check to see if the certificate suggested by the example

// requirements is not present.

if (certColl.Count == 0)

{

Console.WriteLine(

"A suggested certificate to use for this example " +

"is not in the certificate store. Select " +

"an alternate certificate to use for " +

"signing the message.");

}

recipientCerts.Add(certColl[0]);

storeAddressBook.Close();

return recipientCerts;

}

// Encrypt the message for each recipient by using the public

// key of that recipient. This is done by

// enveloping the message by using an EnvelopedCms object.

static public byte[] EncryptMsg(

Byte[] msg,

X509Certificate2Collection recipientCerts)

{

// Place message in a ContentInfo object.

// This is required to build an EnvelopedCms object.

ContentInfo contentInfo = new ContentInfo(msg);

// Instantiate EnvelopedCms object with the ContentInfo

// above.

// Has default SubjectIdentifierType IssuerAndSerialNumber.

// Has default ContentEncryptionAlgorithm property value

// RSA\_DES\_EDE3\_CBC.

EnvelopedCms envelopedCms = new EnvelopedCms(contentInfo);

// Formulate a CmsRecipientCollection object that

// represents information about the set of recipients

// to encrypt the message for.

CmsRecipientCollection recips =

new CmsRecipientCollection(

SubjectIdentifierType.IssuerAndSerialNumber,

recipientCerts);

Console.WriteLine("\nEncrypting data for multiple " +

"recipients with subject names: ");

foreach (CmsRecipient recip in recips)

{

Console.WriteLine("\t" +

recip.Certificate.SubjectName.Name);

}

// Encrypt the message for the collection of recipients.

envelopedCms.Encrypt(recips);

Console.WriteLine("Done.");

// The encoded EnvelopedCms message contains the message

// ciphertext and the information about each recipient

// that the message was enveloped for.

return envelopedCms.Encode();

}

// Decrypt the encoded EnvelopedCms message for one of the

// recipients.

static public Byte[] DecryptMsg(byte[] encodedEnvelopedCms)

{

// Prepare object in which to decode and decrypt.

EnvelopedCms envelopedCms = new EnvelopedCms();

// Decode the message.

envelopedCms.Decode(encodedEnvelopedCms);

// Display the number of recipients the message is

// enveloped for; it should be 2 for this example.

DisplayEnvelopedCms(envelopedCms, false);

// Decrypt the message.

// The message is decrypted for the recipient that

// you find the first matching private key for.

// A line similar to the following, however,

// decrypts the message for a specified recipient. In

// in this case, the first recipient.

// envelopedCms.Decrypt(envelopedCms.RecipientInfos[0]);

Console.Write("Decrypting Data for one recipient ... ");

envelopedCms.Decrypt();

Console.WriteLine("Done.");

// The decrypted message occupies the ContentInfo property

// after the Decrypt method is invoked.

return envelopedCms.ContentInfo.Content;

}

// Display the ContentInfo property of an EnvelopedCms object.

static private void DisplayEnvelopedCmsContent(String desc,

EnvelopedCms envelopedCms)

{

Console.WriteLine(desc + " (length {0}): ",

envelopedCms.ContentInfo.Content.Length);

foreach (byte b in envelopedCms.ContentInfo.Content)

{

Console.Write(b.ToString() + " ");

}

Console.WriteLine();

}

// Display some properties of an EnvelopedCms object.

static private void DisplayEnvelopedCms(EnvelopedCms e,

Boolean displayContent)

{

Console.WriteLine("\nEnveloped PKCS #7 Message Information:");

Console.WriteLine(

"\tThe number of recipients for the Enveloped PKCS #7 " +

"is: {0}", e.RecipientInfos.Count);

for (int i = 0; i < e.RecipientInfos.Count; i++)

{

Console.WriteLine(

"\tRecipient #{0} has type {1}.",

i + 1,

e.RecipientInfos[i].RecipientIdentifier.Type);

}

if (displayContent)

{

DisplayEnvelopedCmsContent("Enveloped PKCS #7 Content", e);

}

Console.WriteLine();

}

}

}

### How to: Sign and Envelop a Message

**.NET Framework 2.0, 3.0**

This example creates a CMS/PKCS #7 enveloped signed message by using [System.Security.Cryptography.Pkcs](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs(v=vs.85).aspx). The message is first signed by a single signer and then encrypted for a single recipient. The message is then decrypted by using the recipient's private key, and the signature is verified. The example uses the [SignedCms](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.signedcms(v=vs.85).aspx) object, which allows a message to be signed and countersigned by one or more signers. It also uses the [EnvelopedCms](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.envelopedcms(v=vs.85).aspx) object, which allows messages to be encrypted for one or more recipients, or *enveloped*.

**Example**

This example uses the following classes:

* [CmsRecipient](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.cmsrecipient(v=vs.85).aspx)
* [CmsSigner](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.cmssigner(v=vs.85).aspx)
* [ContentInfo](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.contentinfo(v=vs.85).aspx)
* **EnvelopedCms**
* **SignedCms**
* [X509Certificate2](https://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.x509certificate2(v=vs.85).aspx)
* [X509Certificate2Collection](https://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.x509certificate2collection(v=vs.85).aspx)
* [X509Store](https://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.x509store(v=vs.85).aspx)

To run on a single computer, this example requires the following:

* One public key certificate with the subject name "MessageSigner1" be in the My certificate store.
* Another certificate with the subject name "Recipient1" be contained in both the AddressBook and My certificate stores.
* Associated private keys be stored on the single computer.

The example code first acts as the message sender and then as the message recipient. The example uses the same public key credentials in each role. As such, the example requires that the public key certificate for the subject name "Recipient1" be in two stores. As the sender, it searches the AddressBook certificate store for the recipient's certificate, and uses that to encrypt the message. As the recipient, it searches the My certificate store for the certificate and uses the associated private key to decrypt the message.

|  |
| --- |
| **NoteNote** |
| This example is only for illustrative purposes. Production environments might use a different model in which the sender and the recipient of the message execute in different processes with their unique public key credentials. |

Set up this example by using the Makecert.exe utility, one of several ways to do so. [Certificate Creation Tool (Makecert.exe)](https://msdn.microsoft.com/en-us/library/bfsktky3(v=vs.85).aspx) is a convenient utility for testing certificates. In a production environment, certificates are generated by a certification authority.

The following **Makecert** commands generate the required public key certificates and private keys.

**Makecert -n "CN=MessageSigner1" -ss My**

**Makecert -n "CN=Recipient1" -sky exchange -ss M**

These commands place the appropriate public key certificates in the My certificate store. To get the public key certificate for subject name "Recipient1" in the AddressBook certificate store, export the public key certificate, and then import it into the AddressBook store by following the procedure in [How to: Export and Import a Public Key Certificate](https://msdn.microsoft.com/en-us/library/ms180963(v=vs.85).aspx).

C#

// Copyright (c) Microsoft Corporation. All rights reserved.

#region Using directives

using System;

using System.Security.Cryptography;

using System.Security.Cryptography.Pkcs;

using System.Security.Cryptography.X509Certificates;

using System.Text;

#endregion

namespace SigningAndEnvelopingMessage

{

class EnvelopedSignedCms

{

const String signerName = "MessageSigner1";

const String recipientName = "Recipient1";

static void Main(string[] args)

{

byte[] origMsg;

Console.WriteLine("System.Security.Cryptography.Pkcs " +

"Sample: Encrypted, signed, decrypted, and " +

"verified message");

// Original message.

const String msg = "Here are the sales figures for the " +

"upcoming quarterly report to Wall Street.";

Console.WriteLine("\nOriginal message (len {0}): {1} ",

msg.Length, msg);

// Convert message to array of bytes for signing.

Encoding unicode = Encoding.Unicode;

byte[] msgBytes = unicode.GetBytes(msg);

Console.WriteLine("\n\n------------------------------");

Console.WriteLine(" SETUP OF CREDENTIALS ");

Console.WriteLine("------------------------------\n");

// The signer's private key, obtained by association with

// their signing certificate, is necessary to sign the

// message.

X509Certificate2 signerCert = GetSignerCert();

// The recipient's certificate is necessary to encrypt

// the message for that recipient.

X509Certificate2 recipientCert = GetRecipientCert();

Console.WriteLine("\n\n----------------------");

Console.WriteLine(" SENDER SIDE ");

Console.WriteLine("----------------------\n");

byte[] encodedSignedCms = SignMsg(msgBytes, signerCert);

// Encrypt the encoded SignedCms message.

byte[] encodedEnvelopedCms = EncryptMsg(encodedSignedCms,

recipientCert);

Console.Write("\nMessage after encryption (len {0}): ",

encodedEnvelopedCms.Length);

foreach (byte b in encodedEnvelopedCms)

{

Console.Write("{0:x}", b);

}

Console.WriteLine();

Console.WriteLine("\n\n------------------------");

Console.WriteLine(" RECIPIENT SIDE ");

Console.WriteLine("------------------------\n");

encodedSignedCms = DecryptMsg(encodedEnvelopedCms);

// Get the original message back after verification so

// it can be displayed.

if (VerifyMsg(encodedSignedCms, out origMsg))

{

Console.WriteLine("\nMessage verified");

}

else

{

Console.WriteLine("\nMessage failed to verify");

}

// Convert Unicode bytes to the original message string.

Console.WriteLine("\nDecrypted Authenticated Message: {0}",

unicode.GetString(origMsg));

}

// Open the My (or Personal) certificate store. Search for

// credentials with which to sign the message. The certificate

// must have a the subject name "MessageSigner1".

static public X509Certificate2 GetSignerCert()

{

// Open the My certificate store.

X509Store storeMy = new X509Store(StoreName.My,

StoreLocation.CurrentUser);

storeMy.Open(OpenFlags.ReadOnly);

// Display certificates to help troubleshoot the

// example's setup.

Console.WriteLine("Found certs with the following subject " +

"names in the {0} store:", storeMy.Name);

foreach (X509Certificate2 cert in storeMy.Certificates)

{

Console.WriteLine("\t{0}", cert.SubjectName.Name);

}

// Find the signer's certificate.

X509Certificate2Collection certColl =

storeMy.Certificates.Find(X509FindType.FindBySubjectName,

signerName, false);

Console.WriteLine(

"Found {0} certificates in the {1} store with name {2}",

certColl.Count, storeMy.Name, signerName);

// Check to see if the certificate suggested by the example

// requirements is not present.

if (certColl.Count == 0)

{

Console.WriteLine(

"A suggested certificate to use for this example " +

"is not in the certificate store. Select " +

"an alternate certificate to use for " +

"signing the message.");

}

storeMy.Close();

return certColl[0];

}

// Open the AddressBook (called Other in Internet Explorer)

// certificate store and search for a recipient

// certificate with which to encrypt the message. The certificate

// must have the subject name "Recipient1".

static public X509Certificate2 GetRecipientCert()

{

// Open the AddressBook local user X509 certificate store.

X509Store storeAddressBook = new X509Store(StoreName.

AddressBook, StoreLocation.CurrentUser);

storeAddressBook.Open(OpenFlags.ReadOnly);

// Display certificates to help troubleshoot the

// example's setup.

Console.WriteLine(

"Found certs with the following subject names in the " +

"{0} store:",

storeAddressBook.Name);

foreach (X509Certificate2 cert in storeAddressBook.Certificates)

{

Console.WriteLine("\t{0}", cert.SubjectName.Name);

}

// Get recipient certificate.

// For purposes of this sample, do not validate the

// certificate. Note that in a production environment,

// validating the certificate will probably be necessary.

X509Certificate2Collection certColl = storeAddressBook.

Certificates.Find(X509FindType.FindBySubjectName,

recipientName, false);

Console.WriteLine(

"Found {0} certificates in the {1} store with name {2}",

certColl.Count, storeAddressBook.Name, recipientName);

// Check to see if the certificate suggested by the example

// requirements is not present.

if (certColl.Count == 0)

{

Console.WriteLine(

"A suggested certificate to use for this example " +

"is not in the certificate store. Select " +

"an alternate certificate to use for " +

"signing the message.");

}

storeAddressBook.Close();

return certColl[0];

}

// Sign the message by the using the private key of the signer.

// Note that signer's public key certificate is input here

// because it is used to locate the corresponding private key.

static public byte[] SignMsg(

Byte[] msg,

X509Certificate2 signerCert)

{

// Place message in a ContentInfo object.

// This is required to build a SignedCms object.

ContentInfo contentInfo = new ContentInfo(msg);

// Instantiate SignedCms object with the ContentInfo above.

// Has default SubjectIdentifierType IssuerAndSerialNumber.

// Has default Detached property value false, so message is

// included in the encoded SignedCms.

SignedCms signedCms = new SignedCms(contentInfo);

// Formulate a CmsSigner object, which has all the needed

// characteristics of the signer.

CmsSigner cmsSigner = new CmsSigner(signerCert);

// Sign the PKCS #7 message.

Console.Write("Computing signature with signer subject " +

"name {0} ... ", signerCert.SubjectName.Name);

signedCms.ComputeSignature(cmsSigner);

Console.WriteLine("Done.");

// Encode the PKCS #7 message.

return signedCms.Encode();

}

// Verify the encoded SignedCms message and return a Boolean

// value that specifies whether the verification was successful.

// Also return the original message that was signed, which is

// available as part of the SignedCms message after it

// is decoded.

static public bool VerifyMsg(byte[] encodedSignedCms,

out byte[] origMsg)

{

// Prepare a SignedCms object in which to decode

// and verify.

SignedCms signedCms = new SignedCms();

signedCms.Decode(encodedSignedCms);

// Catch a verification exception in the event you want to

// advise the message recipient that security actions

// might be appropriate.

try

{

// Verify signature. Do not validate signer

// certificate for the purposes of this example.

// Note that in a production environment, validating

// the signer certificate chain will probably be

// necessary.

Console.Write("Checking signature on message ... ");

signedCms.CheckSignature(true);

Console.WriteLine("Done.");

}

catch (System.Security.Cryptography.CryptographicException e)

{

Console.WriteLine("VerifyMsg caught exception: {0}",

e.Message);

Console.WriteLine("The message may have been modified " +

"in transit or storage. Authenticity of the " +

"message is not guaranteed.");

origMsg = null;

return false;

}

origMsg = signedCms.ContentInfo.Content;

return true;

}

// Encrypt the message with the public key of

// the recipient. This is done by enveloping the message by

// using a EnvelopedCms object.

static public byte[] EncryptMsg(Byte[] msg,

X509Certificate2 recipientCert)

{

// Place message in a ContentInfo object.

// This is required to build an EnvelopedCms object.

ContentInfo contentInfo = new ContentInfo(msg);

// Instantiate EnvelopedCms object with the ContentInfo

// above.

// Has default SubjectIdentifierType IssuerAndSerialNumber.

// Has default ContentEncryptionAlgorithm property value

// RSA\_DES\_EDE3\_CBC.

EnvelopedCms envelopedCms = new EnvelopedCms(contentInfo);

// Formulate a CmsRecipient object that

// represents information about the recipient

// to encrypt the message for.

CmsRecipient recip1 = new CmsRecipient(

SubjectIdentifierType.IssuerAndSerialNumber,

recipientCert);

Console.Write("Encrypting data for a single recipient of " +

"subject name {0} ... ",

recip1.Certificate.SubjectName.Name);

// Encrypt the message for the recipient.

envelopedCms.Encrypt(recip1);

Console.WriteLine("Done.");

// The encoded EnvelopedCms message contains the encrypted

// message and the information about each recipient that

// the message was enveloped for.

return envelopedCms.Encode();

}

// Decrypt the encoded EnvelopedCms message.

static public Byte[] DecryptMsg(byte[] encodedEnvelopedCms)

{

// Prepare object in which to decode and decrypt.

EnvelopedCms envelopedCms = new EnvelopedCms();

// Decode the message.

envelopedCms.Decode(encodedEnvelopedCms);

// Display the number of recipients the message is

// enveloped for; it should be 1 for this example.

DisplayEnvelopedCms(envelopedCms, false);

// Decrypt the message for the single recipient.

// Note that the following call to the Decrypt method

// accomplishes the same result:

// envelopedCms.Decrypt();

Console.Write("Decrypting Data ... ");

envelopedCms.Decrypt(envelopedCms.RecipientInfos[0]);

Console.WriteLine("Done.");

return envelopedCms.Encode();

}

// Display the ContentInfo property of a SignedCms object.

private void DisplaySignedCmsContent(String desc,

SignedCms signedCms)

{

Console.WriteLine(desc + " (length {0}): ",

signedCms.ContentInfo.Content.Length);

foreach (byte b in signedCms.ContentInfo.Content)

{

Console.Write(b.ToString() + " ");

}

Console.WriteLine();

}

// Display the ContentInfo property of an EnvelopedCms object.

static private void DisplayEnvelopedCmsContent(String desc,

EnvelopedCms envelopedCms)

{

Console.WriteLine(desc + " (length {0}): ",

envelopedCms.ContentInfo.Content.Length);

foreach (byte b in envelopedCms.ContentInfo.Content)

{

Console.Write(b.ToString() + " ");

}

Console.WriteLine();

}

// Display some properties of an EnvelopedCms object.

static private void DisplayEnvelopedCms(EnvelopedCms e,

Boolean displayContent)

{

Console.WriteLine("\nEnveloped PKCS #7 Message Information:");

Console.WriteLine(

"\tThe number of recipients for the Enveloped PKCS #7 " +

"is: {0}",

e.RecipientInfos.Count);

for (int i = 0; i < e.RecipientInfos.Count; i++)

{

Console.WriteLine(

"\tRecipient #{0} has type {1}.",

i + 1,

e.RecipientInfos[i].RecipientIdentifier.Type);

}

if (displayContent)

{

DisplayEnvelopedCmsContent("Enveloped PKCS #7 Content", e);

}

Console.WriteLine();

}

}

}

### Supporting Tasks for Using System.Security.Cryptography.Pkcs

**.NET Framework 2.0, 3.0**

The following procedures support the main tasks in [Using System.Security.Cryptography.Pkcs](https://msdn.microsoft.com/en-us/library/ms180955(v=vs.85).aspx).

**In This Section**

[How to: Export and Import a Public Key Certificate](https://msdn.microsoft.com/en-us/library/ms180963(v=vs.85).aspx)

Describes how to facilitate the setup of the encryption examples by exporting a public key certificate from the My certificate store and importing it into the AddressBook certificate store.

[How to: Add an Authenticated Attribute to a Signed Message](https://msdn.microsoft.com/en-us/library/ms180964(v=vs.85).aspx)

Creates a CMS/PKCS #7 signed message. The message is signed by a single signer. The message includes a time stamp as an authenticated attribute.

#### How to: Export and Import a Public Key Certificate

**.NET Framework 2.0, 3.0**

[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635736413806036745

This procedure describes how to facilitate the setup of the encryption examples. This is done by exporting a public key certificate from the My certificate store, without exporting the private key, and then importing that public key certificate into the AddressBook certificate store.

To set up the following examples, first follow the setup instructions in that example topic. Those instructions suggest the use of the procedure in this topic to position one or more of the generated public key certificates into the AddressBook certificate store.

* [How to: Envelope a Message for One Recipient](https://msdn.microsoft.com/en-us/library/ms180959(v=vs.85).aspx)
* [How to: Envelope a Message for Multiple Recipients](https://msdn.microsoft.com/en-us/library/ms180960(v=vs.85).aspx)
* [How to: Sign and Envelop a Message](https://msdn.microsoft.com/en-us/library/ms180961(v=vs.85).aspx)

To run the preceding examples on a single computer, the following is required:

* Message recipient's key certificate in both the AddressBook certificate store and the My certificate store
* Message recipient's private key be on that computer and associated with the public key certificate in the My certificate store

The examples first act as the sender of an encrypted message and later as the recipient of that same encrypted message. These two roles would usually be assumed by different entities that have unique public key credentials, often on different computers. When acting as the sender of an encrypted message, the examples require that the recipient's certificate be in the AddressBook certificate store. When acting as the recipient of an encrypted message, the examples require that the recipient's certificate be in the My certificate store, and that the corresponding private key be on that computer.

Set up these examples by using the Makecert.exe utility, one of several ways to do so. [Certificate Creation Tool (Makecert.exe)](https://msdn.microsoft.com/en-us/library/bfsktky3(v=vs.85).aspx) is a convenient utility for generating test certificates. In a production environment, certificates are generated by a certification authority.

The following **Makecert** command generates the required public key certificate and the private key of an entity with the certificate subject name "Recipient1". The certificate is placed in the My certificate store.

**Makecert -n "CN=Recipient1" -ss My**

The following procedure shows how to set up the certificate stores to satisfy these requirements. First, the recipient's certificate and private key are generated, and the certificate is stored in the My certificate store. The certificate is then exported from the My store and imported into the AddressBook store.

**To export the public key certificate from the My certificate store**

1. Open Internet Explorer.
2. On the **Tools** menu, click **Internet Options**, and then click the **Content** tab.
3. Click the **Certificates** button.
4. On the **Personal** tab, select the certificate where "Recipient1" is listed under **Issued To**. (The **Personal** tab lists the certificates in the My certificate store.)
5. Click **Export**. (This opens the export wizard.) Click **Next**.
6. Click **No, do not export the private key**, and then click **Next**.
7. Click **Next** to accept the default export format.
8. Type or browse to a file name for the exported certificate, click **Next**, and then click **Finish**.

**To import the certificate into the AddressBook certificate store**

1. In the **Certificates** dialog box, click the **Other People** tab. (The **Other People** tab lists certificates in the AddressBook certificate store.)
2. Click **Import**, and then click **Next**.
3. Type or browse to the location of the file name where the exported certificate was stored, and then click **Next**.
4. Click **Next** to accept the placement of the imported certificate in the Other People certificate store. Click **Finish**, and then click **OK**.

#### How to: Add an Authenticated Attribute to a Signed Message

**.NET Framework 2.0, 3.0**

This example creates a CMS/PKCS #7 signed message by using [System.Security.Cryptography.Pkcs](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs(v=vs.85).aspx). The message is signed by a single signer. The message includes a timestamp as an authenticated attribute. This means that both the message content and the timestamp are signed. The signature on that message is then verified, which verifies that both the message content and the timestamp are authentic.

**Example**

This example also illustrates the use of a detached CMS/PKCS #7 message. This means that the message content is not stored in the CMS/PKCS #7 message. For this reason, the message content must be passed into the method that verifies the CMS/PKCS #7 message.

This example uses the following classes:

* [CmsSigner](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.cmssigner(v=vs.85).aspx)
* [ContentInfo](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.contentinfo(v=vs.85).aspx)
* [SignedCms](https://msdn.microsoft.com/en-us/library/system.security.cryptography.pkcs.signedcms(v=vs.85).aspx)
* [X509Certificate2](https://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.x509certificate2(v=vs.85).aspx)
* [X509Certificate2Collection](https://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.x509certificate2collection(v=vs.85).aspx)
* [X509Store](https://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.x509store(v=vs.85).aspx)

The following example requires that a public key certificate with the subject name "MessageSigner1" be contained in the My certificate store, and that the associated private key exists.

|  |
| --- |
| **NoteNote** |
| This example is only for illustrative purposes. Production environments might use a different model in which the sender and the recipient of the message execute in different processes with their unique public key credentials. |

Set up this example by using the Makecert.exe utility, one of several ways to do so. [Certificate Creation Tool (Makecert.exe)](https://msdn.microsoft.com/en-us/library/bfsktky3(v=vs.85).aspx) is a convenient utility for generating test certificates. In a production environment, certificates are generated by a certification authority.

The following **Makecert** command generates the required public key certificates and private keys.

**Makecert -n "CN=MessageSigner1" -ss My**

C#

// Copyright (c) Microsoft Corporation. All rights reserved.

#region Using directives

using System;

using System.Security.Cryptography;

using System.Security.Cryptography.Pkcs;

using System.Security.Cryptography.X509Certificates;

using System.Text;

#endregion

namespace AddAnAuthenticatedAttributeToASignedMessage

{

class SignedCmsAuthenticatedAttribute

{

const String signerName = "MessageSigner1";

static void Main(string[] args)

{

Console.WriteLine("System.Security.Cryptography.Pkcs " +

"Sample: Single-signer signed and verified message");

// Original message.

const String msg = "This is the message to be signed. " +

"A time stamp is included as an authenticated " +

"attribute.";

Console.WriteLine("\nOriginal message (len {0}): {1} ",

msg.Length, msg);

// Convert message to array of bytes for signing.

Encoding unicode = Encoding.Unicode;

byte[] msgBytes = unicode.GetBytes(msg);

Console.WriteLine("\n\n------------------------------");

Console.WriteLine(" SETUP OF CREDENTIALS ");

Console.WriteLine("------------------------------\n");

X509Certificate2 signerCert = GetSignerCert();

Console.WriteLine("\n\n----------------------");

Console.WriteLine(" SENDER SIDE ");

Console.WriteLine("----------------------\n");

byte[] encodedSignedCms = SignMsg(msgBytes, signerCert);

Console.WriteLine("\n\n------------------------");

Console.WriteLine(" RECIPIENT SIDE ");

Console.WriteLine("------------------------\n");

if (VerifyMsg(msgBytes, encodedSignedCms))

{

Console.WriteLine("\nMessage verified");

}

else

{

Console.WriteLine("\nMessage failed to verify");

}

}

// Open the My (or Personal) certificate store and search for

// credentials to sign the message with. The certificate

// must have the subject name "MessageSigner1".

static public X509Certificate2 GetSignerCert()

{

// Open the My certificate store.

X509Store storeMy = new X509Store(StoreName.My,

StoreLocation.CurrentUser);

storeMy.Open(OpenFlags.ReadOnly);

// Display certificates to help troubleshoot

// the example's setup.

Console.WriteLine("Found certs with the following subject " +

"names in the {0} store:", storeMy.Name);

foreach (X509Certificate2 cert in storeMy.Certificates)

{

Console.WriteLine("\t{0}", cert.SubjectName.Name);

}

// Find the signer's certificate.

X509Certificate2Collection certColl =

storeMy.Certificates.Find(X509FindType.FindBySubjectName,

signerName, false);

Console.WriteLine(

"Found {0} certificates in the {1} store with name {2}",

certColl.Count, storeMy.Name, signerName);

// Check to see if the certificate suggested by the example

// requirements is not present.

if (certColl.Count == 0)

{

Console.WriteLine(

"A suggested certificate to use for this example " +

"is not in the certificate store. Select " +

"an alternate certificate to use for " +

"signing the message.");

}

storeMy.Close();

// If more than one matching cert, return the first one.

return certColl[0];

}

// Sign the message with the private key of the signer.

static public byte[] SignMsg(

Byte[] msg,

X509Certificate2 signerCert)

{

// Place message in a ContentInfo object.

// This is required to build a SignedCms object.

ContentInfo contentInfo = new ContentInfo(msg);

// Instantiate SignedCms object with the ContentInfo above.

// Has default SubjectIdentifierType IssuerAndSerialNumber.

// Set the Detached property value to true, so message is

// not included in the encoded SignedCms.

SignedCms signedCms = new SignedCms(contentInfo, true);

// Formulate a CmsSigner object for the signer.

CmsSigner cmsSigner = new CmsSigner(signerCert);

// Add an authenticated time stamp attribute to the signer.

// The signing time is the current time.

cmsSigner.SignedAttributes.Add(new Pkcs9SigningTime());

// Sign the CMS/PKCS #7 message.

Console.Write("Computing signature with signer subject " +

"name {0} ... ", signerCert.SubjectName.Name);

signedCms.ComputeSignature(cmsSigner);

Console.WriteLine("Done.");

// Encode the CMS/PKCS #7 message.

return signedCms.Encode();

}

// Verify the encoded SignedCms message and return a Boolean

// value that specifies whether the verification was successful.

static public bool VerifyMsg(byte[] msgBytes, byte[] encodedSignedCms)

{

Pkcs9SigningTime st = new Pkcs9SigningTime();

// Build a ContentInfo object with the message bytes. This

// is necessary because the message is detached from the

// SignedCms object.

ContentInfo contentInfo = new ContentInfo(msgBytes);

// Prepare an object in which to decode and verify.

SignedCms signedCms = new SignedCms(contentInfo, true);

signedCms.Decode(encodedSignedCms);

// Catch a verification exception if you want to

// advise the message recipient that

// security actions might be appropriate.

try

{

// Verify signature. Do not validate signer

// certificate for the purposes of this example.

// Note that in a production environment, validating

// the signer certificate chain will probably

// be necessary.

Console.Write("Checking signature on message ... ");

signedCms.CheckSignature(true);

Console.WriteLine("Done.");

// Report the signing time for the CMS/PKCS #7 message.

for (int i = 0; i < signedCms.SignerInfos[0].SignedAttributes.Count; i++)

{

//if (signedCms.SignerInfos[0].SignedAttributes[i].

//Values[0].GetType().Equals(st.GetType()))

if (signedCms.SignerInfos[0].SignedAttributes[i].Values[0] is Pkcs9SigningTime)

{

Pkcs9SigningTime signingTime = (Pkcs9SigningTime)signedCms.SignerInfos[0].SignedAttributes[i].Values[0];

Console.WriteLine("Signing time: {0}", signingTime.SigningTime);

}

}

}

catch (System.Security.Cryptography.CryptographicException e)

{

Console.WriteLine("VerifyMsg caught exception: {0}",

e.Message);

Console.WriteLine("Verification of the signed PKCS #7 " +

"failed. The message, signatures, " +

" countersignatures, or authenticated attributes " +

" may have been modified in transit or storage. The " +

" message signers or countersigners may not be who " +

" they claim to be. The message's authenticity or " +

" integrity, or both, are not guaranteed.");

return false;

}

return true;

}

}

}

## Windows Identity Foundation

**.NET Framework 4.6 and 4.5**

* [What's New in Windows Identity Foundation 4.5](https://msdn.microsoft.com/en-us/library/hh873305(v=vs.110).aspx)
* [Windows Identity Foundation 4.5 Overview](https://msdn.microsoft.com/en-us/library/hh291066(v=vs.110).aspx)
  + [Claims-Based Identity Model](https://msdn.microsoft.com/en-us/library/hh873308(v=vs.110).aspx)
  + [Claims Based Authorization Using WIF](https://msdn.microsoft.com/en-us/library/hh545448(v=vs.110).aspx)
  + [WIF Claims Programming Model](https://msdn.microsoft.com/en-us/library/hh873304(v=vs.110).aspx)
* [Getting Started With WIF](https://msdn.microsoft.com/en-us/library/hh545426(v=vs.110).aspx)
  + [Building My First Claims-Aware ASP.NET Web Application](https://msdn.microsoft.com/en-us/library/hh545401(v=vs.110).aspx)
  + [Building My First Claims-Aware WCF Service](https://msdn.microsoft.com/en-us/library/hh545447(v=vs.110).aspx)
* [WIF Features](https://msdn.microsoft.com/en-us/library/hh545417(v=vs.110).aspx)
  + [Identity and Access Tool for Visual Studio 2012](https://msdn.microsoft.com/en-us/library/hh545418(v=vs.110).aspx)
  + [WIF Session Management](https://msdn.microsoft.com/en-us/library/hh873347(v=vs.110).aspx)
  + [WIF and Web Farms](https://msdn.microsoft.com/en-us/library/hh545457(v=vs.110).aspx)
  + [WSFederation Authentication Module Overview](https://msdn.microsoft.com/en-us/library/jj191638(v=vs.110).aspx)
  + [WSTrustChannelFactory and WSTrustChannel](https://msdn.microsoft.com/en-us/library/hh873343(v=vs.110).aspx)
* [WIF How-To's Index](https://msdn.microsoft.com/en-us/library/hh362520(v=vs.110).aspx)
  + [How To: Build Claims-Aware ASP.NET MVC Web Application Using WIF](https://msdn.microsoft.com/en-us/library/hh291061(v=vs.110).aspx)
  + [How To: Build Claims-Aware ASP.NET Web Forms Application Using WIF](https://msdn.microsoft.com/en-us/library/hh987037(v=vs.110).aspx)
  + [How To: Build Claims-Aware ASP.NET Application Using Forms-Based Authentication](https://msdn.microsoft.com/en-us/library/hh291068(v=vs.110).aspx)
  + [How To: Build Claims-Aware ASP.NET Application Using Windows Authentication](https://msdn.microsoft.com/en-us/library/hh987035(v=vs.110).aspx)
  + [How To: Debug Claims-Aware Applications And Services Using WIF Tracing](https://msdn.microsoft.com/en-us/library/hh291063(v=vs.110).aspx)
  + [How To: Display Signed In Status Using WIF](https://msdn.microsoft.com/en-us/library/hh874923(v=vs.110).aspx)
  + [How To: Enable Token Replay Detection](https://msdn.microsoft.com/en-us/library/jj161103(v=vs.110).aspx)
  + [How To: Enable WIF Tracing](https://msdn.microsoft.com/en-us/library/jj161102(v=vs.110).aspx)
  + [How To: Enable WIF for a WCF Web Service Application](https://msdn.microsoft.com/en-us/library/jj161104(v=vs.110).aspx)
  + [How To: Transform Incoming Claims](https://msdn.microsoft.com/en-us/library/hh987036(v=vs.110).aspx)
* [WIF Guidelines](https://msdn.microsoft.com/en-us/library/jj157088(v=vs.110).aspx)
  + [Guidelines for Migrating an Application Built Using WIF 3.5 to WIF 4.5](https://msdn.microsoft.com/en-us/library/jj157089(v=vs.110).aspx)
  + [Namespace Mapping between WIF 3.5 and WIF 4.5](https://msdn.microsoft.com/en-us/library/jj157091(v=vs.110).aspx)
* [WIF Code Sample Index](https://msdn.microsoft.com/en-us/library/hh545413(v=vs.110).aspx)
* [WIF Extensions](https://msdn.microsoft.com/en-us/library/dn205066(v=vs.110).aspx)
* [WIF API Reference](https://msdn.microsoft.com/en-us/library/jj729788(v=vs.110).aspx)
* [WIF Configuration Reference](https://msdn.microsoft.com/en-us/library/jj729787(v=vs.110).aspx)
  + [WIF Configuration Schema Conventions](https://msdn.microsoft.com/en-us/library/hh598928(v=vs.110).aspx)

### What's New in Windows Identity Foundation 4.5

**.NET Framework 4.6 and 4.5**

The first version of Windows Identity Foundation (WIF) shipped as a standalone download and is known as WIF 3.5 because it was introduced in the .NET 3.5 SP1 timeframe. Starting with .NET 4.5, WIF is part of the .NET framework. Having the WIF classes directly available in the framework itself allows for a much deeper integration of claims-based identity in the .NET platform, making it easier to use claims. Applications written for WIF 3.5 will need to be modified in order to take advantage of the new model; for information, see [Guidelines for Migrating an Application Built Using WIF 3.5 to WIF 4.5](https://msdn.microsoft.com/en-us/library/jj157089(v=vs.110).aspx).

Below you can find some highlights of the main changes.

[**WIF Is Now Part of the .NET Framework**](javascript:void(0))

WIF classes are now spread across several assemblies, the main ones being **mscorlib**, **System.IdentityModel**,**System.IdentityModel.Services**, and **System.ServiceModel**. Likewise, the WIF classes are spread across several namespaces:[System.Security.Claims](https://msdn.microsoft.com/en-us/library/system.security.claims(v=vs.110).aspx), several [System.IdentityModel](http://go.microsoft.com/fwlink/?LinkId=272004) namespaces, and [System.ServiceModel.Security](https://msdn.microsoft.com/en-us/library/system.servicemodel.security(v=vs.110).aspx). The [System.Security.Claims](https://msdn.microsoft.com/en-us/library/system.security.claims(v=vs.110).aspx)namespace contains the new [ClaimsPrincipal](https://msdn.microsoft.com/en-us/library/system.security.claims.claimsprincipal(v=vs.110).aspx) and [ClaimsIdentity](https://msdn.microsoft.com/en-us/library/system.security.claims.claimsidentity(v=vs.110).aspx) classes (see below). All principals in .NET now derive from [ClaimsPrincipal](https://msdn.microsoft.com/en-us/library/system.security.claims.claimsprincipal(v=vs.110).aspx). For more detailed information about the WIF namespaces and the kinds of classes that they contain, see [WIF API Reference](https://msdn.microsoft.com/en-us/library/jj729788(v=vs.110).aspx). For information about how namespaces map between WIF 3.5 and WIF 4.5, see [Namespace Mapping between WIF 3.5 and WIF 4.5](https://msdn.microsoft.com/en-us/library/jj157091(v=vs.110).aspx).

[**New Claims Model and Principal Object**](javascript:void(0))

Claims are at the very core of the .NET Framework 4.5. The base claim classes ([Claim](https://msdn.microsoft.com/en-us/library/system.security.claims.claim(v=vs.110).aspx), [ClaimsIdentity](https://msdn.microsoft.com/en-us/library/system.security.claims.claimsidentity(v=vs.110).aspx), [ClaimsPrincipal](https://msdn.microsoft.com/en-us/library/system.security.claims.claimsprincipal(v=vs.110).aspx), [ClaimTypes](https://msdn.microsoft.com/en-us/library/system.security.claims.claimtypes(v=vs.110).aspx), and[ClaimValueTypes](https://msdn.microsoft.com/en-us/library/system.security.claims.claimvaluetypes(v=vs.110).aspx)) all live directly in **mscorlib** in the [System.Security.Claims](https://msdn.microsoft.com/en-us/library/system.security.claims(v=vs.110).aspx) namespace. It is no longer necessary to use interfaces in order to plug claims into the .NET identity system: [WindowsPrincipal](https://msdn.microsoft.com/en-us/library/system.security.principal.windowsprincipal(v=vs.110).aspx), [GenericPrincipal](https://msdn.microsoft.com/en-us/library/system.security.principal.genericprincipal(v=vs.110).aspx), and [RolePrincipal](https://msdn.microsoft.com/en-us/library/system.web.security.roleprincipal(v=vs.110).aspx) now inherit from [ClaimsPrincipal](https://msdn.microsoft.com/en-us/library/system.security.claims.claimsprincipal(v=vs.110).aspx); and[WindowsIdentity](https://msdn.microsoft.com/en-us/library/system.security.principal.windowsidentity(v=vs.110).aspx), [GenericIdentity](https://msdn.microsoft.com/en-us/library/system.security.principal.genericidentity(v=vs.110).aspx), and [FormsIdentity](https://msdn.microsoft.com/en-us/library/system.web.security.formsidentity(v=vs.110).aspx) now inherit from [ClaimsIdentity](https://msdn.microsoft.com/en-us/library/system.security.claims.claimsidentity(v=vs.110).aspx). In short, every principal class will now serve claims. The WIF 3.5 integration classes and interfaces (**WindowsClaimsIdentity**, **WindowsClaimsPrincipal**, **IClaimsPrincipal**, **IClaimsIdentity**) have thus been removed. In addition, the [ClaimsIdentity](https://msdn.microsoft.com/en-us/library/system.security.claims.claimsidentity(v=vs.110).aspx) class now exposes methods which make it easier to query the identity’s claims collection.

[**Changes to the WIF 4.5 Visual Studio Experience**](javascript:void(0))

* The **Add STS Reference …** Visual Studio functionality (cmdline utility FedUtil) no longer exists; instead you can use the new Visual Studio extension **Identity and Access Tool for Visual Studio 2012**. This allows you to federate with an existing STS or use LocalSTS to test your solutions. After installing the extension you can right-click on your project and look for **Identity and Access** in the context menu.
* The ASP.NET and STS templates are no longer provided as claims can be used directly in existing project templates for ASP.Net, web sites, and WCF.
* The controls in the **Microsoft.IdentityModel.Web.Controls** namespace (**SignInControl**, **FederatedPassiveSignInControl**, and**FederatedPassiveSignInStatus**) are not carried over into WIF 4.5.

[**Changes to the WIF 4.5 API**](javascript:void(0))

* In general, claims related classes are in the [System.Security.Claims](https://msdn.microsoft.com/en-us/library/system.security.claims(v=vs.110).aspx) namespace; WCF related classes –- service contracts, channels, channel factories, and service hosts that are used for WS-Trust scenarios -- are in [System.ServiceModel.Security](https://msdn.microsoft.com/en-us/library/system.servicemodel.security(v=vs.110).aspx); and all other WIF classes are spread across various [System.IdentityModel](http://go.microsoft.com/fwlink/?LinkId=272004) namespaces – these include, for example, classes that represent WS-\* and SAML artifacts, token handlers and related classes, and classes used in WS-Federation scenarios. For more information, see[Namespace Mapping between WIF 3.5 and WIF 4.5](https://msdn.microsoft.com/en-us/library/jj157091(v=vs.110).aspx) and [WIF API Reference](https://msdn.microsoft.com/en-us/library/jj729788(v=vs.110).aspx).
* Machine key has been enabled for use in session cookies for web farm scenarios. For more information, see [WIF and Web Farms](https://msdn.microsoft.com/en-us/library/hh545457(v=vs.110).aspx).
* You now declaratively configure WIF under the [<system.identityModel>](https://msdn.microsoft.com/en-us/library/hh568638(v=vs.110).aspx) and [<system.identityModel.services>](https://msdn.microsoft.com/en-us/library/hh568674(v=vs.110).aspx) elements. For more information about WIF configuration, see [WIF Configuration Reference](https://msdn.microsoft.com/en-us/library/jj729787(v=vs.110).aspx).

[**Other notable .NET changes or features that are caused by the integration of WIF into .NET**](javascript:void(0))

* The potential for performing claims-based authorization (CBAC) is now integral to the .NET framework. You can configure a[ClaimsAuthorizationManager](https://msdn.microsoft.com/en-us/library/system.security.claims.claimsauthorizationmanager(v=vs.110).aspx) object and then use the [ClaimsPrincipalPermission](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.claimsprincipalpermission(v=vs.110).aspx) and [ClaimsPrincipalPermissionAttribute](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.claimsprincipalpermissionattribute(v=vs.110).aspx) classes to perform imperative and declarative access checks in your code. CBAC provides more flexibility and greater granularity than traditional role-based access checks (RBAC). It also allows greater separation of authorization policy from business logic because the business logic can base the access check on a specific claim or set of claims and the authorization policy for those claims can be configured declaratively under the [<claimsAuthorizationManager>](https://msdn.microsoft.com/en-us/library/hh568658(v=vs.110).aspx) element.

[**WCF changes as a result of WIF integration:**](javascript:void(0))

* The WCF claims-based identity model is superseded by WIF. This means that classes in the [System.IdentityModel.Claims](https://msdn.microsoft.com/en-us/library/system.identitymodel.claims(v=vs.110).aspx),[System.IdentityModel.Policy](https://msdn.microsoft.com/en-us/library/system.identitymodel.policy(v=vs.110).aspx), and [System.IdentityModel.Selectors](https://msdn.microsoft.com/en-us/library/system.identitymodel.selectors(v=vs.110).aspx) namespaces should be dropped in favor of using WIF classes.
* WIF is now enabled on a WCF service by specifying the **useIdentityConfiguration** attribute on the**<system.serviceModel>**/**<behaviors>**/**<serviceBehaviors>**/**<serviceCredentials>** element as in the following XML:

Xml

<serviceCredentials useIdentityConfiguration="true">

<!--Certificate added by Identity And Access VS Package. Subject='CN=localhost', Issuer='CN=localhost'. Make sure you have this certificate installed. The Identity and Access tool does not install this certificate.-->

<serviceCertificate findValue="CN=localhost" storeLocation="LocalMachine" storeName="My" x509FindType="FindBySubjectDistinguishedName" />

</serviceCredentials>

When you use the **Identity and Access Tool for Visual Studio 2012** (see **Changes to the Visual Studio Experience** above), the tool adds a **<serviceCredentials>** element with the **useIdentityConfiguration** attribute set to the configuration file for you. It also adds a corresponding [<system.identityModel>](https://msdn.microsoft.com/en-us/library/hh568638(v=vs.110).aspx) element that contains the WIF configuration settings and adds a binding and other settings necessary to outsource authentication to your chosen STS.

### Windows Identity Foundation 4.5 Overview

**.NET Framework 4.6 and 4.5**

Windows Identity Foundation 4.5 is a set of .NET Framework classes for implementing claims-based identity in your applications. By using it, you’ll more easily reap the benefits of claims-aware applications and services. WIF 4.5 can be used in any Web application or Web service that uses the .NET Framework version 4.5 or later. WIF is just one part of Microsoft’s Federated Identity software family that implements the shared industry vision based on open standards. Federated Identity comprises three components: [Active Directory® Federation Services](http://go.microsoft.com/fwlink/?LinkID=247516) (AD FS) 2.0,[Windows Azure Access Control Services](http://go.microsoft.com/fwlink/?LinkID=247517) (ACS), and WIF. Together, these three components form the core of Microsoft’s new claims-based cloud identity and access platform.

For more information about WIF, see the [Windows Identity Foundation Web site](http://go.microsoft.com/fwlink/?LinkId=149009) (http://go.microsoft.com/fwlink/?LinkId=149009) at the Security Developer Center on MSDN. For an introduction to creating applications using WIF, see [Programming Windows Identity Foundation](http://go.microsoft.com/fwlink/?LinkId=210158)(http://go.microsoft.com/fwlink/?LinkId=210158) by Vittorio Bertocci (published by Microsoft Press).

[**WIF 4.5 Features**](javascript:void(0))

WIF 4.5 is a framework for building identity-aware applications. The framework abstracts the WS-Trust and WS-Federation protocols and presents developers with APIs for building claims-aware applications and, if needed, security token services (STS)s. Applications can use WIF to process tokens issued from STSs, such as AD FS 2.0 and ACS, and make identity-based decisions at the web application or web service.

WIF 4.5 has the following major features:

* Build claims-aware applications (relying party applications). WIF helps developers build claims-aware applications. In addition to providing a claims model, it provides application developers with a rich set of APIs to help making user access decisions based on claims. WIF also provides developers with a consistent programming experience whether they choose to build their applications in ASP.NET or in WCF environments.
* Build identity delegation support into claims-aware applications. WIF offers the capability of maintaining the identities of original requestors across the multiple service boundaries. This capability can be achieved by either using the "ActAs" or the "OnBehalfOf" functionality in the framework and it offers developers the ability to add identity delegation support into their claims-aware applications.
* Build custom STSs. WIF makes it substantially easier to build a custom STS that supports the WS-Trust protocol. Such an STS is also referred to as an Active STS.

In addition, the framework also provides support for building an STS that supports WS-Federation to enable Web browser clients. Such an STS is also referred to as a Passive STS.

* New identity and access tool for Visual Studio 11 that enables you to secure your application with claims based identity and accept users from multiple identity providers. You can download this WIF tool from the following URL: <http://go.microsoft.com/fwlink/?LinkID=245849> or directly from within Visual Studio 11 by searching for “identity” directly in the Extensions Manager. For more information, see [Identity and Access Tool for Visual Studio 2012](https://msdn.microsoft.com/en-us/library/hh545418(v=vs.110).aspx).

WIF supports the following major scenarios:

* Federation. WIF makes it possible to enable federation between two or more partners. Its support for building claims-aware applications (RPs) and custom STSs helps developers achieve this scenario.
* Identity Delegation. WIF makes it easy to maintain the identities across the service boundaries so that developers can achieve an identity delegation scenario.
* Step-up Authentication. Authentication requirements for different resources within an application may vary. WIF provides developers the ability to build applications that can require incremental authentication requirements (for example: initial login with Username/Password authentication and then step-up to Smart Card authentication).

By using WIF, you’ll more easily reap the benefits of the claims-based identity model. For more information, see [Windows Identity Foundation White Paper for Developers](http://go.microsoft.com/fwlink/?LinkId=122266).

#### Claims-Based Identity Model

**.NET Framework 4.6 and 4.5**

When you build claims-aware applications, the user identity is represented in your application as a set of claims. One claim could be the user’s name, another might be an e-mail address. The idea is that an external identity system is configured to give your application everything it needs to know about the user with each request she makes, along with cryptographic assurance that the identity data you receive comes from a trusted source.

Under this model, single sign-on is much easier to achieve, and your application is no longer responsible for the following:

* Authenticating users.
* Storing user accounts and passwords.
* Calling to enterprise directories to look up user identity details.
* Integrating with identity systems from other platforms or companies.

Under this model, your application makes identity-related decisions based on claims supplied by the system that authenticated your user. This could be anything from simple application personalization with the user’s first name, to authorizing the user to access higher valued features and resources in your application.

This topic provides the following information:

* [Introduction to Claims-Based Identity](https://msdn.microsoft.com/en-us/library/hh873308(v=vs.110).aspx#bkmk_1)
* [Basic Scenario for a Claims-Based Identity Model](https://msdn.microsoft.com/en-us/library/hh873308(v=vs.110).aspx#bkmk_2)

[**Introduction to Claims-Based Identity**](javascript:void(0))

The following terminology and concepts can help you understand this new architecture for identity.

[**Identity**](javascript:void(0))

For the purposes of describing the programming model in Windows Identity Foundation (WIF), we will use the term “identity” to represent a set of attributes that describe a user or some other entity in a system that you want to secure.

[**Claim**](javascript:void(0))

Think of a claim as a piece of identity information such as name, e-mail address, age, membership in the Sales role. The more claims your application receives, the more you’ll know about your user. You may be wondering why these are called “claims,” rather than “attributes,” as is commonly used in describing enterprise directories. The reason has to do with the delivery method. In this model, your application doesn’t look up user attributes in a directory. Instead, the user delivers claims to your application, and your application examines them. Each claim is made by an issuer, and you trust the claim only as much as you trust the issuer. For example, you trust a claim made by your company’s domain controller more than you trust a claim made by the user herself. WIF represents claims with a [Claim](https://msdn.microsoft.com/en-us/library/system.security.claims.claim(v=vs.110).aspx) type, which has an [Issuer](https://msdn.microsoft.com/en-us/library/hh194503(v=vs.110).aspx) property that allows you to find out who issued the claim.

[**Security Token**](javascript:void(0))

The user delivers a set of claims to your application along with a request. In a Web service, these claims are carried in the security header of the SOAP envelope. In a browser-based Web application, the claims arrive through an HTTP POST from the user’s browser, and may later be cached in a cookie if a session is desired. Regardless of how these claims arrive, they must be serialized, which is where security tokens come in. A security token is a serialized set of claims that is digitally signed by the issuing authority. The signature is important: it gives you assurance that the user didn’t just make up a bunch of claims and send them to you. In low security situations where cryptography isn’t necessary or desired, you can use unsigned tokens, but that scenario is not described in this topic.

One of the core features in WIF is the ability to create and read security tokens. WIF and the .NET Framework handle all of the cryptographic work, and present your application with a set of claims that you can read.

[**Issuing Authority**](javascript:void(0))

There are lots of different types of issuing authorities, from domain controllers that issue Kerberos tickets, to certification authorities that issue X.509 certificates, but the specific type of authority discussed in this topic issues security tokens that contain claims. This issuing authority is a Web application or Web service that knows how to issue security tokens. It must have enough knowledge to be able to issue the proper claims given the target relying party and the user making the request, and might be responsible for interacting with user stores to look up claims and authenticate the users themselves.

Whatever issuing authority you choose, it plays a central role in your identity solution. When you factor authentication out of your application by relying on claims, you’re passing responsibility to that authority and asking it to authenticate users on your behalf.

[**Security Token Service (STS)**](javascript:void(0))

A security token service (STS) is the service component that builds, signs, and issues security tokens according to the WS-Trust and WS-Federation protocols. There’s a lot of work that goes into implementing these protocols, but WIF does all of this work for you, making it feasible for someone who isn’t an expert in the protocols to get an STS up and running with very little effort. You can use a pre-built STS such as [Active Directory® Federation Services (AD FS) 2.0](http://go.microsoft.com/fwlink/?LinkID=247516), a cloud STS such as a [Windows Azure Access Control Service (ACS)](http://go.microsoft.com/fwlink/?LinkID=247517), or, if you want to issue custom tokens or provide custom authentication or authorization, you can build your own custom STS using WIF. WIF makes it easy to build your own STS.

[**Relying Party Application**](javascript:void(0))

When you build an application that relies on claims, you are building a relying party (RP) application. Synonyms for an RP include “claims-aware application” and “claims-based application”. Web applications and Web services can both be RPs. A RP application consumes the tokens issued by a STS and extracts the claims from tokens to use them for identity related tasks. WIF offers functionalities to help you build RP applications.

[**Standards**](javascript:void(0))

In order to make all of this interoperable, several WS-\* standards are used in the previous scenario. Policy is retrieved using WS-MetadataExchange, and the policy itself is structured according to the WS-Policy specification. The STS exposes endpoints that implement the WS-Trust specification, which describes how to request and receive security tokens. Most STSs today issue tokens formatted with Security Assertion Markup Langauge (SAML). SAML is an industry-recognized XML vocabulary that can be used to represent claims in an interoperable way. Or, in a multi-platform situation, this allows you to communicate with an STS on an entirely different platform and achieve single sign-on across all of your applications, regardless of platform.

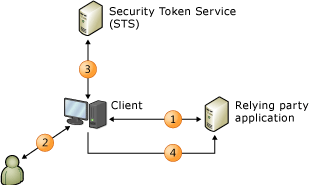
[**Browser-Based Applications**](javascript:void(0))

Smart clients aren’t the only ones who can use the claims-based identity model. Browser-based applications (also referred to as passive clients) can use it as well. The following scenario describes how this works.

First, the user points a browser at a claims-aware Web application (the relying party application). The Web application redirects the browser to the STS so the user can be authenticated. The STS is hosted in a simple web application that reads the incoming request, authenticates the user using standard HTTP mechanisms, and then creates a SAML token and replies with a piece of JavaScript code that causes the browser to initiate an HTTP POST that sends the SAML token back to the RP. The body of this POST contains the claims that the RP requested. At this point, it is common for the RP to package the claims into a cookie so that the user doesn’t have to be redirected for each request.

[**Basic Scenario for a Claims-Based Identity Model**](javascript:void(0))

The following is an example of a claims-based system.



This diagram shows a Web site (the relying party application, RP) that has been configured to use WIF for authentication and a client, a web browser, that wants to use that site.

1. When an unauthenticated user requests a page their browser is redirected to the identity provider (IP) pages.
2. The IP requires the user to present their credentials, e.g. username/password, Kerberos, etc.
3. The IP issues a token back to that is returned to the browser.
4. The browser is now redirected back to the originally requested page where WIF determines if the token satisfies the requirements to access the page. If so a cookie is issued to establish a session so the authentication only needs to occur once, and control is passed to the application.

#### Claims Based Authorization Using WIF

**.NET Framework 4.6 and 4.5**

In a relying party application, authorization determines what resources an authenticated identity is allowed to access and what operations it is allowed to perform on those resources. Improper or weak authorization leads to information disclosure and data tampering. This topic outlines the available approaches to implementing authorization for claims-aware ASP.NET web applications and services using Windows Identity Foundation (WIF) and a Security Token Service (STS), for example, the Windows Azure Access Control Service (ACS).

[**Overview**](javascript:void(0))

Since its first version, the .NET Framework has offered a flexible mechanism for implementing authorization. This mechanism is based on two simple interfaces—**IPrincipal** and **IIdentity**. Concrete implementations of **IIdentity** represent an authenticated user. For example, the**WindowsIdentity** implementation represents a user who is authenticated by Active Directory, and **GenericIdentity** represents a user whose identity is verified via a custom authentication process. Concrete implementations of **IPrincipal** help to check permissions using roles depending on the role store. For example, **WindowsPrincipal** checks **WindowsIdentity** for membership in Active Directory groups. This check is performed by calling the **IsInRole** method on the **IPrincipal** interface. Checking access based on roles is called Role-Based Access Control (RBAC). For more information, see [Role-Based Access Control](https://msdn.microsoft.com/en-us/library/hh545448(v=vs.110).aspx#bkmk_1). Claims can be used to carry information about roles to support familiar, role-based authorization mechanisms.

Claims can also be used to enable more complicated authorization decisions beyond roles. Claims can be based on virtually any information about the user - age, zip code, shoe size, etc. An access control mechanism that is based on arbitrary claims is called claims-based authorization. For more information, see [Claims-based Authorization](https://msdn.microsoft.com/en-us/library/hh545448(v=vs.110).aspx#bkmk_2).

[**Role-Based Access Control**](javascript:void(0))

RBAC is an authorization approach in which user permissions are managed and enforced by an application based on user roles. If a user has a role that is required to perform an action, the access is granted; otherwise, access is denied.

[**IPrincipal.IsInRole Method**](javascript:void(0))

To implement the RBAC approach in claims-aware applications, use the **IsInRole()** method in the **IPrinicpal** interface, just as you would in non-claims-aware applications. There are several ways of using the **IsInRole()** method:

* Explicitly calling on **IPrincipal.IsInRole(“Administrator”)**. In this approach, the outcome is a Boolean. Use it in your conditional statements. It can be used arbitrarily any place in your code.
* Using the security demand **PrincipalPermission.Demand()**. In this approach, the outcome is an exception in case the demand is not satisfied. This should fit your exception handling strategy. Throwing exceptions is much more expensive from a performance perspective compared to retiring Boolean. This can be used any place in your code.
* Using the declarative attributes **[PrincipalPermission(SecurityAction.Demand, Role = “Administrator”)]**. This approach is called declarative, because it is used to decorate methods. It cannot be used in code blocks inside the method’s implementations. The outcome is an exception in case the demand is not satisfied. You should make sure that it fits your exception-handling strategy.
* Using URL authorization, using the **<authorization>** section in **web.config**. This approach is suitable when you are managing authorization on a URL level. This is the most coarse level among those previously mentioned. The advantage of this approach is that changes are made in the configuration file, which means that the code should not be compiled to take advantage of the change.

[**Expressing Roles as Claims**](javascript:void(0))

When the **IsInRole()** method is called, there is a check made to see if the current user has that role. In claims-aware applications, the role is expressed by a role claim type that should be available in the token. The role claim type is expressed using the following URI:

http://schemas.microsoft.com/ws/2008/06/identity/claims/role

There are several ways to enrich a token with a role claim type:

* **During token issuance**. When a user is authenticated the role claim can be issued by the identity provider STS or by a federation drovider such as the Windows Azure Access Control Service (ACS).
* **Transforming arbitrary claims into of claims role type using ClaimsAuthenticationManager**. The ClaimsAuthenticationManager is a component that ships as part of WIF. It allows requests to be intercepted when they launch an application, inspecting tokens and transforming them by adding, changing, or removing claims. For more information about how to use ClaimsAuthenticationManager for transforming claims, see [How To: Implement Role Based Access Control (RBAC) in a Claims Aware ASP.NET Application Using WIF and ACS](http://go.microsoft.com/fwlink/?LinkID=247445) (http://go.microsoft.com/fwlink/?LinkID=247444).
* **Mapping arbitrary claims to a role type using the samlSecurityTokenRequirement configuration section**—A declarative approach where the claims transformation is done using only the configuration and no coding is required.

[**Claims-based Authorization**](javascript:void(0))

Claims-based authorization is an approach where the authorization decision to grant or deny access is based on arbitrary logic that uses data available in claims to make the decision. Recall that in the case of RBAC, the only claim used was role type claim. A role type claim was used to check if the user belongs to specific role or not. To illustrate the process of making the authorization decisions using claims-based authorization approach, consider the following steps:

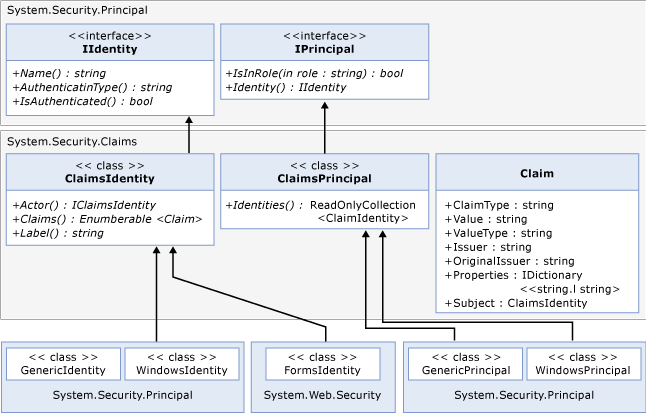
1. The application receives a request that requires the user is authenticated.
2. WIF redirects the user to their identity provider, after they are authenticated the application request is made with an associated security token representing the user containing claims about them. WIF associates those claims with the principal that represents the user.
3. The application passes the claims to the decision logic mechanism. It can be in-memory code, a call to a web service, a query to a database, a sophisticated rules engine, or using the ClaimsAuthorizationManager.
4. The decision mechanism calculates the outcome based on the claims.
5. Access is granted if the outcome is true and denied if it is false. For example, the rule might be that the user is of age 21 or above and lives in Washington State.

[ClaimsAuthorizationManager](https://msdn.microsoft.com/en-us/library/system.security.claims.claimsauthorizationmanager(v=vs.110).aspx) is useful for externalizing the decision logic for claims-based authorization in your applications. ClaimsAuthorizationManager is a WIF component that ships as part of .NET 4.5. ClaimsAuthorizationManager allows you to intercept incoming requests and implement any logic of your choice to make authorization decisions based on the incoming claims. This becomes important when authorization logic needs to be changed. In that case, using ClaimsAuthorizationManager will not affect the application’s integrity, thereby reducing the likelihood of an application error as a result of the change. To learn more about how to use ClaimsAuthorizationManager to implement claims-based access control, see [How To: Implement Claims Authorization in a Claims Aware ASP.NET Application Using WIF and ACS](http://go.microsoft.com/fwlink/?LinkID=247446).

#### WIF Claims Programming Model

**.NET Framework 4.6 and 4.5**

ASP.NET and Windows Communication Foundation (WCF) developers ordinarily use the IIdentity and IPrincipal interfaces to work with the user’s identity information. In .NET 4.5, Windows Identity Foundation (WIF) has been integrated such that claims are now always present for any principal as illustrated in the following diagram:



In .NET 4.5, System.Security.Claims contains the new ClaimsPrincipal and ClaimsIdentity classes (see diagram above). All principals in .NET now derive from ClaimsPrincipal. All built-in identity classes, like FormsIdentity for ASP.NET and WindowsIdentity now derive from ClaimsIdentity. Similarly, all built-in principal classes like GenericPrincipal and WindowsPrincipal derive from ClaimsPrincipal.

A claim is represented by [Claim](https://msdn.microsoft.com/en-us/library/system.security.claims.claim(v=vs.110).aspx) class. This class has the following important properties:

* [Type](https://msdn.microsoft.com/en-us/library/hh159715(v=vs.110).aspx) represents the type of claim and is typically a URI. For example, the e-mail address claim is represented ashttp://schemas.microsoft.com/ws/2008/06/identity/claims/email.
* [Value](https://msdn.microsoft.com/en-us/library/hh194538(v=vs.110).aspx) contains the value of the claim and is represented as a string. For example, the e-mail address can be represented as “someone@contoso.com”.
* [ValueType](https://msdn.microsoft.com/en-us/library/hh194424(v=vs.110).aspx) represents the type of the claim value and is typically a URI. For example, the string type is represented ashttp://www.w3.org/2001/XMLSchema#string. The value type must be a QName according to the XML schema. The value should be of the format namespace#format to enable WIF to output a valid QName value. If the namespace is not a well-defined namespace, the generated XML probably cannot be schema validated, because there will not be a published XSD file for that namespace. The default value type is http://www.w3.org/2001/XMLSchema#string. Please see[http://www.w3.org/2001/XMLSchema](http://go.microsoft.com/fwlink/?LinkId=209155) for well-known value types that you can use safely.
* [Issuer](https://msdn.microsoft.com/en-us/library/hh194503(v=vs.110).aspx) is the identifier of the security token service (STS) that issued the claim. This can be represented as URL of the STS or a name that represents the STS, such as https://sts1.contoso.com/sts.
* [OriginalIssuer](https://msdn.microsoft.com/en-us/library/hh159766(v=vs.110).aspx) is the identifier of the STS that originally issued the claim, regardless of how many STSs are in the chain. This is represented just like [Issuer](https://msdn.microsoft.com/en-us/library/hh194503(v=vs.110).aspx).
* [Subject](https://msdn.microsoft.com/en-us/library/hh159760(v=vs.110).aspx) is the subject whose identity is being examined. It contains a [ClaimsIdentity](https://msdn.microsoft.com/en-us/library/system.security.claims.claimsidentity(v=vs.110).aspx).
* [Properties](https://msdn.microsoft.com/en-us/library/hh194436(v=vs.110).aspx) is a dictionary that lets the developer provide application-specific data to be transferred on the wire together with the other properties, and can be used for custom validation.

[**Identity Delegation**](javascript:void(0))

An important property of [ClaimsIdentity](https://msdn.microsoft.com/en-us/library/system.security.claims.claimsidentity(v=vs.110).aspx) is [Actor](https://msdn.microsoft.com/en-us/library/hh194460(v=vs.110).aspx). This property enables the delegation of credentials in a multi-tier system in which a middle tier acts as the client to make requests to a back-end service.

[**Accessing Claims through Thread.CurrentPrincipal**](javascript:void(0))

To access the current user’s set of claims in an RP application, use Thread.CurrentPrincipal.

The following code sample shows the usage of this method to get a System.Security.Claims.ClaimsIdentity:

ClaimsPrincipal claimsPrincipal = Thread.CurrentPrincipal as ClaimsPrincipal;

For more information, see [System.Security.Claims](https://msdn.microsoft.com/en-us/library/system.security.claims(v=vs.110).aspx).

[**Role Claim Type**](javascript:void(0))

Part of configuring your RP application is to determine what your role claim type should be. This claim type is used by System.Security.Claims.ClaimsPrincipal.IsInRole(System.String). The default claim type ishttp://schemas.microsoft.com/ws/2008/06/identity/claims/role.

[**Claims Extracted by Windows Identity Foundation from Different Token Types**](javascript:void(0))

WIF supports several combinations of authentication mechanisms out of the box. The following table lists the claims that WIF extracts from different token types.

|  |  |  |
| --- | --- | --- |
| Token Type | Claim Generated | Map To Windows Access Token |
| SAML 1.1 | 1. All claims from System.IdentityModel.SecurityTokenService.GetOutputClaimsIdentity(System.Security.Claims.ClaimsPrincipal,System.IdentityModel.Protocols.WSTrust.RequestSecurityToken,System.IdentityModel.Scope). 2. The http://schemas.microsoft.com/ws/2008/06/identity/claims/confirmationkey claim that contains the XML serialization of the confirmation key, if the token contains a proof token. 3. The http://schemas.microsoft.com/ws/2008/06/identity/claims/samlissuername claim from the Issuer element. 4. AuthenticationMethod and AuthenticationInstant claims, if the token contains an authentication statement. | In addition to the claims listed in “SAML 1.1”, except claims of typehttp://schemas.xmlsoap.org/ws/2005/05/identity/claims/name, Windows authentication related claims will be added and the identity will be represented by WindowsClaimsIdentity. |
| SAML 2.0 | Same as “SAML 1.1”. | Same as “SAML 1.1 Mapped to Windows Account”. |
| X509 | 1. Claims with the X500 distinguished name, emailName, dnsName, SimpleName, UpnName, UrlName, thumbprint, RsaKey (this can be extracted using the RSACryptoServiceProvider.ExportParameters method from the X509Certificate2.PublicKey.Key property), DsaKey (this can be extracted using the DSACryptoServiceProvider.ExportParameters method from the X509Certificate2.PublicKey.Key property), SerialNumber properties from the X509 Certificate. 2. AuthenticationMethod claim with value http://schemas.microsoft.com/ws/2008/06/identity/authenticationmethod/x509. AuthenticationInstant claim with the value of the time when the certificate was validated in XmlSchema DateTime format. | 1. It uses the Windows account fully qualified domain name as thehttp://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameclaim value. . 2. Claims from the X509 Certificate not mapped to Windows, and claims from the windows account obtained by mapping the certificate to Windows. |
| UPN | 1. Claims are similar to the claims in the Windows authentication section. 2. AuthenticationMethod claim with value http://schemas.microsoft.com/ws/2008/06/identity/authenticationmethod/password. The AuthenticationInstant claim with the value of the time when the password was validated in XmlSchema DateTime format. |  |
| Windows (Kerberos or NTLM) | 1. Claims generated from the access token such as: PrimarySID, DenyOnlyPrimarySID, PrimaryGroupSID, DenyOnlyPrimaryGroupSID, GroupSID, DenyOnlySID, and Name 2. AuthenticationMethod with the value http://schemas.microsoft.com/ws/2008/06/identity/authenticationmethod/windows. AuthenticationInstant with the value of the time when the Windows access token was created in the XMLSchema DateTime format. |  |
| RSA Key Pair | 1. The http://schemas.xmlsoap.org/ws/2005/05/identity/claims/rsa claim with the value of RSAKeyValue. 2. AuthenticationMethod claim with the value http://schemas.microsoft.com/ws/2008/06/identity/authenticationmethod/signature. AuthenticationInstant claim with the value of the time when the RSA key was authenticated (that is, the signature was verified) in the XMLSchema DateTime format. |  |
| Authentication Type | URI emitted in “AuthenticationMethod” claim |
| Password | urn:oasis:names:tc:SAML:1.0:am:password |
| Kerberos | urn:ietf:rfc:1510 |
| SecureRemotePassword | urn:ietf:rfc:2945 |
| TLSClient | urn:ietf:rfc:2246 |
| X509 | urn:oasis:names:tc:SAML:1.0:am:X509-PKI |
| PGP | urn:oasis:names:tc:SAML:1.0:am:PGP |
| Spki | urn:oasis:names:tc:SAML:1.0:am:SPKI |
| XmlDSig | urn:ietf:rfc:3075 |
| Unspecified | urn:oasis:names:tc:SAML:1.0:am:unspecified |

### Getting Started With WIF

**.NET Framework 4.6 and 4.5**

* [Building My First Claims-Aware ASP.NET Web Application](https://msdn.microsoft.com/en-us/library/hh545401(v=vs.110).aspx)
* [Building My First Claims-Aware WCF Service](https://msdn.microsoft.com/en-us/library/hh545447(v=vs.110).aspx)

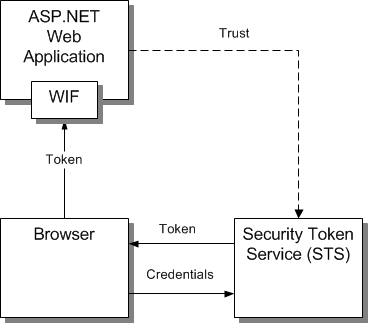
#### Building My First Claims-Aware ASP.NET Web Application

**.NET Framework 4.6 and 4.5**

[**Applies To**](javascript:void(0))

* Windows Identity Foundation (WIF)
* ASP.NET

This topic outlines the scenario of building claims-aware ASP.NET web applications using WIF. There are usually three participants in a claims-aware application scenario: the application itself, the end user, and the Security Token Service (STS). The following figure describes this scenario:



1. The claims-aware application uses WIF to identify unauthenticated requests and to redirect them to the STS.
2. The end user provides credentials to the STS and upon successful authentication the user is issued a token by the STS.
3. The user is redirected from the STS to the claims-aware application with the STS-issued token in the request.
4. The claims-aware application is configured to trust the STS and the tokens it issues. The claims-aware application uses WIF to validate the token and to parse it. Developers use the appropriate WIF API and types, for example, **ClaimsPrincpal** for the application’s needs, such as implementing authorization for it.

Starting from .NET 4.5, WIF is part of the .NET framework package. Having the WIF classes directly available in the framework itself allows a much deeper integration of claims-based identity in the .NET platform, making it easier to use claims. With WIF 4.5, you do not need to install any out-of-band components in order to start developing claims-aware web applications. WIF classes are now spread across various assemblies, the main ones being System.Security.Claims, System.IdentityModel and System.IdentityModel.Services.

STS is a service that issues tokens upon successful authentication. Microsoft offers two industry standard STS’s:

* [Active Directory Federation Services (AD FS) 2.0](http://go.microsoft.com/fwlink/?LinkID=247516) (http://go.microsoft.com/fwlink/?LinkID=247516)
* [Windows Azure Access Control Service (ACS)](http://go.microsoft.com/fwlink/?LinkID=247517) (http://go.microsoft.com/fwlink/?LinkID=247517).

AD FS 2.0 is part of the Windows Server R2 and can be used as an STS for on-premise scenarios. ACS is a cloud service, offered as part of the Windows Azure Platform. For testing or educational purposes, you can also use other STS’s in order to build your claims-aware applications. For example, you can use the Local Development STS that is part of the [Identity and Access Tool for Visual Studio](http://go.microsoft.com/fwlink/?LinkID=245849)(http://go.microsoft.com/fwlink/?LinkID=245849) which is freely available online.

To build your first claims-aware ASP.NET application using WIF, follow the instructions in one of the following:

* [How To: Build Claims-Aware ASP.NET MVC Web Application Using WIF](https://msdn.microsoft.com/en-us/library/hh291061(v=vs.110).aspx)
* [How To: Build Claims-Aware ASP.NET Web Forms Application Using WIF](https://msdn.microsoft.com/en-us/library/hh987037(v=vs.110).aspx)
* [How To: Build Claims-Aware ASP.NET Application Using Forms-Based Authentication](https://msdn.microsoft.com/en-us/library/hh291068(v=vs.110).aspx)

#### Building My First Claims-Aware WCF Service

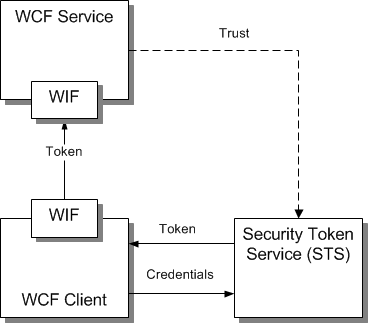
**.NET Framework 4.6 and 4.5**

[**Applies To**](javascript:void(0))

* Windows Identity Foundation (WIF)
* Windows Communication Foundation (WCF)

[**Overview**](javascript:void(0))

This topic outlines the scenario of building claims-aware WCF services using WIF. There are usually three participants in a claims-aware web service scenario: the web service itself, the end user, and the Security Token Service (STS). The following figure describes this scenario:



1. The WCF service client (sometimes called agent) uses WIF to send credentials to the STS and upon successful authentication, the agent is issued a token by the STS.
2. The agent sends this STS-issued token to the WCF service.
3. The claims-aware WCF service is configured to trust the STS and the tokens it issues. The claims-aware WCF service uses WIF to validate the token and to parse it. Developers use the appropriate WIF API and types, for example, **ClaimsPrincipal** for the application’s needs, such as implementing authorization for it.

Starting from .NET 4.5, WIF is part of the .NET framework package. Having the WIF classes directly available in the framework itself allows a much deeper integration of claims-based identity in the .NET platform, making it easier to use claims. With WIF 4.5, you do not need to install any out-of-band components in order to start developing claims-aware web applications. WIF classes are now spread across various assemblies, the main ones being System.Security.Claims, System.IdentityModel and System.IdentityModel.Services.

STS is a service that issues tokens upon successful authentication. Microsoft offers two industry standard STS’s:

* [Active Directory Federation Services (AD FS) 2.0](http://go.microsoft.com/fwlink/?LinkID=247516) (http://go.microsoft.com/fwlink/?LinkID=247516)
* [Windows Azure Access Control Service (ACS)](http://go.microsoft.com/fwlink/?LinkID=247517) (http://go.microsoft.com/fwlink/?LinkID=247517).

AD FS 2.0 is part of the Windows Server R2 and can be used as an STS for on-premise scenarios. Azure Active Directory Access Control (also known as Access Control Service or ACS) is a cloud service, offered as part of Microsoft Azure. For testing or educational purposes, you can also use other STS’s in order to build your claims-aware applications. For example, you can use the Local Development STS that is part of the [Identity and Access Tool for Visual Studio](http://go.microsoft.com/fwlink/?LinkID=245849) (http://go.microsoft.com/fwlink/?LinkID=245849) which is freely available online.

To build your first claims-aware WCF service using WIF, see [How To: Build Claims-Aware WCF Service Using WIF](https://msdn.microsoft.com/en-us/library/hh291064(v=vs.110).aspx).

### WIF Features

**.NET Framework 4.6 and 4.5**

* [Identity and Access Tool for Visual Studio 2012](https://msdn.microsoft.com/en-us/library/hh545418(v=vs.110).aspx)
* [WIF Session Management](https://msdn.microsoft.com/en-us/library/hh873347(v=vs.110).aspx)
* [WIF and Web Farms](https://msdn.microsoft.com/en-us/library/hh545457(v=vs.110).aspx)
* [WSFederation Authentication Module Overview](https://msdn.microsoft.com/en-us/library/jj191638(v=vs.110).aspx)
* [WSTrustChannelFactory and WSTrustChannel](https://msdn.microsoft.com/en-us/library/hh873343(v=vs.110).aspx)

#### Custom Token Handlers

**.NET Framework 4.6 and 4.5**

This topic discusses token handlers in WIF and how they are used to process tokens. The topic also covers what is necessary to create custom token handlers for token types that are not supported by default in WIF.

[**Introduction to Token Handlers in WIF**](javascript:void(0))

WIF relies on security token handlers to create, read, write, and validate tokens for a relying party (RP) application or a security token service (STS). Token handlers are extensibility points for you to add a custom token handler in the WIF pipeline, or to customize the way that an existing token handler manages tokens. WIF provides nine built-in security token handlers that can be modified or entirely overridden to change the functionality as necessary.

[**Built-In Security Token Handlers in WIF**](javascript:void(0))

WIF 4.5 includes nine security token handler classes that derive from the abstract base class [SecurityTokenHandler](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.securitytokenhandler(v=vs.110).aspx):

* [EncryptedSecurityTokenHandler](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.encryptedsecuritytokenhandler(v=vs.110).aspx)
* [KerberosSecurityTokenHandler](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.kerberossecuritytokenhandler(v=vs.110).aspx)
* [RsaSecurityTokenHandler](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.rsasecuritytokenhandler(v=vs.110).aspx)
* [SamlSecurityTokenHandler](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.samlsecuritytokenhandler(v=vs.110).aspx)
* [Saml2SecurityTokenHandler](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.saml2securitytokenhandler(v=vs.110).aspx)
* [SessionSecurityTokenHandler](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.sessionsecuritytokenhandler(v=vs.110).aspx)
* [UserNameSecurityTokenHandler](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.usernamesecuritytokenhandler(v=vs.110).aspx)
* [WindowsUserNameSecurityTokenHandler](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.windowsusernamesecuritytokenhandler(v=vs.110).aspx)
* [X509SecurityTokenHandler](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.x509securitytokenhandler(v=vs.110).aspx)

[**Adding a Custom Token Handler**](javascript:void(0))

Some token types, such as Simple Web Tokens (SWT) and JSON Web Tokens (JWT) do not have built-in token handlers provided by WIF. For these token types and for others that do not have a built-in handler, you need to perform the following steps to create a custom token handler.

**Adding a custom token handler**

1. Create a new class that derives from [SecurityTokenHandler](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.securitytokenhandler(v=vs.110).aspx).
2. Override the following methods and provide your own implementation:
   * [CanReadToken](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.securitytokenhandler.canreadtoken(v=vs.110).aspx)
   * [ReadToken](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.securitytokenhandler.readtoken(v=vs.110).aspx)
   * [CanWriteToken](https://msdn.microsoft.com/en-us/library/hh193461(v=vs.110).aspx)
   * [WriteToken](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.securitytokenhandler.writetoken(v=vs.110).aspx)
   * [CanValidateToken](https://msdn.microsoft.com/en-us/library/hh138034(v=vs.110).aspx)
   * [ValidateToken](https://msdn.microsoft.com/en-us/library/hh138061(v=vs.110).aspx)
3. Add a reference to the new custom token handler in the Web.config or App.config file, within the **<system.identityModel>** section that applies to WIF. For example, the following configuration markup specifies a new token handler named**MyCustomTokenHandler** that resides in the **CustomToken** namespace.
4. <system.identityModel>
5. <identityConfiguration saveBootstrapContext="true">
6. <securityTokenHandlers>
7. <add type="CustomToken.MyCustomTokenHandler, CustomToken" />
8. </securityTokenHandlers>
9. </identityConfiguration>
10. </system.identityModel>

Note that if you are providing your own token handler to handle a token type that already has a built-in token handler, you need to add a **<remove>** element to drop the default handler and use your custom handler instead. For example, the following configuration replaces the default [SamlSecurityTokenHandler](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.samlsecuritytokenhandler(v=vs.110).aspx) with the custom token handler:

<system.identityModel>

<identityConfiguration saveBootstrapContext="true">

<securityTokenHandlers>

<remove type=”System.IdentityModel.Tokens.SamlSecurityTokenHandler, System.IdentityModel, Version=4.0.0.0, Culture=neutral, PublicKeyToken=abcdefg123456789”>

<add type="CustomToken.MyCustomTokenHandler, CustomToken" />

</securityTokenHandlers>

</identityConfiguration>

</system.identityModel>

#### Identity and Access Tool for Visual Studio 2012

**.NET Framework 4.6 and 4.5**

This topic describes the new Identity and Access Tool for Visual Studio 11. You can download this tool from the following URL:<http://go.microsoft.com/fwlink/?LinkID=245849> or directly from within Visual Studio 11 by searching for “identity” directly in the Extensions Manager.

The Identity and Access Tool for Visual Studio 11 delivers a dramatically simplified development-time experience with the following highlights:

* Using the new tool, you can develop using web applications project types and target IIS express.
* Unlike with the blanket protection-only authentication, with the new tool, you can specify your local home realm discovery page/controller (or any other endpoint handling the authentication experience within your application) and WIF will configure all unauthenticated requests to go there, instead of redirecting them to the STS.
* The tool includes a test Security Token Service (STS) which runs on your local machine when you launch a debug session. Therefore, you no longer need to create custom STS projects and tweak them in order to get the claims you need to test your applications. The claim types and values are fully customizable.
* You can modify common settings directly via the tool’s UI, without the need to edit web.config.
* You can establish federation with Active Directory Federation Services (AD FS) 2.0 (or other WS-Federation providers) in a single screen.
* The tool leverages Windows Azure Access Control Service (ACS) capabilities with a simple list of checkboxes for all the identity providers that you want to use: Facebook, Google, Live ID, Yahoo!, any OpenID provider, and any WS-Federation provider. Select your identity providers, click OK, then F5, and both your application and ACS will be automatically configured and your test application will be ACS-aware.

#### WIF Session Management

**.NET Framework 4.6 and 4.5**

When a client first tries to access a protected resource that is hosted by a relying party, the client must first authenticate itself to a security token service (STS) that is trusted by the relying party. The STS then issues a security token to the client. The client presents this token to the relying party, which then grants the client access to the protected resource. However, you don’t want the client to have to re-authenticate to the STS for each request, especially because it might not even be on the same computer or in the same domain as the relying party. Instead, Windows Identity Foundation (WIF) has the client and relying party establish a session in which the client uses a session security token to authenticate itself to the relying party for all requests after the first request. The relying party can use this session security token, which is stored inside a cookie, to reconstruct the client’s [System.Security.Claims.ClaimsPrincipal](https://msdn.microsoft.com/en-us/library/system.security.claims.claimsprincipal(v=vs.110).aspx).

The STS defines what authentication the client must provide. However, the client might have multiple credentials with which it can authenticate itself to the STS. For example, it might have a token from Windows Live, a user name and password, a certificate, and a smartkey. In that case, the STS grants the client several identities, with each identity corresponding to one of the credentials that the client presents. The relying party can use one or more of these identities when it decides what level of access to grant the client.

The [System.IdentityModel.Tokens.SessionSecurityToken](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.sessionsecuritytoken(v=vs.110).aspx) is used to reconstruct the client’s [System.Security.Claims.ClaimsPrincipal](https://msdn.microsoft.com/en-us/library/system.security.claims.claimsprincipal(v=vs.110).aspx), which contains all of the client’s identities in [Identities](https://msdn.microsoft.com/en-us/library/hh194537(v=vs.110).aspx). Each [System.Security.Claims.ClaimsIdentity](https://msdn.microsoft.com/en-us/library/system.security.claims.claimsidentity(v=vs.110).aspx) in the collection contains the bootstrap tokens that are associated with that identity.

If a new session token is issued with the session ID of the original session token,[System.IdentityModel.Tokens.SessionSecurityTokenHandler](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.sessionsecuritytokenhandler(v=vs.110).aspx) does not update the session token in the token cache. You should always instantiate a session token with a unique session ID.

|  |
| --- |
| **Note Note** |
| Session.SecurityTokenHandler.ReadToken throws a [XmlException](https://msdn.microsoft.com/en-us/library/system.xml.xmlexception(v=vs.110).aspx) exception if it receives invalid input; for example, if the cookie that contains the session token is corrupted. We recommend that you catch this exception and provide application-specific behavior. |

If a protected Web page contains lots of resources (such as small graphics) that are also in the protected domain, the client must re-authenticate itself to the relying party to download each of those resources. Use of a session authentication token avoids the need to authenticate to the STS for each request, but it still means that many cookies are being sent over. You might want to set up the Web page so that the important data and resources are stored in the protected domain while minor items are stored in an unprotected domain and linked to from the main Web page. Also, set the cookie path to reference only the protected domain.

To operate in reference mode, Microsoft recommends providing a handler for the [SessionSecurityTokenCreated](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.wsfederationauthenticationmodule.sessionsecuritytokencreated(v=vs.110).aspx) event in the **global.asax.cs**file and setting the **IsReferenceMode** property on the token passed in the [SessionToken](https://msdn.microsoft.com/en-us/library/hh158078(v=vs.110).aspx) property. These updates will ensure that the session token operates in reference mode for every request and is favored over merely setting the [IsReferenceMode](https://msdn.microsoft.com/en-us/library/hh737298(v=vs.110).aspx) property on the Session Authentication Module.

[**Extensibility**](javascript:void(0))

You can extend the session management mechanism. One reason for this would be to improve the performance. For example, you could create a custom cookie handler that transforms or optimizes the session security token between its in-memory state and what goes into the cookie. To do so, you can configure the [SessionAuthenticationModule.CookieHandler](https://msdn.microsoft.com/en-us/library/hh158069(v=vs.110).aspx) property of the[System.IdentityModel.Services.SessionAuthenticationModule](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.sessionauthenticationmodule(v=vs.110).aspx) to use a custom cookie handler that derives from[System.IdentityModel.Services.CookieHandler](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.cookiehandler(v=vs.110).aspx). [System.IdentityModel.Services.ChunkedCookieHandler](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.chunkedcookiehandler(v=vs.110).aspx) is the default cookie handler because the cookies exceed the allowable size for Hypertext Transfer Protocol (HTTP); if you use a custom cookie handler instead, you must implement chunking.

For more information, see [ClaimsAwareWebFarm](http://go.microsoft.com/fwlink/?LinkID=248408) (http://go.microsoft.com/fwlink/?LinkID=248408) sample. This sample shows a farm ready session cache (as opposed to a tokenreplycache) so that you can use sessions by reference instead of exchanging big cookies; this sample also demonstrates an easier way of securing cookies in a farm. The session cache is WCF-based. With regard to session securing, the sample demonstrates a new capability in WIF 4.5 of a cookie transform based on MachineKey, which can be activated by simply pasting the appropriate snippet in the web.config. The sample itself is not “farmed”, but it demonstrates what you need for making your app farm-ready.

#### WIF and Web Farms

**.NET Framework 4.6 and 4.5**

When you use Windows Identity Foundation (WIF) to secure the resources of a relying party (RP) application that is deployed in a web farm, you must take specific steps to ensure that WIF can process tokens from instances of the RP application running on different computers in the farm. This processing includes validating session token signatures, encrypting and decrypting session tokens, caching session tokens, and detecting replayed security tokens.

In the typical case, when WIF is used to secure resources of an RP application – whether the RP is running on a single computer or in a web farm -- a session is established with the client based on the security token that was obtained from the security token service (STS). This is to avoid forcing the client to have to authenticate at the STS for every application resource that is secured using WIF. For more information about how WIF handles sessions, see [WIF Session Management](https://msdn.microsoft.com/en-us/library/hh873347(v=vs.110).aspx).

When default settings are used, WIF does the following:

* It uses an instance of the [SessionSecurityTokenHandler](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.sessionsecuritytokenhandler(v=vs.110).aspx) class to read and write a session token (an instance of the[SessionSecurityToken](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.sessionsecuritytoken(v=vs.110).aspx) class) that carries the claims and other information about the security token that was used for authentication as well as information about the session itself. The session token is packaged and stored in a session cookie. By default,[SessionSecurityTokenHandler](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.sessionsecuritytokenhandler(v=vs.110).aspx) uses the [ProtectedDataCookieTransform](https://msdn.microsoft.com/en-us/library/system.identitymodel.protecteddatacookietransform(v=vs.110).aspx) class, which uses the Data Protection API (DPAPI), to protect the session token. The DPAPI provides protection by using the user or machine credentials and stores the key data in the user profile.
* It uses a default, in-memory implementation of the [SessionSecurityTokenCache](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.sessionsecuritytokencache(v=vs.110).aspx) class to store and process the session token.

These default settings work in scenarios in which the RP application is deployed on a single computer; however, when deployed in a web farm, each HTTP request may be sent to and processed by a different instance of the RP application running on a different computer. In this scenario, the default WIF settings described above will not work because both token protection and token caching are dependent on a specific computer.

To deploy an RP application in a web farm, you must ensure that the processing of session tokens (as well as of replayed tokens) is not dependent on the application running on a specific computer. One way to do this is to implement your RP application so that it uses the functionality provided by the ASP.NET **<machineKey>** configuration element and provides distributed caching for processing session tokens and replayed tokens. The **<machineKey>** element allows you to specify the keys needed to validate, encrypt, and decrypt tokens in a configuration file, which enables you to specify the same keys on different computers in the web farm. WIF provides a specialized session token handler, the [MachineKeySessionSecurityTokenHandler](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.tokens.machinekeysessionsecuritytokenhandler(v=vs.110).aspx), that protects tokens by using the keys specified in the **<machineKey>** element. To implement this strategy, you can follow these guidelines:

* Use the ASP.NET **<machineKey>** element in configuration to explicitly specify signing and encryption keys that can be used across computers in the farm. The following XML shows the specification of the **<machineKey>** element under the **<system.web>** element in a configuration file.

Xml

<machineKey compatibilityMode="Framework45" decryptionKey="CC510D … 8925E6" validationKey="BEAC8 … 6A4B1DE" />

* Configure the application to use the [MachineKeySessionSecurityTokenHandler](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.tokens.machinekeysessionsecuritytokenhandler(v=vs.110).aspx) by adding it to the token handler collection. You must first remove the [SessionSecurityTokenHandler](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.sessionsecuritytokenhandler(v=vs.110).aspx) (or any handler derived from the [SessionSecurityTokenHandler](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.sessionsecuritytokenhandler(v=vs.110).aspx) class) from the token handler collection if such a handler is present. The [MachineKeySessionSecurityTokenHandler](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.tokens.machinekeysessionsecuritytokenhandler(v=vs.110).aspx) uses the [MachineKeyTransform](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.machinekeytransform(v=vs.110).aspx) class, which protects the session cookie data by using the cryptographic material specified in the **<machineKey>** element. The following XML shows how to add the [MachineKeySessionSecurityTokenHandler](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.tokens.machinekeysessionsecuritytokenhandler(v=vs.110).aspx) to a token handler collection.

Xml

<securityTokenHandlers>

<remove type="System.IdentityModel.Tokens.SessionSecurityTokenHandler, System.IdentityModel, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />

<add type="System.IdentityModel.Services.Tokens.MachineKeySessionSecurityTokenHandler, System.IdentityModel.Services, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />

</securityTokenHandlers>

* Derive from [SessionSecurityTokenCache](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.sessionsecuritytokencache(v=vs.110).aspx) and implement distributed caching, that is, a cache that is accessible from all computers in the farm on which the RP might run. Configure the RP to use your distributed cache by specifying the [<sessionSecurityTokenCache>](https://msdn.microsoft.com/en-us/library/hh568669(v=vs.110).aspx)element in the configuration file. You can override the [SessionSecurityTokenCache.LoadCustomConfiguration](https://msdn.microsoft.com/en-us/library/hh137718(v=vs.110).aspx) method in your derived class to implement child elements of the **<sessionSecurityTokenCache>** element if they are required.

Xml

<caches>

<sessionSecurityTokenCache type="MyCacheLibrary.MySharedSessionSecurityTokenCache, MyCacheLibrary">

<!—optional child configuration elements, if implemented by the derived class -->

</sessionSecurityTokenCache>

</caches>

One way to implement distributed caching is to provide a WCF front end for your custom cache. For more information about implementing a WCF caching service, see [The WCF Caching Service](https://msdn.microsoft.com/en-us/library/hh545457(v=vs.110).aspx#BKMK_TheWCFCachingService). For more information about implementing a WCF client that the RP application can use to call the caching service, see [The WCF Caching Client](https://msdn.microsoft.com/en-us/library/hh545457(v=vs.110).aspx#BKMK_TheWCFClient).

* If your application detects replayed tokens you must follow a similar distributed caching strategy for the token replay cache by deriving from [TokenReplayCache](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.tokenreplaycache(v=vs.110).aspx) and pointing to your token replay caching service in the [<tokenReplayCache>](https://msdn.microsoft.com/en-us/library/hh568635(v=vs.110).aspx) configuration element.

|  |
| --- |
| **Important note Important** |
| All of the example XML and code in this topic is taken from the [ClaimsAwareWebFarm](http://go.microsoft.com/fwlink/?LinkID=248408) (http://go.microsoft.com/fwlink/?LinkID=248408) sample. |
| **Security note Security Note** |
| The examples in this topic are provided as-is and are not intended to be used in production code without modification. |

[**The WCF Caching Service**](javascript:void(0))

The following interface defines the contract between the WCF caching service and the WCF client used by the relying party application to communicate with it. It essentially exposes the methods of the [SessionSecurityTokenCache](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.sessionsecuritytokencache(v=vs.110).aspx) class as service operations.

[ServiceContract()]

public interface ISessionSecurityTokenCacheService

{

[OperationContract]

void AddOrUpdate(string endpointId, string contextId, string keyGeneration, SessionSecurityToken value, DateTime expiryTime);

[OperationContract]

IEnumerable<SessionSecurityToken> GetAll(string endpointId, string contextId);

[OperationContract]

SessionSecurityToken Get(string endpointId, string contextId, string keyGeneration);

[OperationContract(Name = "RemoveAll")]

void RemoveAll(string endpointId, string contextId);

[OperationContract(Name = "RemoveAllByEndpointId")]

void RemoveAll(string endpointId);

[OperationContract]

void Remove(string endpointId, string contextId, string keyGeneration);

}

The following code shows the implementation of the WCF caching service. In this example, the default, in-memory session token cache implemented by WIF is used. Alternatively, you could implement a durable cache backed by a database.**ISessionSecurityTokenCacheService** defines the interface shown above. In this example, not all of the methods required to implement the interface are shown for brevity.

using System;

using System.Collections.Generic;

using System.IdentityModel.Configuration;

using System.IdentityModel.Tokens;

using System.ServiceModel;

using System.Xml;

namespace WcfSessionSecurityTokenCacheService

{

[ServiceBehavior(InstanceContextMode = InstanceContextMode.Single)]

public class SessionSecurityTokenCacheService : ISessionSecurityTokenCacheService

{

SessionSecurityTokenCache internalCache;

// sets the internal cache used by the service to the default WIF in-memory cache.

public SessionSecurityTokenCacheService()

{

internalCache = new IdentityModelCaches().SessionSecurityTokenCache;

}

...

public SessionSecurityToken Get(string endpointId, string contextId, string keyGeneration)

{

// Delegates to the default, in-memory MruSessionSecurityTokenCache used by WIF

SessionSecurityToken token = internalCache.Get(new SessionSecurityTokenCacheKey(endpointId, GetContextId(contextId), GetKeyGeneration(keyGeneration)));

return token;

}

...

private static UniqueId GetContextId(string contextIdString)

{

return contextIdString == null ? null : new UniqueId(contextIdString);

}

private static UniqueId GetKeyGeneration(string keyGenerationString)

{

return keyGenerationString == null ? null : new UniqueId(keyGenerationString);

}

}

}

[**The WCF Caching Client**](javascript:void(0))

This section shows the implementation of a class that derives from [SessionSecurityTokenCache](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.sessionsecuritytokencache(v=vs.110).aspx) and that delegates calls to the caching service. You configure the RP application to use this class through the [<sessionSecurityTokenCache>](https://msdn.microsoft.com/en-us/library/hh568669(v=vs.110).aspx) element as in the following XML

<caches>

<sessionSecurityTokenCache type="CacheLibrary.SharedSessionSecurityTokenCache, CacheLibrary">

<!--cacheServiceAddress points to the centralized session security token cache service running in the web farm.-->

<cacheServiceAddress url="http://localhost:4161/SessionSecurityTokenCacheService.svc" />

</sessionSecurityTokenCache>

</caches>

The class overrides the [LoadCustomConfiguration](https://msdn.microsoft.com/en-us/library/hh137718(v=vs.110).aspx) method to get the service endpoint from the custom **<cacheServiceAddress>** child element of the **<sessionSecurityTokenCache>** element. It uses this endpoint to initialize an **ISessionSecurityTokenCacheService** channel over which it can communicate with the service. In this example, not all of the methods required to implement the[SessionSecurityTokenCache](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.sessionsecuritytokencache(v=vs.110).aspx) class are shown for brevity.

using System;

using System.Configuration;

using System.IdentityModel.Configuration;

using System.IdentityModel.Tokens;

using System.ServiceModel;

using System.Xml;

namespace CacheLibrary

{

/// <summary>

/// This class acts as a proxy to the WcfSessionSecurityTokenCacheService.

/// </summary>

public class SharedSessionSecurityTokenCache : SessionSecurityTokenCache, ICustomIdentityConfiguration

{

private ISessionSecurityTokenCacheService WcfSessionSecurityTokenCacheServiceClient;

internal SharedSessionSecurityTokenCache()

{

}

/// <summary>

/// Creates a client channel to call the service host.

/// </summary>

protected void Initialize(string cacheServiceAddress)

{

if (this.WcfSessionSecurityTokenCacheServiceClient != null)

{

return;

}

ChannelFactory<ISessionSecurityTokenCacheService> cf = new ChannelFactory<ISessionSecurityTokenCacheService>(

new WS2007HttpBinding(SecurityMode.None),

new EndpointAddress(cacheServiceAddress));

this.WcfSessionSecurityTokenCacheServiceClient = cf.CreateChannel();

}

#region SessionSecurityTokenCache Members

// Delegates the following operations to the centralized session security token cache service in the web farm.

...

public override SessionSecurityToken Get(SessionSecurityTokenCacheKey key)

{

return this.WcfSessionSecurityTokenCacheServiceClient.Get(key.EndpointId, GetContextIdString(key), GetKeyGenerationString(key));

}

...

#endregion

#region ICustomIdentityConfiguration Members

// Called from configuration infrastructure to load custom elements

public void LoadCustomConfiguration(XmlNodeList nodeList)

{

// Retrieve the endpoint address of the centralized session security token cache service running in the web farm

if (nodeList.Count == 0)

{

throw new ConfigurationException("No child config element found under <sessionSecurityTokenCache>.");

}

XmlElement cacheServiceAddressElement = nodeList.Item(0) as XmlElement;

if (cacheServiceAddressElement.LocalName != "cacheServiceAddress")

{

throw new ConfigurationException("First child config element under <sessionSecurityTokenCache> is expected to be <cacheServiceAddress>.");

}

string cacheServiceAddress = null;

if (cacheServiceAddressElement.Attributes["url"] != null)

{

cacheServiceAddress = cacheServiceAddressElement.Attributes["url"].Value;

}

else

{

throw new ConfigurationException("<cacheServiceAddress> is expected to contain a 'url' attribute.");

}

// Initialize the proxy to the WebFarmSessionSecurityTokenCacheService

this.Initialize(cacheServiceAddress);

}

#endregion

private static string GetKeyGenerationString(SessionSecurityTokenCacheKey key)

{

return key.KeyGeneration == null ? null : key.KeyGeneration.ToString();

}

private static string GetContextIdString(SessionSecurityTokenCacheKey key)

{

return GetContextIdString(key.ContextId);

}

private static string GetContextIdString(UniqueId contextId)

{

return contextId == null ? null : contextId.ToString();

}

}

}

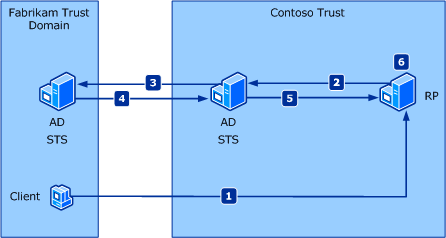
#### WSFederation Authentication Module Overview

**.NET Framework 4.6 and 4.5**

Windows Identity Foundation (WIF) includes support for federated authentication in ASP.NET applications through the WS-Federated Authentication Module (WS-FAM). This topic will help you understand how federated authentication works and how to use it.

[**Overview of Federated Authentication**](javascript:void(0))

Federated authentication allows a Security Token Service (STS) in one trust domain to provide authentication information to an STS in another trust domain when there is a trust relationship between the two domains. An example of this is shown in the following illustration.



1. A client in the Fabrikam trust domain sends a request to a Relying Party (RP) application in the Contoso trust domain.
2. The RP redirects the client to an STS in the Contoso trust domain. This STS has no knowledge of the client.
3. The Contoso STS redirects the client to an STS in the Fabrikam trust domain, with which the Contoso trust domain has a trust relationship.
4. The Fabrikam STS verifies the client’s identity and issues a security token to the Contoso STS.
5. The Contoso STS uses the Fabrikam token to create its own token that can be used by the RP and sends it to the RP.
6. The RP extracts the client’s claims from the security token and makes an authorization decision.

[**Using the Federated Authentication Module with ASP.NET**](javascript:void(0))

[WSFederationAuthenticationModule](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.wsfederationauthenticationmodule(v=vs.110).aspx) (WS-FAM) is an HTTP module that lets you add federated authentication to an ASP.NET application. Federated authentication lets authentication logic be handled by the STS and lets you focus on writing business logic.

You configure the WS-FAM to specify the STS to which non-authenticated requests should be redirected. WIF lets you authenticate a user in two ways:

1. Passive redirect: When an unauthenticated user tries to access a protected resource, and you want to simply redirect them to an STS without requiring a login page, then this is the right approach. The STS verifies the user’s identity, and issues a security token that contains the appropriate claims for that user. This option requires the WS-FAM to be added in the HTTP Modules pipeline. You can use the Identity and Access Tool for Visual Studio 2012 to modify your application’s configuration file to use the WS-FAM and to federate with an STS. For more information, see [Identity and Access Tool for Visual Studio 2012](https://msdn.microsoft.com/en-us/library/hh545418(v=vs.110).aspx).
2. You can call the [WSFederationAuthenticationModule.SignIn](https://msdn.microsoft.com/en-us/library/hh562218(v=vs.110).aspx) method or the [RedirectToIdentityProvider](https://msdn.microsoft.com/en-us/library/hh562191(v=vs.110).aspx) method from the code-behind for a sign-in page in your RP application.

In passive redirect, all communication is performed through response/redirect from the client (typically a browser). You can add the WS-FAM to your application’s HTTP pipeline, where it watches for unauthenticated user requests and redirects users to the STS you specify.

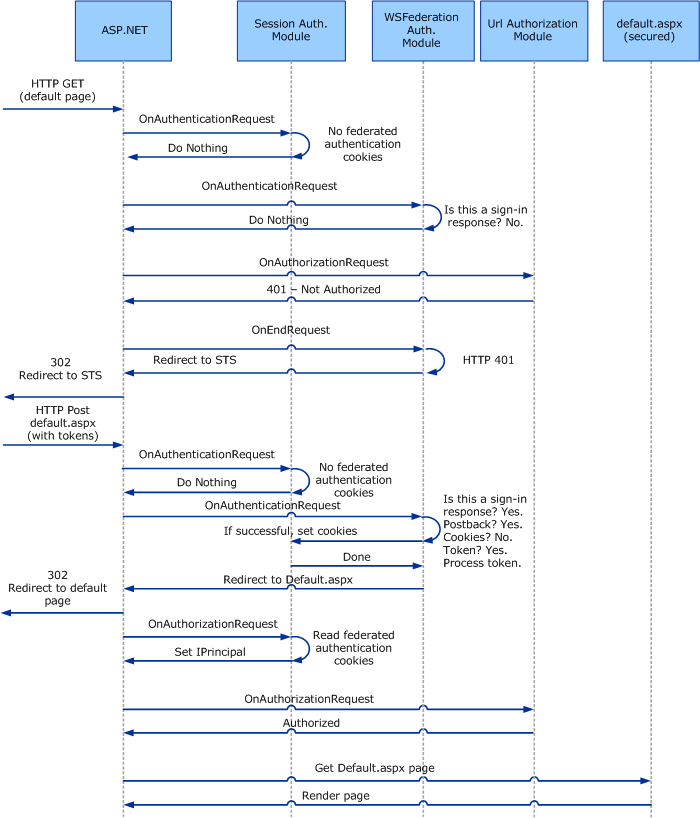
The WS-FAM also raises several events that let you customize its functionality in an ASP.NET application.

[**How the WS-FAM Works**](javascript:void(0))

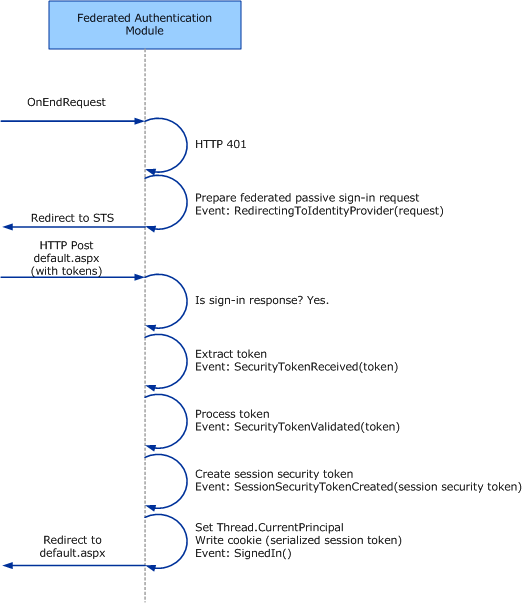
The WS-FAM is implemented in the [WSFederationAuthenticationModule](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.wsfederationauthenticationmodule(v=vs.110).aspx) class. Typically, you add the WS-FAM to the HTTP pipeline of your ASP.NET RP application. When an unauthenticated user tries to access a protected resource, the RP returns a “401 authorization denied” HTTP response. The WS-FAM intercepts this response instead of allowing the client to receive it, then it redirects the user to the specified STS. The STS issues a security token, which the WS-FAM again intercepts. The WS-FAM uses the token to create an instance of[ClaimsPrincipal](https://msdn.microsoft.com/en-us/library/system.security.claims.claimsprincipal(v=vs.110).aspx) for the authenticated user, which enables regular .NET Framework authorization mechanisms to function.

Because HTTP is stateless, we need a way to avoid repeating this whole process every time that the user tries to access another protected resource. This is where the [SessionAuthenticationModule](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.sessionauthenticationmodule(v=vs.110).aspx) comes in. When the STS issues a security token for the user,[SessionAuthenticationModule](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.sessionauthenticationmodule(v=vs.110).aspx) also creates a session security token for the user and puts it in a cookie. On subsequent requests, the[SessionAuthenticationModule](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.sessionauthenticationmodule(v=vs.110).aspx) intercepts this cookie and uses it to reconstruct the user’s [ClaimsPrincipal](https://msdn.microsoft.com/en-us/library/system.security.claims.claimsprincipal(v=vs.110).aspx).

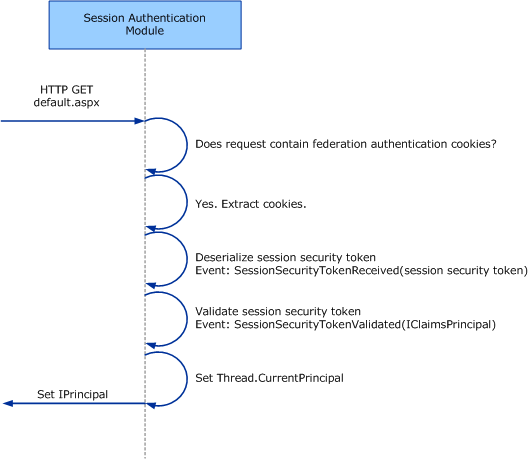
The following diagram shows the overall flow of information in the passive redirect case. The request is automatically redirected via the STS to establish credentials without a login page:



The following diagram shows more detail on what happens when the user has authenticated to the STS and their security tokens are processed by the [WSFederationAuthenticationModule](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.wsfederationauthenticationmodule(v=vs.110).aspx):



The following diagram shows more detail on what happens when the user’s security tokens have been serialized into cookies and are intercepted by the [SessionAuthenticationModule](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.sessionauthenticationmodule(v=vs.110).aspx):



[**Events**](javascript:void(0))

[WSFederationAuthenticationModule](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.wsfederationauthenticationmodule(v=vs.110).aspx) , [SessionAuthenticationModule](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.sessionauthenticationmodule(v=vs.110).aspx), and their parent class, [HttpModuleBase](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.httpmodulebase(v=vs.110).aspx), raise events at various stages of processing of an HTTP request. You can handle these events in the global.asax file of your ASP.NET application.

* The ASP.NET infrastructure invokes the module’s [Init](https://msdn.microsoft.com/en-us/library/hh562217(v=vs.110).aspx) method to initialize the module.
* The [FederatedAuthentication.FederationConfigurationCreated](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.federatedauthentication.federationconfigurationcreated(v=vs.110).aspx) event is raised when the ASP.NET infrastructure invokes the [Init](https://msdn.microsoft.com/en-us/library/hh562217(v=vs.110).aspx)method for the first time on one of the application’s modules that derive from [HttpModuleBase](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.httpmodulebase(v=vs.110).aspx). This method accesses the static[FederatedAuthentication.FederationConfiguration](https://msdn.microsoft.com/en-us/library/hh737321(v=vs.110).aspx) property, which causes configuration to be loaded from the Web.config file. This event is only raised the first time this property is accessed. The [FederationConfiguration](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.configuration.federationconfiguration(v=vs.110).aspx) object that is initialized from configuration can be accessed through the [FederationConfigurationCreatedEventArgs.FederationConfiguration](https://msdn.microsoft.com/en-us/library/hh562321(v=vs.110).aspx) property in an event handler. You can use this event to modify the configuration before it is applied to any modules. You can add a handler for this event in the Application\_Start method:
* void Application\_Start(object sender, EventArgs e)
* {
* FederatedAuthentication.FederationConfigurationCreated += new EventHandler<FederationConfigurationCreatedEventArgs>(FederatedAuthentication\_FederationConfigurationCreated);
* }

Each module overrides the [HttpModuleBase.InitializeModule](https://msdn.microsoft.com/en-us/library/hh562214(v=vs.110).aspx) and [HttpModuleBase.InitializePropertiesFromConfiguration](https://msdn.microsoft.com/en-us/library/hh737313(v=vs.110).aspx) abstract methods. The first of these methods adds handlers for ASP.NET pipeline events that are of interest to the module. In most cases the module’s default implementation will suffice. The second of these methods initializes the module’s properties from its[HttpModuleBase.FederationConfiguration](https://msdn.microsoft.com/en-us/library/hh562230(v=vs.110).aspx) property. (This is a copy of the configuration that was loaded previously.) You may need to override this second method if you want to support the initialization of new properties from configuration in classes that you derive from [WSFederationAuthenticationModule](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.wsfederationauthenticationmodule(v=vs.110).aspx) or [SessionAuthenticationModule](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.sessionauthenticationmodule(v=vs.110).aspx). In such cases you would also need to derive from the appropriate configuration objects to support the added configuration properties; for example, from [IdentityConfiguration](https://msdn.microsoft.com/en-us/library/system.identitymodel.configuration.identityconfiguration(v=vs.110).aspx),[WsFederationConfiguration](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.configuration.wsfederationconfiguration(v=vs.110).aspx), or [FederationConfiguration](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.configuration.federationconfiguration(v=vs.110).aspx).

* The WS-FAM raises the [SecurityTokenReceived](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.wsfederationauthenticationmodule.securitytokenreceived(v=vs.110).aspx) event when it intercepts a security token that has been issued by the STS.
* The WS-FAM raises the [SecurityTokenValidated](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.wsfederationauthenticationmodule.securitytokenvalidated(v=vs.110).aspx) event after it has validated the token.
* The [SessionAuthenticationModule](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.sessionauthenticationmodule(v=vs.110).aspx) raises the [SessionSecurityTokenCreated](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.sessionauthenticationmodule.sessionsecuritytokencreated(v=vs.110).aspx) event when it creates a session security token for the user.
* The [SessionAuthenticationModule](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.sessionauthenticationmodule(v=vs.110).aspx) raises the [SessionSecurityTokenReceived](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.sessionauthenticationmodule.sessionsecuritytokenreceived(v=vs.110).aspx) event when it intercepts subsequent requests with the cookie that contains the session security token.
* Before the WS-FAM redirects the user to the issuer, it raises the [RedirectingToIdentityProvider](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.wsfederationauthenticationmodule.redirectingtoidentityprovider(v=vs.110).aspx) event. The WS-Federation sign-in request is available through the [SignInRequestMessage](https://msdn.microsoft.com/en-us/library/hh137598(v=vs.110).aspx) property of the [RedirectingToIdentityProviderEventArgs](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.redirectingtoidentityprovidereventargs(v=vs.110).aspx) passed in the event. You may choose to modify the request before sending this out to the issuer.
* The WS-FAM raises the [SignedIn](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.wsfederationauthenticationmodule.signedin(v=vs.110).aspx) event when the cookie is successfully written and the user is signed in.
* The WS-FAM raises the [SigningOut](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.wsfederationauthenticationmodule.signingout(v=vs.110).aspx) event one time per session as the session is being closed down for each user. It is not raised if the session is closed down on the client-side (for example, by deleting the session cookie). In an SSO environment, the IP-STS can request each RP to sign out, too. This will also raise this event, with [IsIPInitiated](https://msdn.microsoft.com/en-us/library/hh193211(v=vs.110).aspx) set to **true**.

|  |
| --- |
| **Note Note** |
| You should not use the [Thread.CurrentPrincipal](https://msdn.microsoft.com/en-us/library/system.threading.thread.currentprincipal(v=vs.110).aspx) property during any event raised by [WSFederationAuthenticationModule](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.wsfederationauthenticationmodule(v=vs.110).aspx) or[SessionAuthenticationModule](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.sessionauthenticationmodule(v=vs.110).aspx). This is because [Thread.CurrentPrincipal](https://msdn.microsoft.com/en-us/library/system.threading.thread.currentprincipal(v=vs.110).aspx) is set after the authentication process, while events are raised during the authentication process. |

[**Configuration of Federated Authentication**](javascript:void(0))

The WS-FAM and SAM are configured through the [<federationConfiguration>](https://msdn.microsoft.com/en-us/library/hh568657(v=vs.110).aspx) element. The [<wsFederation>](https://msdn.microsoft.com/en-us/library/hh568665(v=vs.110).aspx) child element configures default values for the WS-FAM properties; such as the [Issuer](https://msdn.microsoft.com/en-us/library/hh551705(v=vs.110).aspx) property and the [Realm](https://msdn.microsoft.com/en-us/library/hh551696(v=vs.110).aspx) property. (These values can be changed on a per request basis by providing handlers for some of the WS-FAM events; for example, [RedirectingToIdentityProvider](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.wsfederationauthenticationmodule.redirectingtoidentityprovider(v=vs.110).aspx).) The cookie handler that is used by the SAM is configured through the [<cookieHandler>](https://msdn.microsoft.com/en-us/library/hh568663(v=vs.110).aspx) child element. WIF provides a default cookie handler implemented in the[ChunkedCookieHandler](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.chunkedcookiehandler(v=vs.110).aspx) class that can have its chunk size set through the [<chunkedCookieHandler>](https://msdn.microsoft.com/en-us/library/hh568653(v=vs.110).aspx) element. The**<federationConfiguration>** element references an [IdentityConfiguration](https://msdn.microsoft.com/en-us/library/system.identitymodel.configuration.identityconfiguration(v=vs.110).aspx), which provides configuration for other WIF components used in the application, such as the [ClaimsAuthenticationManager](https://msdn.microsoft.com/en-us/library/system.security.claims.claimsauthenticationmanager(v=vs.110).aspx) and the [ClaimsAuthorizationManager](https://msdn.microsoft.com/en-us/library/system.security.claims.claimsauthorizationmanager(v=vs.110).aspx). The identity configuration may be referenced explicitly by specifying a named [<identityConfiguration>](https://msdn.microsoft.com/en-us/library/hh568637(v=vs.110).aspx) element in the **identityConfigurationName** attribute of the**<federationConfiguration>** element. If the identity configuration is not referenced explicitly, the default identity configuration will be used.

The following XML shows a configuration of an ASP.NET relying party (RP) application. The [SystemIdentityModelSection](https://msdn.microsoft.com/en-us/library/system.identitymodel.configuration.systemidentitymodelsection(v=vs.110).aspx) and[SystemIdentityModelServicesSection](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.configuration.systemidentitymodelservicessection(v=vs.110).aspx) configuration sections are added under the **<configSections>** element. The SAM and WS-FAM are added to the HTTP Modules under the **<system.webServer>**/**<modules>** element. Finally the WIF components are configured under the**<system.identityModel>**/**<identityConfiguration>** and **<system.identityModel.services>**/**<federationConfiguration>** elements. This configuration specifies the chunked cookie handler as it is the default cookie handler and there is not a cookie handler type specified in the**<cookieHandler>** element.

|  |
| --- |
| **Caution note Caution** |
| In the following example, both the **requireHttps** attribute of the **<wsFederation>** element and the **requireSsl** attribute of the**<cookieHandler>** element are **false**. This presents a potential security threat. In production, both these values should be set **true**. |

<configuration>

<configSections>

<section name="system.identityModel" type="System.IdentityModel.Configuration.SystemIdentityModelSection, System.IdentityModel, Version=4.0.0.0, Culture=neutral, PublicKeyToken=B77A5C561934E089" />

<section name="system.identityModel.services" type="System.IdentityModel.Services.Configuration.SystemIdentityModelServicesSection, System.IdentityModel.Services, Version=4.0.0.0, Culture=neutral, PublicKeyToken=B77A5C561934E089" />

</configSections>

...

<system.webServer>

<modules>

<add name="WSFederationAuthenticationModule" type="System.IdentityModel.Services.WSFederationAuthenticationModule, System.IdentityModel.Services, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" preCondition="managedHandler" />

<add name="SessionAuthenticationModule" type="System.IdentityModel.Services.SessionAuthenticationModule, System.IdentityModel.Services, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" preCondition="managedHandler" />

</modules>

</system.webServer>

<system.identityModel>

<identityConfiguration>

<audienceUris>

<add value="http://localhost:50969/" />

</audienceUris>

<certificateValidation certificateValidationMode="None" />

<issuerNameRegistry type="System.IdentityModel.Tokens.ConfigurationBasedIssuerNameRegistry, System.IdentityModel, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089">

<trustedIssuers>

<add thumbprint="9B74CB2F320F7AAFC156E1252270B1DC01EF40D0" name="LocalSTS" />

</trustedIssuers>

</issuerNameRegistry>

</identityConfiguration>

</system.identityModel>

<system.identityModel.services>

<federationConfiguration>

<wsFederation passiveRedirectEnabled="true" issuer="http://localhost:15839/wsFederationSTS/Issue" realm="http://localhost:50969/" reply="http://localhost:50969/" requireHttps="false" />

<cookieHandler requireSsl="false" />

</federationConfiguration>

</system.identityModel.services>

</configuration>

#### WSTrustChannelFactory and WSTrustChannel

**.NET Framework 4.6 and 4.5**

If you are already familiar with Windows Communication Foundation (WCF), you know that a WCF client is already federation aware. By configuring a WCF client with a [WSFederationHttpBinding](https://msdn.microsoft.com/en-us/library/system.servicemodel.wsfederationhttpbinding(v=vs.110).aspx) or similar custom binding, you can enable federated authentication to a service.

WCF obtains the token that is issued by the security token service (STS) behind the scenes and uses this token to authenticate to the service. The main limitation to this approach is that there is no visibility into the client’s communications with the server. WCF automatically generates the request security token (RST) to the STS based on the issued token parameters on the binding. This means that the client cannot vary the RST parameters per request, inspect the request security token response (RSTR) to get information such as display claims, or cache the token for future use.

Currently, the WCF client is suitable for basic federation scenarios. However, one of the major scenarios that Windows Identity Foundation (WIF) supports requires control over the RST at a level that WCF does not easily allow. Therefore, WIF adds features that give you more control over communication with the STS.

WIF supports the following federation scenarios:

* Using a WCF client without any WIF dependencies to authenticate to a federated service
* Enabling WIF on a WCF client to insert an ActAs or OnBehalfOf element into the RST to the STS
* Using WIF alone to obtain a token from the STS and then enable a WCF client to authenticate with this token. For more information, see [ClaimsAwareWebService](http://go.microsoft.com/fwlink/?LinkID=248406) sample.

The first scenario is self-explanatory: Existing WCF clients will continue to work with WIF relying parties and STSs. This topic discusses the remaining two scenarios.

[**Enhancing an Existing WCF Client with ActAs / OnBehalfOf**](javascript:void(0))

In a typical identity delegation scenario, a client calls a middle-tier service, which then calls a back-end service. The middle-tier service acts as, or acts on behalf of, the client.

|  |
| --- |
| **Tip Tip** |
| What is the difference between ActAs and OnBehalfOf?  From the WS-Trust procotol standpoint:   1. An ActAs RST element indicates that the requestor wants a token that contains claims about two distinct entities: the requestor, and an external entity represented by the token in the ActAs element. 2. An OnBehalfOf RST element indicates that the requestor wants a token that contains claims only about one entity: the external entity represented by the token in the OnBehalfOf element.   The ActAs feature is typically used in scenarios that require composite delegation, where the final recipient of the issued token can inspect the entire delegation chain and see not just the client, but all intermediaries. This lets it perform access control, auditing and other related activities based on the entire identity delegation chain. The ActAs feature is commonly used in multi-tiered systems to authenticate and pass information about identities between the tiers without having to pass this information at the application/business logic layer.  The OnBehalfOf feature is used in scenarios where only the identity of the original client is important and is effectively the same as the identity impersonation feature available in Windows. When OnBehalfOf is used, the final recipient of the issued token can only see claims about the original client, and the information about intermediaries is not preserved. One common pattern where the OnBehalfOf feature is used is the proxy pattern where the client cannot access the STS directly but instead communicates through a proxy gateway. The proxy gateway authenticates the caller and puts information about the caller into the OnBehalfOf element of the RST message that it then sends to the real STS for processing. The resulting token contains only claims related to the client of the proxy, making the proxy completely transparent to the receiver of the issued token.Note that WIF does not support <wsse:SecurityTokenReference> or <wsa:EndpointReferences> as a child of <wst:OnBehalfOf>. The WS-Trust specification allows for three ways to identify the original requestor (on behalf of whom the proxy is acting). These are:   * Security token reference. A reference to a token, either in the message, or possibly retrieved out of band). * Endpoint reference. Used as a key to look up data, again out of band. * Security token. Identifies the original requestor directly.   WIF supports only security tokens, either encrypted or unencrypted, as a direct child element of <wst:OnBehalfOf>. |

This information is conveyed to a WS-Trust issuer using the ActAs and OnBehalfOf token elements in the RST.

WCF exposes an extensibility point on the binding that allows arbitrary XML elements to be added to the RST. However, because the extensibility point is tied to the binding, scenarios that require the RST contents to vary per call must re-create the client for every call, which decreases performance. WIF uses extension methods on the ChannelFactory class to allow developers to attach any token that is obtained out of band to the RST. The following code example shows how to take a token that represents the client (such as an X.509, username, or Security Assertion Markup Language (SAML) token) and attach it to the RST that is sent to the issuer.

IHelloService serviceChannel = channelFactory.CreateChannelActingAs<IHelloService>( clientSamlToken );

serviceChannel.Hello(“Hi!”);

WIF provides the following benefits:

* The RST can be modified per channel; therefore, middle-tier services do not have to re-create the channel factory for each client, which improves performance.
* This works with existing WCF clients, which makes an easy upgrade path possible for existing WCF middle-tier services that want to enable identity delegation semantics.

However, there is still no visibility into the client’s communication with the STS. We’ll examine this in the third scenario.

[**Communicating Directly with an Issuer and Using the Issued Token to Authenticate**](javascript:void(0))

For some advanced scenarios, enhancing a WCF client is not enough. Developers who use only WCF typically use Message In / Message Out contracts and handle client-side parsing of the issuer response manually.

WIF introduces the [WSTrustChannelFactory](https://msdn.microsoft.com/en-us/library/system.servicemodel.security.wstrustchannelfactory(v=vs.110).aspx) and [WSTrustChannel](https://msdn.microsoft.com/en-us/library/system.servicemodel.security.wstrustchannel(v=vs.110).aspx) classes to let the client communicate directly with a WS-Trust issuer. The[WSTrustChannelFactory](https://msdn.microsoft.com/en-us/library/system.servicemodel.security.wstrustchannelfactory(v=vs.110).aspx) and [WSTrustChannel](https://msdn.microsoft.com/en-us/library/system.servicemodel.security.wstrustchannel(v=vs.110).aspx) classes enable strongly typed RST and RSTR objects to flow between the client and issuer, as shown in the following code example.

WSTrustChannelFactory trustChannelFactory = new WSTrustChannelFactory( stsBinding, stsAddress );

WSTrustChannel channel = (WSTrustChannel) trustChannelFactory.CreateChannel();

RequestSecurityToken rst = new RequestSecurityToken(RequestTypes.Issue);

rst.AppliesTo = new EndpointAddress(serviceAddress);

RequestSecurityTokenResponse rstr = null;

SecurityToken token = channel.Issue(rst, out rstr);

Note that the *out* parameter on the [Issue](https://msdn.microsoft.com/en-us/library/system.servicemodel.security.wstrustchannel.issue(v=vs.110).aspx) method allows access to the RSTR for client-side inspection.

So far, we’ve only seen how to obtain a token. The token that is returned from the [WSTrustChannel](https://msdn.microsoft.com/en-us/library/system.servicemodel.security.wstrustchannel(v=vs.110).aspx) object is a GenericXmlSecurityTokenthat contains all of the information that is necessary for authentication to a relying party. The following code example shows how to use this token.

IHelloService serviceChannel = channelFactory.CreateChannelWithIssuedToken<IHelloService>( token ); serviceChannel.Hello(“Hi!”);

The [CreateChannelWithIssuedToken](https://msdn.microsoft.com/en-us/library/hh194605(v=vs.110).aspx) extension method on the ChannelFactory object indicates to WIF that you have obtained a token out of band, and that it should stop the normal WCF call to the issuer and instead use the token that you obtained to authenticate to the relying party. This has the following benefits:

* It gives you complete control over the token issuance process.
* It supports ActAs / OnBehalfOf scenarios by directly setting these properties on the outgoing RST.
* It enables dynamic client-side trust decisions to be made based on the contents of the RSTR.
* It lets you cache and reuse the token that is returned from the [Issue](https://msdn.microsoft.com/en-us/library/system.servicemodel.security.wstrustchannel.issue(v=vs.110).aspx) method.
* [WSTrustChannelFactory](https://msdn.microsoft.com/en-us/library/system.servicemodel.security.wstrustchannelfactory(v=vs.110).aspx) and [WSTrustChannel](https://msdn.microsoft.com/en-us/library/system.servicemodel.security.wstrustchannel(v=vs.110).aspx) allow for control of channel caching, fault, and recovery semantics according to WCF best practices.

## WIF How-To's Index

**.NET Framework 4.6 and 4.5**

* [How To: Build Claims-Aware ASP.NET MVC Web Application Using WIF](https://msdn.microsoft.com/en-us/library/hh291061(v=vs.110).aspx)
* [How To: Build Claims-Aware ASP.NET Web Forms Application Using WIF](https://msdn.microsoft.com/en-us/library/hh987037(v=vs.110).aspx)
* [How To: Build Claims-Aware ASP.NET Application Using Forms-Based Authentication](https://msdn.microsoft.com/en-us/library/hh291068(v=vs.110).aspx)
* [How To: Build Claims-Aware ASP.NET Application Using Windows Authentication](https://msdn.microsoft.com/en-us/library/hh987035(v=vs.110).aspx)
* [How To: Debug Claims-Aware Applications And Services Using WIF Tracing](https://msdn.microsoft.com/en-us/library/hh291063(v=vs.110).aspx)
* [How To: Display Signed In Status Using WIF](https://msdn.microsoft.com/en-us/library/hh874923(v=vs.110).aspx)
* [How To: Enable Token Replay Detection](https://msdn.microsoft.com/en-us/library/jj161103(v=vs.110).aspx)
* [How To: Enable WIF Tracing](https://msdn.microsoft.com/en-us/library/jj161102(v=vs.110).aspx)
* [How To: Enable WIF for a WCF Web Service Application](https://msdn.microsoft.com/en-us/library/jj161104(v=vs.110).aspx)
* [How To: Transform Incoming Claims](https://msdn.microsoft.com/en-us/library/hh987036(v=vs.110).aspx)

### How To: Build Claims-Aware ASP.NET MVC Web Application Using WIF

**.NET Framework 4.6 and 4.5**

[**Applies To**](javascript:void(0))

* Microsoft® Windows® Identity Foundation (WIF)
* ASP.NET® MVC

[**Summary**](javascript:void(0))

This How-To provides detailed step-by-step procedures for creating simple claims-aware ASP.NET MVC web application. It also provides instructions how to test the simple claims-aware ASP.NET MVC web application for successful implementation of claims-based authentication. This How-To does not have detailed instructions for creating a Security Token Service (STS), and assumes you have already configured an STS.

[**Contents**](javascript:void(0))

* Objectives
* Summary of Steps
* Step 1 – Create Simple ASP.NET MVC Application
* Step 2 – Configure ASP.NET MVC Application for Claims-Based Authentication
* Step 3 – Test Your Solution
* Related Items

[**Objectives**](javascript:void(0))

* Configure ASP.NET MVC web application for claims-based authentication
* Test successful claims-aware ASP.NET MVC web application

[**Summary of Steps**](javascript:void(0))

* Step 1 – Create Simple ASP.NET MVC Application
* Step 2 – Configure ASP.NET MVC Application for Claims-Based Authentication
* Step 3 – Test Your Solution

[**Step 1 – Create Simple ASP.NET MVC Application**](javascript:void(0))

In this step, you will create a new ASP.NET MVC application.

To create simple ASP.NET MVC application

1. Start Visual Studio and click **File**, **New**, and then **Project**.
2. In the **New Project** window, click **ASP.NET MVC 3 Web Application**.
3. In **Name**, enter TestApp and press **OK**.
4. In the **New ASP.NET MVC 3 Project** dialog, select **Internet Application** from the available templates, ensure **View Engine** is set to**Razor**, and then click **OK**.
5. When the new project opens, right-click the **TestApp** project in **Solution Explorer** and select the **Properties** option.
6. On the project’s properties page, click on the **Web** tab on the left and ensure that the **Use Local IIS Web Server** option is selected.

[**Step 2 – Configure ASP.NET MVC Application for Claims-Based Authentication**](javascript:void(0))

In this step you will add configuration entries to the Web.config configuration file of your ASP.NET MVC web application to make it claims-aware.

**To configure ASP.NET MVC application for claims-based authentication**

1. Add the following configuration section definitions to the Web.config configuration file. These define configuration sections required by Windows Identity Foundation. Add the definitions immediately after the **<configuration>** opening element:

Xml

<configSections>

<section name="system.identityModel" type="System.IdentityModel.Configuration.SystemIdentityModelSection, System.IdentityModel, Version=4.0.0.0, Culture=neutral, PublicKeyToken=B77A5C561934E089" />

<section name="system.identityModel.services" type="System.IdentityModel.Services.Configuration.SystemIdentityModelServicesSection, System.IdentityModel.Services, Version=4.0.0.0, Culture=neutral, PublicKeyToken=B77A5C561934E089" />

</configSections>

1. Add a **<location>** element that enables access to the application’s federation metadata:

Xml

<location path="FederationMetadata">

<system.web>

<authorization>

<allow users="\*" />

</authorization>

</system.web>

</location>

1. Add the following configuration entries within the **<system.web>** elements to deny users, disable native authentication, and enable WIF to manage authentication.

Xml

<authorization>

<deny users="?" />

</authorization>

<authentication mode="None" />

1. Add the following Windows Identity Foundation related configuration entries and ensure that your ASP.NET application’s URL and port number match the values in the **<audienceUris>** entry, **realm** attribute of the **<wsFederation>** element, and the **reply**attribute of the **<wsFederation>** element. Also ensure that the **issuer** value fits your Security Token Service (STS) URL.

Xml

<system.identityModel>

<identityConfiguration>

<audienceUris>

<add value="http://localhost:28503/" />

</audienceUris>

<issuerNameRegistry type="System.IdentityModel.Tokens.ConfigurationBasedIssuerNameRegistry, System.IdentityModel, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089">

<trustedIssuers>

<add thumbprint="1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ1234" name="YourSTSName" />

</trustedIssuers>

</issuerNameRegistry>

<certificateValidation certificateValidationMode="None" />

</identityConfiguration>

</system.identityModel>

<system.identityModel.services>

<federationConfiguration>

<cookieHandler requireSsl="false" />

<wsFederation passiveRedirectEnabled="true" issuer="http://localhost:13922/wsFederationSTS/Issue" realm="http://localhost:28503/" reply="http://localhost:28503/" requireHttps="false" />

</federationConfiguration>

</system.identityModel.services>

1. Add reference to the [System.IdentityModel] assembly.
2. Compile the solution to make sure there are errors.

[**Step 3 – Test Your Solution**](javascript:void(0))

In this step you will test your ASP.NET MVC web application configured for claims-based authentication. To perform basic test you will add simple code that displays claims in the token issued by the Security Token Service (STS).

**To test your ASP.NET MVC application for claims-based authentication**

1. In the **Solution Explorer**, expand the **Controllers** folder and open HomeController.cs file in the editor. Add the following code to the **Index** method:

C#

public ActionResult Index()

{

ViewBag.ClaimsIdentity = Thread.CurrentPrincipal.Identity;

return View();

}

1. In the **Solution Explorer** expand **Views** and then **Home** folders and open Index.cshtml file in the editor. Delete its contents and add the following markup:

HTML

@{

ViewBag.Title = "Home Page";

}

<h2>Welcome: @ViewBag.ClaimsIdentity.Name</h2>

<h3>Values from Identity</h3>

<table>

<tr>

<th>

IsAuthenticated

</th>

<td>

@ViewBag.ClaimsIdentity.IsAuthenticated

</td>

</tr>

<tr>

<th>

Name

</th>

<td>

@ViewBag.ClaimsIdentity.Name

</td>

</tr>

</table>

<h3>Claims from ClaimsIdentity</h3>

<table>

<tr>

<th>

Claim Type

</th>

<th>

Claim Value

</th>

<th>

Value Type

</th>

<th>

Subject Name

</th>

<th>

Issuer Name

</th>

</tr>

@foreach (System.Security.Claims.Claim claim in ViewBag.ClaimsIdentity.Claims ) {

<tr>

<td>

@claim.Type

</td>

<td>

@claim.Value

</td>

<td>

@claim.ValueType

</td>

<td>

@claim.Subject.Name

</td>

<td>

@claim.Issuer

</td>

</tr>

}

</table>

1. Run the solution by pressing the **F5** key.
2. You should be presented with the page that displays the claims in the token that was issued to you by Security Token Service.

### How To: Build Claims-Aware ASP.NET Web Forms Application Using WIF

**.NET Framework 4.6 and 4.5**

[**Applies To**](javascript:void(0))

* Microsoft® Windows® Identity Foundation (WIF)
* ASP.NET® Web Forms

[**Summary**](javascript:void(0))

This How-To provides detailed step-by-step procedures for creating simple claims-aware ASP.NET Web Forms application. It also provides instructions for how to test the simple claims-aware ASP.NET Web Forms application for successful implementation of federated authentication. This How-To does not have detailed instructions for creating a Security Token Service (STS), and assumes you have already configured an STS.

[**Contents**](javascript:void(0))

* Objectives
* Summary of Steps
* Step 1 – Create a Simple ASP.NET Web Forms Application
* Step 2 – Configure ASP.NET Web Forms Application for Claims-Based Authentication
* Step 3 – Test Your Solution

[**Objectives**](javascript:void(0))

* Configure ASP.NET Web Forms application for claims-based authentication
* Test successful claims-aware ASP.NET Web Forms application

[**Summary of Steps**](javascript:void(0))

* Step 1 – Create Simple ASP.NET Web Forms Application
* Step 2 – Configure ASP.NET Web Forms Application for Federated Authentication
* Step 3 – Test Your Solution

[**Step 1 – Create a Simple ASP.NET Web Forms Application**](javascript:void(0))

In this step, you will create a new ASP.NET Web Forms application.

**To create a simple ASP.NET application**

1. Start Visual Studio and click **File**, **New**, and then **Project**.
2. In the **New Project** window, click **ASP.NET Web Forms Application**.
3. In **Name**, enter TestApp and press **OK**.

[**Step 2 – Configure ASP.NET Web Forms Application for Claims-Based Authentication**](javascript:void(0))

In this step you will add configuration entries to the Web.config configuration file of your ASP.NET Web Forms application to make it claims-aware.

**To configure ASP.NET application for claims-based authentication**

1. Add the following configuration section entries to the Web.config configuration file immediately after the **<configuration>** opening element:

Xml

<configSections>

<section name="system.identityModel" type="System.IdentityModel.Configuration.SystemIdentityModelSection, System.IdentityModel, Version=4.0.0.0, Culture=neutral, PublicKeyToken=B77A5C561934E089" />

<section name="system.identityModel.services" type="System.IdentityModel.Services.Configuration.SystemIdentityModelServicesSection, System.IdentityModel.Services, Version=4.0.0.0, Culture=neutral, PublicKeyToken=B77A5C561934E089" />

</configSections>

1. Add a **<location>** element that enables access to the application’s federation metadata:

Xml

<location path="FederationMetadata">

<system.web>

<authorization>

<allow users="\*" />

</authorization>

</system.web>

</location>

1. Add the following configuration entries within the **<system.web>** elements to deny users, disable native authentication, and enable WIF to manage authentication.

Xml

<authorization>

<deny users="?" />

</authorization>

<authentication mode="None" />

1. Add a **<system.webServer>** element that defines the modules for federated authentication. Note that the PublicKeyToken attribute must be the same as the PublicKeyToken attribute for the **<configSections>** entries added earlier:
2. <system.webServer>
3. <modules>
4. <add name="WSFederationAuthenticationModule" type="System.IdentityModel.Services.WSFederationAuthenticationModule, System.IdentityModel.Services, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" preCondition="managedHandler" />
5. <add name="SessionAuthenticationModule" type="System.IdentityModel.Services.SessionAuthenticationModule, System.IdentityModel.Services, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" preCondition="managedHandler" />
6. </modules>
7. </system.webServer>
8. Add the following Windows Identity Foundation related configuration entries and ensure that your ASP.NET application’s URL and port number match the values in the **<audienceUris>** entry, **realm** attribute of the **<wsFederation>** element, and the **reply**attribute of the **<wsFederation>** element. Also ensure that the **issuer** value fits your Security Token Service (STS) URL.

Xml

<system.identityModel>

<identityConfiguration>

<audienceUris>

<add value="http://localhost:28503/" />

</audienceUris>

<issuerNameRegistry type="System.IdentityModel.Tokens.ConfigurationBasedIssuerNameRegistry, System.IdentityModel, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089">

<trustedIssuers>

<add thumbprint="1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ1234" name="YourSTSName" />

</trustedIssuers>

</issuerNameRegistry>

<certificateValidation certificateValidationMode="None" />

</identityConfiguration>

</system.identityModel>

<system.identityModel.services>

<federationConfiguration>

<cookieHandler requireSsl="false" />

<wsFederation passiveRedirectEnabled="true" issuer="http://localhost:13922/wsFederationSTS/Issue" realm="http://localhost:28503/" reply="http://localhost:28503/" requireHttps="false" />

</federationConfiguration>

</system.identityModel.services>

1. Add reference to the [System.IdentityModel] assembly.
2. Compile the solution to make sure there are errors.

[**Step 3 – Test Your Solution**](javascript:void(0))

In this step you will test your ASP.NET Web Forms application configured for claims-based authentication. To perform a basic test, you will add code that displays claims in the token issued by the Security Token Service (STS).

**To test your ASP.NET Web Form application for claims-based authentication**

1. Open the **Default.aspx** file under the **TestApp** project and replace its existing markup with the following markup:
2. %@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="\_Default" %>
3. <!DOCTYPE html>
4. <html xmlns="http://www.w3.org/1999/xhtml">
5. <head id="Head1" runat="server">
6. <title></title>
7. </head>
8. <body>
9. <h1><asp:label ID="signedIn" runat="server" /></h1>
10. <asp:label ID="claimType" runat="server" />
11. <asp:label ID="claimValue" runat="server" />
12. <asp:label ID="claimValueType" runat="server" />
13. <asp:label ID="claimSubjectName" runat="server" />
14. <asp:label ID="claimIssuer" runat="server" />
15. </body>
16. </html>
17. Save **Default.aspx**, and then open its code behind file named **Default.aspx.cs**.

|  |
| --- |
| **Note Note** |
| **Default.aspx.cs** may be hidden beneath **Default.aspx** in Solution Explorer. If **Default.aspx.cs** is not visible, expand **Default.aspx**by clicking on the triangle next to it. |

1. Replace the existing code in the **Page\_Load** method of **Default.aspx.cs** with the following code:

C#

using System;

using System.IdentityModel;

using System.Security.Claims;

using System.Threading;

using System.Web.UI;

namespace TestApp

{

public partial class \_Default : System.Web.UI.Page

{

protected void Page\_Load(object sender, EventArgs e)

{

ClaimsPrincipal claimsPrincipal = Thread.CurrentPrincipal as ClaimsPrincipal;

if (claimsPrincipal != null)

{

signedIn.Text = "You are signed in.";

foreach (Claim claim in claimsPrincipal.Claims)

{

claimType.Text = claim.Type;

claimValue.Text = claim.Value;

claimValueType.Text = claim.ValueType;

claimSubjectName.Text = claim.Subject.Name;

claimIssuer.Text = claim.Issuer;

}

}

else

{

signedIn.Text = "You are not signed in.";

}

}

}

}

1. Save **Default.aspx.cs**, and build the solution.
2. Run the solution by pressing the **F5** key.
3. You should be presented with the page that displays the claims in the token that was issued to you by the Security Token Service.

### How To: Build Claims-Aware ASP.NET Application Using Forms-Based Authentication

**.NET Framework 4.6 and 4.5**

[**Applies To**](javascript:void(0))

* Microsoft® Windows® Identity Foundation (WIF)
* ASP.NET® Web Forms

[**Summary**](javascript:void(0))

This How-To provides detailed step-by-step procedures for creating a simple claims-aware ASP.NET Web Forms application that uses Forms authentication. It also provides instructions for how to test the application to verify that claims are presented when a user signs in with Forms authentication.

[**Contents**](javascript:void(0))

* Objectives
* Overview
* Summary of Steps
* Step 1 – Create a Simple ASP.NET Web Forms Application
* Step 2 – Configure ASP.NET Web Forms Application for Claims Using Forms Authentication
* Step 3 – Test Your Solution

[**Objectives**](javascript:void(0))

* Configure an ASP.NET Web Forms application for claims using Forms authentication
* Test the ASP.NET Web Forms application to see if it is working properly

[**Overview**](javascript:void(0))

In .NET 4.5, WIF and its claims-based authorization have been included as an integral part of the Framework. Previously, if you wanted claims from an ASP.NET user, you were required to install WIF, and then cast interfaces to Principal objects such asThread.CurrentPrincipal or HttpContext.Current.User. Now, claims are served automatically by these Principal objects.

Forms authentication has benefited from WIF’s inclusion in .NET 4.5 because all users authenticated by Forms automatically have claims associated with them. You can begin using these claims immediately in an ASP.NET application that uses Forms authentication, as this How-To demonstrates.

[**Summary of Steps**](javascript:void(0))

* Step 1 – Create a Simple ASP.NET Web Forms Application
* Step 2 – Configure ASP.NET Web Forms Application for Claims Using Forms Authentication
* Step 3 – Test Your Solution

[**Step 1 – Create a Simple ASP.NET Web Forms Application**](javascript:void(0))

In this step, you will create a new ASP.NET Web Forms application.

**To create a simple ASP.NET application**

1. Start Visual Studio and click **File**, **New**, and then **Project**.
2. In the **New Project** window, click **ASP.NET Web Forms Application**.
3. In **Name**, enter TestApp and press **OK**.

[**Step 2 – Configure ASP.NET Web Forms Application for Claims Using Forms Authentication**](javascript:void(0))

In this step you will add a configuration entry to the Web.config configuration file and edit the Default.aspx file to display claims information for an account.

**To configure ASP.NET application for claims using Forms authentication**

1. In the Default.aspx file, replace the existing markup with the following:
2. <%@ Page Title="Home Page" Language="C#" MasterPageFile="~/Site.Master" AutoEventWireup="true" CodeBehind="Default.aspx.cs" Inherits="TestApp.\_Default" %>
3. <asp:Content runat="server" ID="BodyContent" ContentPlaceHolderID="MainContent">
4. <p>
5. This page displays the claims associated with a Forms authenticated user.
6. </p>
7. <h3>Your Claims</h3>
8. <p>
9. <asp:GridView ID="ClaimsGridView" runat="server" CellPadding="3">
10. <AlternatingRowStyle BackColor="White" />
11. <HeaderStyle BackColor="#7AC0DA" ForeColor="White" />
12. </asp:GridView>
13. </p>
14. </asp:Content>

This step adds a GridView control to your Default.aspx page that will be populated with the claims retrieved from Forms authentication.

1. Save the Default.aspx file, then open its code-behind file named Default.aspx.cs. Replace the existing code with the following:

C#

using System;

using System.Web.UI;

using System.Security.Claims;

namespace TestApp

{

public partial class \_Default : Page

{

protected void Page\_Load(object sender, EventArgs e)

{

ClaimsPrincipal claimsPrincipal = Page.User as ClaimsPrincipal;

if (claimsPrincipal != null)

{

this.ClaimsGridView.DataSource = claimsPrincipal.Claims;

this.ClaimsGridView.DataBind();

}

}

}

}

The above code will display claims about an authenticated user, including users identified by Forms authentication.

[**Step 3 – Test Your Solution**](javascript:void(0))

In this step you will test your ASP.NET Web Forms application, and verify that claims are presented when a user signs in with Forms authentication.

**To test your ASP.NET Web Forms application for claims using Forms authentication**

1. Press **F5** to build and run the application. You should be presented with Default.aspx, which has **Register** and **Log in** links in the top right of the page. Click **Register**.
2. On the **Register** page, create a user account, and then click **Register**. Your account will be created using Forms authentication, and you will be automatically signed in.
3. After you have been redirected to the home page, you should see a table beneath the **Your Claims** heading that includes the **Issuer**,**OriginalIssuer**, **Type**, **Value**, and **ValueType** claims information about your account.

### How To: Build Claims-Aware ASP.NET Application Using Windows Authentication

**.NET Framework 4.6 and 4.5**

[**Applies To**](javascript:void(0))

* Microsoft® Windows® Identity Foundation (WIF)
* ASP.NET® Web Forms

[**Summary**](javascript:void(0))

This How-To provides detailed step-by-step procedures for creating a simple claims-aware ASP.NET Web Forms application that uses Windows authentication. It also provides instructions for how to test the application to verify that claims are presented when a user signs in using Windows authentication.

[**Contents**](javascript:void(0))

* Objectives
* Overview
* Summary of Steps
* Step 1 – Create a Simple ASP.NET Web Forms Application
* Step 2 – Configure ASP.NET Web Forms Application for Claims Using Windows Authentication
* Step 3 – Test Your Solution

[**Objectives**](javascript:void(0))

* Configure an ASP.NET Web Forms application for claims using Windows authentication
* Test the ASP.NET Web Forms application to see if it is working properly

[**Overview**](javascript:void(0))

In .NET 4.5, WIF and its claims-based authorization have been included as an integral part of the Framework. Previously, if you wanted claims from an ASP.NET user, you were required to install WIF, and then cast interfaces to Principal objects such asThread.CurrentPrincipal or HttpContext.Current.User. Now, claims are served automatically by these Principal objects.

Windows authentication has benefited from WIF’s inclusion in .NET 4.5 because all users authenticated by Windows credentials automatically have claims associated with them. You can begin using these claims immediately in an ASP.NET application that uses Windows authentication, as this How-To demonstrates.

[**Summary of Steps**](javascript:void(0))

* Step 1 – Create a Simple ASP.NET Web Forms Application
* Step 2 – Configure ASP.NET Web Forms Application for Claims Using Windows Authentication
* Step 3 – Test Your Solution

[**Step 1 – Create a Simple ASP.NET Web Forms Application**](javascript:void(0))

**In this step, you will create a new ASP.NET Web Forms application.**

**To create a simple ASP.NET application**

1. Start Visual Studio, then click **File**, **New**, and then **Project**.
2. In the **New Project** window, click **ASP.NET Web Forms Application**.
3. In **Name**, enter TestApp and press **OK**.
4. After the **TestApp** project has been created, click on it in **Solution Explorer**. The project’s properties will appear in the **Properties**pane below **Solution Explorer**. Set the **Windows Authentication** property to **Enabled**.

|  |
| --- |
| **Caution note Caution** |
| Windows authentication is disabled by default in new ASP.NET applications, so you must manually enable it. |

[**Step 2 – Configure ASP.NET Web Forms Application for Claims Using Windows Authentication**](javascript:void(0))

In this step you will add a configuration entry to the Web.config configuration file and modify the Default.aspx file to display claims information for an account.

**To configure ASP.NET application for claims using Windows authentication**

1. In the **TestApp** project’s Default.aspx file, replace the existing markup with the following:
2. <%@ Page Title="Home Page" Language="C#" MasterPageFile="~/Site.Master" AutoEventWireup="true"
3. CodeBehind="Default.aspx.cs" Inherits="TestApp.\_Default" %>
4. <asp:Content runat="server" ID="BodyContent" ContentPlaceHolderID="MainContent">
5. <p>
6. This page displays the claims associated with a Windows authenticated user.
7. </p>
8. <h3>Your Claims</h3>
9. <p>
10. <asp:GridView ID="ClaimsGridView" runat="server" CellPadding="3">
11. <AlternatingRowStyle BackColor="White" />
12. <HeaderStyle BackColor="#7AC0DA" ForeColor="White" />
13. </asp:GridView>
14. </p>
15. </asp:Content>

This step adds a GridView control to your Default.aspx page that will be populated with the claims retrieved from Windows authentication.

1. Save the Default.aspx file, then open its code-behind file named Default.aspx.cs. Replace the existing code with the following:

C#

using System;

using System.Web.UI;

using System.Security.Claims;

namespace TestApp

{

public partial class \_Default : Page

{

protected void Page\_Load(object sender, EventArgs e)

{

ClaimsPrincipal claimsPrincipal = Page.User as ClaimsPrincipal;

this.ClaimsGridView.DataSource = claimsPrincipal.Claims;

this.ClaimsGridView.DataBind();

}

}

}

The above code will display claims about an authenticated user.

1. To change the application’s authentication type, modify the **<authentication>** block in the **<system.web>** section of the project’s root Web.config file so that it only includes the following configuration entry:
2. <authentication mode="Windows" />
3. Finally, modify the **<authorization>** block in the **<system.web>** section of the same Web.config file to force authentication:
4. <authorization>
5. <deny users="?" />
6. </authorization>

[**Step 3 – Test Your Solution**](javascript:void(0))

In this step you will test your ASP.NET Web Forms application, and verify that claims are presented when a user signs in with Windows authentication.

**To test your ASP.NET Web Forms application for claims using Windows authentication**

* Press **F5** to build and run the application. You should be presented with Default.aspx, and your Windows account name (including domain name) should already appear as the authenticated user in the top right of the page. The page’s content should include a table filled with claims retrieved from your Windows account.

### How To: Debug Claims-Aware Applications And Services Using WIF Tracing

**.NET Framework 4.6 and 4.5**

[**Applies To**](javascript:void(0))

* Microsoft® Windows® Identity Foundation (WIF)
* Service Trace Viewer Tool (SvcTraceViewer.exe)
* Troubleshooting and Debugging WIF Applications

[**Summary**](javascript:void(0))

This How-To describes required steps for how to configure WIF tracing, collect trace logs, and how to analyze the trace logs using Trace Viewer tool. It provides general mapping for trace entries to actions needed to troubleshoot issues related to WIF.

[**Contents**](javascript:void(0))

* Objectives
* Summary of Steps
* Step 1 – Configure WIF Tracing Using Web.config Configuration File
* Step 2 – Analyze WIF Trace Files Using Trace Viewer Tool
* Step 3 – Identify Solutions to Fix WIF Related Issues
* Related Items

[**Objectives**](javascript:void(0))

* Configure WIF tracing.
* View trace logs in the Trace Viewer tool.
* Identify WIF related issues in the trace logs.
* Apply corrective actions to WIF related issues found in the trace logs.

[**Summary of Steps**](javascript:void(0))

* Step 1 – Configure WIF Tracing Using Web.config Configuration File
* Step 2 – Analyze WIF Trace Files Using Trace Viewer Tool
* Step 3 – Identify Solutions to Fix WIF Related Issues

[**Step 1 – Configure WIF Tracing Using Web.config Configuration File**](javascript:void(0))

In this step, you will add changes to configuration sections in the Web.config file that enable WIF to trace its events and store them in a trace log.

**To configure WIF tracing using Web.config configuration file**

1. Open the root **Web.config** or **App.config** configuration file in the Visual Studio editor by double clicking on it in **Solution Explorer**. If your solution does not have **Web.config** or **App.config** configuration file, add it by right clicking on the solution in the**Solution Explorer** and clicking **Add**, then clicking **New Item…**. On the **New Item** dialog, Select **Application Configuration File** for**App.config** or **Web Configuration File** for **Web.config** from the list and click **OK**.
2. Add the configuration entries similar to the following to the configuration file inside **<configuration>** node at the end of the configuration file:

Xml

<system.diagnostics>

<sources>

<source name="System.IdentityModel" switchValue="Verbose">

<listeners>

<add name="xml" type="System.Diagnostics.XmlWriterTraceListener" initializeData="WIFTrace.e2e"/>

</listeners>

</source>

</sources>

<trace autoflush="true"/>

</system.diagnostics>

1. The above configuration instructs WIF to generate verbose trace events and log them into WIFTrace.e2e file. For a complete list of values for the **switchValue** switch, refer to the Trace Level table found in the following topic: [Configuring Tracing](https://msdn.microsoft.com/en-us/library/ms733025.aspx).

[**Step 2 – Analyze WIF Trace Files Using Trace Viewer Tool**](javascript:void(0))

In this step, you will use the Trace Viewer Tool (SvcTraceViewer.exe) to analyze WIF trace logs.

To analyze WIF trace logs using Trace Viewer tool (SvcTraceViewer.exe)

1. Trace Viewer tool (SvcTraceViewer.exe) ships as part of the Windows SDK. If you haven’t already installed the Windows SDK, you can download it here: [Windows SDK](https://www.microsoft.com/download/en/details.aspx?id=8279).
2. Run the Trace Viewer tool (SvcTraceViewer.exe). It is typically available in the **Bin** folder of the installation path.
3. Open the WIF trace log file, for example, WIFTrace.e2e by selecting **File**, **Open…** option in the menu or using the **Ctrl+O** shortcut. The trace log file opens in the Trace Viewer tool.
4. Review entries in the **Activity** tab. Each entry should contain an activity number, the number of traces that were logged, duration of the activity and its start and end timestamps.
5. Click on the **Activity** tab. You should see detailed trace entries in the main area of the tool. Use the **Level** dropdown list on the menu to filter specific level of traces, for example: **All**, **Warning**, **Errors**, **Information**, etc.
6. Click on specific trace entries to review the details in the lower area of the tool. The details can be viewed using **Formatted** and **XML**view by choosing corresponding tabs.

[**Step 3 – Identify Solutions to Fix WIF Related Issues**](javascript:void(0))

In this step, you can identify solutions for WIF-related issues identified by using the WIF trace log and Trace Viewer tool. It outlines general mapping of WIF related exceptions to potential solutions or required actions to troubleshoot the issue.

**To identify solutions to fix WIF related issues**

* Review the following table of WIF exceptions and the required actions to correct the issues.

|  |  |  |
| --- | --- | --- |
| **Error ID** | **Error Message** | **Action needed to fix the error** |
| ID4175 | The issuer of the security token was not recognized by the IssuerNameRegistry. To accept security tokens from this issuer, configure the IssuerNameRegistry to return a valid name for this issuer. | This error can be caused by copying a thumbprint from the MMC snap-in and pasting it into the Web.config file. Specifically, you can get an extra non-printable character in the text string when copying from the certificate properties window. This extra character causes the thumbprint match to fail.The procedure for correctly copying the thumbprint can be found here: [http://msdn.microsoft.com/en-us/library/ff359102.aspx](https://msdn.microsoft.com/en-us/library/ff359102.aspx) |

How To: Display Signed In Status Using WIF

**.NET Framework 4.6 and 4.5**

[**Applies To**](javascript:void(0))

* Microsoft® Windows® Identity Foundation (WIF) 4.5
* ASP.NET® Web Forms

[**Summary**](javascript:void(0))

This topic describes how to display the sign in status in a WIF-enabled ASP.NET application. WIF provides the mechanism for making your application claims-aware, and managing authentication and authorization for application resources.

[**Contents**](javascript:void(0))

* Overview
* Summary of Steps
* Step 1 – Install the Identity and Access Extension
* Step 2 – Create a Relying Party ASP.NET Application
* Step 3 – Enable Local Development STS to Authenticate Users
* Step 4 – Modify Your ASP.NET Application to Display Sign In Status
* Step 5 – Test the Integration Between WIF and Your ASP.NET Application

[**Overview**](javascript:void(0))

This topic demonstrates how to create a simple claims-aware application using WIF and how to easily display whether a user is signed in or not. The following steps use the Local Development STS that is included with the Identity and Access Visual Studio Extension. The Local Development STS is intended for a testing and development environment to provide a simple method of integrating claims into your application. It should never be used in a production environment, as it does not perform real authentication and credentials are not required. However, the imperative code in the following steps is the same for a production-ready application using real authentication.

[**Summary of Steps**](javascript:void(0))

* Step 1 – Install the Identity and Access Extension
* Step 2 – Create a Relying Party ASP.NET Application
* Step 3 – Enable Local Development STS to Authenticate Users
* Step 4 – Modify Your ASP.NET Application to Display Sign In Status
* Step 5 – Test the Integration Between WIF and Your ASP.NET Application

[**Step 1 – Install the Identity and Access Extension**](javascript:void(0))

This step describes how to configure the Identity and Access extension to Visual Studio 2012. This extension automates the process of configuring your application to communicate with STS endpoints.

**To install the Identity and Access extension**

1. Start Visual Studio in elevated mode as administrator.
2. In Visual Studio, click **Tools** and click **Extension Manager**. The **Extension Manager** window appears.
3. In **Extension Manager**, click **Online Extensions** from the left menu, then select **Visual Studio Gallery**.
4. In the top right corner of **Extension Manager**, search for Identity and Access.
5. The **Identity and Access** item will appear in the search results. Click it, and then click **Download**.
6. The **Download and Install** dialog appears. If you agree with the license terms, click **Install**.
7. When the **Identity and Access** extension has finished installing, restart Visual Studio in administrator mode.

[**Step 2 – Create a Relying Party ASP.NET Application**](javascript:void(0))

This step describes how to create a relying party ASP.NET Web Forms application that will integrate with WIF.

**To create a simple ASP.NET application**

1. Start Visual Studio and click **File**, **New**, and then **Project**.
2. In the **New Project** window, click **ASP.NET Web Forms Application**.
3. In **Name**, enter TestApp and press **OK**.

[**Step 3 – Enable Local Development STS to Authenticate Users**](javascript:void(0))

This step describes how to enable Local Development STS in your application. Local Development STS is enabled by using the Identity and Access extension for Visual Studio.

**To enable Local Development STS in your ASP.NET application**

1. In Visual Studio, right-click the **TestApp** project under **Solution Explorer**, then select **Identity and Access**.
2. The **Identity and Access** window appears. Under **Providers**, select **Test your application with the Local Development STS**, then click **Apply**.

[**Step 4 – Modify Your ASP.NET Application to Display Sign In Status**](javascript:void(0))

This step describes how to modify your ASP.NET application to dynamically display whether the current user is signed in. Once your STS provider has been configured, WIF handles the incoming claims. Now you need to configure your application’s code to display the result of the authentication.

**To display sign in status**

1. In Visual Studio, open the **Default.aspx** file under the **TestApp** project.
2. Replace the existing markup in the **Default.aspx** file with the following markup:
3. <%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="\_Default" %>
4. <!DOCTYPE html>
5. <html xmlns="http://www.w3.org/1999/xhtml">
6. <head runat="server">
7. <title>Logged In Status</title>
8. </head>
9. <body>
10. <asp:label ID="myLabel" runat="server" />
11. </body>
12. </html>
13. Save **Default.aspx**, and then open its code behind file named **Default.aspx.cs**.

|  |
| --- |
| **Note Note** |
| **Default.aspx.cs** may be hidden beneath **Default.aspx** in Solution Explorer. If **Default.aspx.cs** is not visible, expand **Default.aspx**by clicking on the triangle next to it. |

1. Replace the existing code in **Default.aspx.cs** with the following code:

C#

using System;

using System.Web.UI;

using System.Security.Claims;

namespace TestApp

{

protected void Page\_Load(object sender, EventArgs e)

{

ClaimsPrincipal claimsPrincipal = Thread.CurrentPrincipal as ClaimsPrincipal;

if (claimsPrincipal != null)

{

myLabel.Text = "You are signed in.";

}

else

{

myLabel.Text = "You are not signed in.";

}

}

}

1. Save **Default.aspx.cs**, and build the application.

[**Step 5 – Test the Integration Between WIF and Your ASP.NET Application**](javascript:void(0))

This step describes how you can test the integration between WIF and your ASP.NET application.

**To test the integration between WIF and ASP.NET**

1. In Visual Studio, press **F5** to start debugging your application. If no errors are found, a new browser window will open.
2. You may notice that the browser silently redirects your request to the STS, and then opens the Default.aspx page. If WIF is properly configured, you should see the site display the following text: **“You are signed in”**.

### How To: Enable WIF Tracing

**.NET Framework 4.6 and 4.5**

[**Applies To**](javascript:void(0))

* Microsoft® Windows® Identity Foundation (WIF)
* ASP.NET® Web Forms

[**Summary**](javascript:void(0))

This How-To provides detailed step-by-step procedures for enabling WIF tracing in an ASP.NET application. It also provides instructions testing the application to verify that the trace listener and log are working correctly. This How-To does not have detailed instructions for creating a Security Token Service (STS), and instead uses the Development STS that comes with the Identity and Access tool. The Development STS does not perform real authentication and is intended for testing purposes only. You will need to install the Identity and Access tool to complete this How-To. It can be downloaded from the following location: [Identity and Access Tool](http://go.microsoft.com/fwlink/?LinkID=245849)

|  |
| --- |
| **Security note Security Note** |
| Enabling WIF tracing for passive applications, that is, applications that use the WS-Federation protocol, can potentially expose the application to denial of service (DoS) attacks or to information disclosure to a malicious party. This includes both passive RPs and passive STSes. For this reason, we recommend that you not enable WIF tracing for passive RPs or STSes in a production environment. |

[**Contents**](javascript:void(0))

* Objectives
* Overview
* Summary of Steps
* Step 1 – Create a Simple ASP.NET Web Forms Application and Enable Tracing
* Step 2 – Test Your Solution

[**Objectives**](javascript:void(0))

* Create a simple ASP.NET application that uses WIF and the Development STS from the Identity and Access Tool
* Enable tracing and verify that it is working

[**Overview**](javascript:void(0))

Tracing enables you to debug and troubleshoot many types of issues with WIF, including tokens, cookies, claims, protocol messages, and more. WIF tracing is similar to WCF tracing; for example, you can choose the verbosity of traces to display everything from critical messages to all messages. WIF traces can be generated in **.xml** files or in **.svclog** files that are viewable by using the Service Trace Viewer Tool. This tool is located in the **bin** directory of the Windows SDK install path on your computer, for example: **C:\Program Files\Microsoft SDKs\Windows\v7.1\Bin\SvcTraceViewer.exe**.

[**Summary of Steps**](javascript:void(0))

* Step 1 – Create a Simple ASP.NET Web Forms Application and Enable Tracing
* Step 2 – Test Your Solution

[**Step 1 – Create a Simple ASP.NET Web Forms Application and Enable Tracing**](javascript:void(0))

In this step, you will create a new ASP.NET Web Forms application and modify the Web.config file to enable tracing.

**To create a simple ASP.NET application**

1. Start Visual Studio and click **File**, **New**, and then **Project**.
2. In the **New Project** window, click **ASP.NET Web Forms Application**.
3. In **Name**, enter TestApp and press **OK**.
4. Right-click the **TestApp** project under **Solution Explorer**, then select **Identity and Access**.
5. The **Identity and Access** window appears. Under **Providers**, select **Test your application with the Local Development STS**, then click **Apply**.
6. Create a new folder in named **logs** in the root of the **C:** drive, like shown: **C:\logs**
7. Add the following **<system.diagnostics>** element to the Web.config configuration file immediately following the closing**</configSections>** element, like shown:
8. <configuration>
9. <configSections>
10. …
11. </configSections>
12. <system.diagnostics>
13. <sources>
14. <source name="System.IdentityModel" switchValue="Verbose">
15. <listeners>
16. <add name="xml" type="System.Diagnostics.XmlWriterTraceListener" initializeData="C:\logs\WIF.xml" />
17. </listeners>
18. </source>
19. </sources>
20. <trace autoflush="true" />
21. </system.diagnostics>

|  |
| --- |
| **NoteNote** |
| The directory location specified in the **initializeData** attribute must exist before logging can begin. If the location does not exist, no logs will be created. |

1. The configuration settings above will enable **Verbose** tracing for WIF and save the resulting log to the **C:\logs\WIF.xml** file.

[**Step 2 – Test Your Solution**](javascript:void(0))

In this step, you will test your WIF-enabled ASP.NET application to verify that logs are being recorded.

To test your WIF-enabled ASP.NET application for successful tracing

1. Run the solution by pressing the **F5** key. You should be presented with the default ASP.NET Home Page and automatically authenticated with the username Terry, which is the default user that is returned by the Development STS.
2. Close the browser window and then navigate to the **C:\logs** folder. Open the **C:\logs\WIF.xml** file using a text editor.
3. Inspect the **WIF.xml** file and verify that it contains entries starting with **<E2ETraceEvent>**. These traces will contain **<TraceRecord>**elements with descriptions for the traced activity, such as **Validating SecurityToken**.

### How To: Enable Token Replay Detection

**.NET Framework 4.6 and 4.5**

[**Applies To**](javascript:void(0))

* Microsoft® Windows® Identity Foundation (WIF)
* ASP.NET® Web Forms

[**Summary**](javascript:void(0))

This How-To provides detailed step-by-step procedures for enabling token replay detection in an ASP.NET application that uses WIF. It also provides instructions for how to test the application to verify that token replay detection is enabled. This How-To does not have detailed instructions for creating a Security Token Service (STS), and instead uses the Development STS that comes with the Identity and Access tool. The Development STS does not perform real authentication and is intended for testing purposes only. You will need to install the Identity and Access tool to complete this How-To. It can be downloaded from the following location: [Identity and Access Tool](http://go.microsoft.com/fwlink/?LinkID=245849)

[**Contents**](javascript:void(0))

* Objectives
* Overview
* Summary of Steps
* Step 1 – Create a Simple ASP.NET Web Forms Application and Enable Replay Detection
* Step 2 – Test Your Solution

[**Objectives**](javascript:void(0))

* Create a simple ASP.NET application that uses WIF and the Development STS from the Identity and Access Tool
* Enable token replay detection and verify that it is working

[**Overview**](javascript:void(0))

A replay attack occurs when a client attempts to authenticate to a relying party with an STS token that the client has already used. To help prevent this attack, WIF contains a replay detection cache of previously used STS tokens. When enabled, replay detection checks the token of the incoming request and verifies whether or not the token has been previously used. If the token has been used already, the request is refused and a [SecurityTokenReplayDetectedException](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.securitytokenreplaydetectedexception(v=vs.110).aspx) exception is thrown.

The following steps demonstrate the configuration changes required to enable replay detection.

[**Summary of Steps**](javascript:void(0))

* Step 1 – Create a Simple ASP.NET Web Forms Application and Enable Replay Detection
* Step 2 – Test Your Solution

[**Step 1 – Create a Simple ASP.NET Web Forms Application and Enable Replay Detection**](javascript:void(0))

In this step, you will create a new ASP.NET Web Forms application and modify the Web.config file to enable replay detection.

**To create a simple ASP.NET application**

1. Start Visual Studio and click **File**, **New**, and then **Project**.
2. In the **New Project** window, click **ASP.NET Web Forms Application**.
3. In **Name**, enter TestApp and press **OK**.
4. Right-click the **TestApp** project under **Solution Explorer**, then select **Identity and Access**.
5. The **Identity and Access** window appears. Under **Providers**, select **Test your application with the Local Development STS**, then click **Apply**.
6. Add the following **<tokenReplayDetection>** element to the Web.config configuration file immediately following the**<system.identityModel>** and **<identityConfiguration>** elements, like shown:
7. <system.identityModel>
8. <identityConfiguration>
9. <tokenReplayDetection enabled=”true”/>

[**Step 2 – Test Your Solution**](javascript:void(0))

In this step, you will test your WIF-enabled ASP.NET application to verify that replay detection has been enabled.

**To test your WIF-enabled ASP.NET application for replay detection**

1. Run the solution by pressing the **F5** key. You should be presented with the default ASP.NET Home Page and automatically authenticated with the username Terry, which is the default user that is returned by the Development STS.
2. Press the browser’s **Back** button. You should be presented with a **Server Error in ‘/’ Application** page with the following description: ID1062: Replay has been detected for: Token: 'System.IdentityModel.Tokens.SamlSecurityToken', followed by an AssertionIdand an Issuer.

You are seeing this error page because an exception of type [SecurityTokenReplayDetectedException](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.securitytokenreplaydetectedexception(v=vs.110).aspx) was thrown when the token replay was detected. This error occurs because you are attempting to re-send the initial POST request when the token was first presented. The **Back** button will not cause this behavior on subsequent requests to the server.

### How To: Enable WIF for a WCF Web Service Application

**.NET Framework 4.6 and 4.5**

[**Applies To**](javascript:void(0))

* Microsoft® Windows® Identity Foundation (WIF)
* Microsoft® Windows® Communication Foundation (WCF)

[**Summary**](javascript:void(0))

This How-To provides detailed step-by-step procedures for enabling WIF in a WCF web service. It also provides instructions for how to test the application to verify that the web service is correctly presenting claims when the application is run. This How-To does not have detailed instructions for creating a Security Token Service (STS), and instead uses the Development STS that comes with the Identity and Access tool. The Development STS does not perform real authentication and is intended for testing purposes only. You will need to install the Identity and Access tool to complete this How-To. It can be downloaded from the following location: [Identity and Access Tool](http://go.microsoft.com/fwlink/?LinkID=245849)

[**Contents**](javascript:void(0))

* Objectives
* Overview
* Summary of Steps
* Step 1 – Create a Simple WCF Service
* Step 2 – Create a Client Application for the WCF Service
* Step 3 – Test Your Solution

[**Objectives**](javascript:void(0))

* Create a WCF service that requires issued tokens
* Create a WCF client that requests a token from an STS and passes it to the WCF service

[**Overview**](javascript:void(0))

This How-To is intended to demonstrate how a developer can use federated authentication when developing WCF services. Some of the benefits of using federation in WCF services include:

1. Factoring authentication logic out of WCF service code
2. Re-using existing identity management solutions
3. Interoperability with other identity solutions
4. Flexibility and resilience to future changes

WIF and the associated Identity and Access tool make it easier to develop and test a WCF service using federated authentication, as the following steps demonstrate.

[**Summary of Steps**](javascript:void(0))

* Step 1 – Create a Simple WCF Service
* Step 2 – Create a Client Application for the WCF Service
* Step 3 – Test Your Solution

[**Step 1 – Create a Simple WCF Service**](javascript:void(0))

In this step, you will create a new WCF service that uses the Development STS that is included with the Identity and Access tool.

**To create a simple WCF service**

1. Start Visual Studio in elevated mode as administrator.
2. In Visual Studio, click **File**, click **New**, and then click **Project**.
3. In the **New Project** window, click **WCF Service Application**.
4. In **Name**, enter TestService and press **OK**.
5. Right-click the **TestService** project under **Solution Explorer**, then select **Identity and Access**.
6. The **Identity and Access** window appears. Under **Providers**, select **Test your application with the Local Development STS**, then click **Apply**. The Identity and Access Tool configures the service to use WIF and to outsource authentication to the local development STS (**LocalSTS**) by adding configuration elements to the Web.config file.
7. In the Service1.svc.cs file, add a **using** directive for the **System.Security.Claims** namespace and replace the existing code with the following, then save the file:

C#

public class Service1 : IService1

{

public string ComputeResponse(string input)

{

// Get the caller's identity from ClaimsPrincipal.Current

ClaimsPrincipal claimsPrincipal = OperationContext.Current.ClaimsPrincipal;

// Start generating the output

StringBuilder builder = new StringBuilder();

builder.AppendLine("Computed by ClaimsAwareWebService");

builder.AppendLine("Input received from client:" + input);

if (claimsPrincipal != null)

{

// Display the claims from the caller. Do not use this code in a production application.

ClaimsIdentity identity = claimsPrincipal.Identity as ClaimsIdentity;

builder.AppendLine("Client Name:" + identity.Name);

builder.AppendLine("IsAuthenticated:" + identity.IsAuthenticated);

builder.AppendLine("The service received the following issued claims of the client:");

// Iterate over the caller’s claims and append to the output

foreach (Claim claim in claimsPrincipal.Claims)

{

builder.AppendLine("ClaimType :" + claim.Type + " ClaimValue:" + claim.Value);

}

}

return builder.ToString();

}

}

The ComputeResponse method displays the properties of various claims that are issued by **LocalSTS**.

1. In the IService1.cs file, replace the existing code with the following, then save the file:

C#

[ServiceContract]

public interface IService1

{

[OperationContract]

string ComputeResponse(string input);

}

1. Build the project.
2. Press **Ctrl-F5** to run the service without starting the debugger. A Web page should open for the service and you can verify that**LocalSTS** is running by looking in the notification area (system tray).

|  |
| --- |
| **Important note Important** |
| Both **TestService** and **LocalSTS** must be running when you add the service reference to the client application in the next step. |

[**Step 2 – Create a Client Application for the WCF Service**](javascript:void(0))

In this step, you will create a console application that uses the Development STS to authenticate with the WCF service you created in the previous step.

**To create a client application**

1. In Visual Studio, right-click on the solution, click **Add**, and then click **New Project**.
2. In the **Add New Project** window, select **Console Application** from the **Visual C#** templates list, enter Client, and then press **OK**. The new project will be created in your solution folder.
3. Right-click on **References** under the **Client** project, and then click **Add Service Reference**.
4. In the **Add Service Reference** window, click the drop-down arrow on the **Discover** button and click **Services in Solution**. The**Address** will automatically populate with the WCF service you created earlier, and the **Namespace** will be set to **ServiceReference1**. Click **OK**.

|  |
| --- |
| **Important note Important** |
| Both **TestService** and **LocalSTS** must be running when you add the service reference to the client. |

1. Visual Studio will generate proxy classes for the WCF service, and add all of the necessary reference information. It will also add elements to the App.config file to configure the client to get a token from the STS to authenticate with the service. When this process is finished, the **Program.cs** file will open. Add a **using** directive for **System.ServiceModel** and another for**Client.ServiceReference1**, replace the **Main** method with the following code, then save the file:

C#

static void Main(string[] args)

{

// Create the client for the service

Service1Client client = new Service1Client();

Console.WriteLine("-------------WCF Client Application--------------\n");

while (!ShouldQuitApplication())

{

try

{

Console.WriteLine(client.ComputeResponse("Hello World"));

}

catch (CommunicationException e)

{

Console.WriteLine(e.Message);

Console.WriteLine(e.StackTrace);

Exception ex = e.InnerException;

while (ex != null)

{

Console.WriteLine("===========================");

Console.WriteLine(ex.Message);

Console.WriteLine(ex.StackTrace);

ex = ex.InnerException;

}

}

catch (TimeoutException)

{

Console.WriteLine("Timed out...");

}

catch (Exception e)

{

Console.WriteLine("An unexpected exception occurred.");

Console.WriteLine(e.StackTrace);

}

}

client.Close();

if (client != null)

{

client.Abort();

}

}

static bool ShouldQuitApplication()

{

Console.WriteLine("---------------------------------------------------------------------");

Console.WriteLine("Press Enter key to invoke service, any other key to quit application:");

Console.WriteLine("----------------------------------------------------------------------");

ConsoleKeyInfo keyInfo = Console.ReadKey();

if (keyInfo.Key == ConsoleKey.Enter)

return false;

return true;

}

1. Open the App.config file and add the following XML as the first child element under the <system.serviceModel> element, then save the file:
2. <behaviors>
3. <endpointBehaviors>
4. <behavior>
5. <clientCredentials>
6. <serviceCertificate>
7. <authentication certificateValidationMode="None"/>
8. </serviceCertificate>
9. </clientCredentials>
10. </behavior>
11. </endpointBehaviors>
12. </behaviors>

This disables certificate validation.

1. Right-click the **TestService** solution and click on **Set StartUp Projects**. The **Startup Project** property page opens. In the **Startup Project** property page, select **Multiple startup projects** then click in the **Action** field for each project and select **Start** from the drop-down menu. Click **OK** to save the settings.
2. Build the solution.

[**Step 3 – Test Your Solution**](javascript:void(0))

In this step you will test your WIF-enabled WCF application and verify that claims are presented.

**To test your WIF-enabled WCF application for claims**

1. Press **F5** to build and run the application. You should be presented with a console window, and the following text: **Press Enter key to invoke service, any other key to quit application:**
2. Press **Enter**, and the following claims information should appear in the console:
3. Computed by Service1
4. Input received from client: Hello World
5. Client Name: Terry
6. IsAuthenticated: True
7. The service received the following issued claims of the client:
8. ClaimType :http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name ClaimValue:Terry
9. ClaimType :http://schema.xmlsoap.org/ws/2005/05/identity/claims/surname ClaimValue:Adams
10. ClaimType :http://schemas.microsoft.com/ws/2008/06/identity/claims/role ClaimValue:developer
11. ClaimType :http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress ClaimValue:terry@contoso.com

|  |
| --- |
| **Important noteImportant** |
| Both **TestService** and **LocalSTS** must be running before you press **Enter**. A Web page should open for the service and you can verify that **LocalSTS** is running by looking in the notification area (system tray). |

1. If these claims appear in the console, you have successfully authenticated with the STS to display claims from the WCF service.

### How To: Transform Incoming Claims

**.NET Framework 4.6 and 4.5**

[**Applies To**](javascript:void(0))

* Microsoft® Windows® Identity Foundation (WIF)
* ASP.NET® Web Forms

[**Summary**](javascript:void(0))

This How-To provides detailed step-by-step procedures for creating a simple claims-aware ASP.NET Web Forms application and transforming incoming claims. It also provides instructions for how to test the application to verify that transformed claims are presented when the application is run.

[**Contents**](javascript:void(0))

* Objectives
* Overview
* Summary of Steps
* Step 1 – Create a Simple ASP.NET Web Forms Application
* Step 2 – Implement Claims Transformation Using a Custom ClaimsAuthenticationManager
* Step 3 – Test Your Solution

[**Objectives**](javascript:void(0))

* Configure an ASP.NET Web Forms application for claims-based authentication
* Transform incoming claims by adding an Administrator role claim
* Test the ASP.NET Web Forms application to see if it is working properly

[**Overview**](javascript:void(0))

WIF exposes a class named [ClaimsAuthenticationManager](https://msdn.microsoft.com/en-us/library/system.security.claims.claimsauthenticationmanager(v=vs.110).aspx) that enables users to modify claims before they are presented to a relying party (RP) application. The [ClaimsAuthenticationManager](https://msdn.microsoft.com/en-us/library/system.security.claims.claimsauthenticationmanager(v=vs.110).aspx) is useful for separation of concerns between authentication and the underlying application code. The example below demonstrates how to add a role to the claims in the incoming [ClaimsPrincipal](https://msdn.microsoft.com/en-us/library/system.security.claims.claimsprincipal(v=vs.110).aspx) that may be required by the RP.

[**Summary of Steps**](javascript:void(0))

* Step 1 – Create a Simple ASP.NET Web Forms Application
* Step 2 – Implement Claims Transformation Using a Custom ClaimsAuthenticationManager
* Step 3 – Test Your Solution

[**Step 1 – Create a Simple ASP.NET Web Forms Application**](javascript:void(0))

In this step, you will create a new ASP.NET Web Forms application.

**To create a simple ASP.NET application**

1. Start Visual Studio in elevated mode as administrator.
2. In Visual Studio, click **File**, click **New**, and then click **Project**.
3. In the **New Project** window, click **ASP.NET Web Forms Application**.
4. In **Name**, enter TestApp and press **OK**.
5. Right-click the **TestApp** project under **Solution Explorer**, then select **Identity and Access**.
6. The **Identity and Access** window appears. Under **Providers**, select **Test your application with the Local Development STS**, then click **Apply**.
7. In the Default.aspx file, replace the existing markup with the following, then save the file:
8. <%@ Page Title="Home Page" Language="C#" MasterPageFile="~/Site.Master" AutoEventWireup="true"
9. CodeBehind="Default.aspx.cs" Inherits="TestApp.\_Default" %>
10. <asp:Content runat="server" ID="BodyContent" ContentPlaceHolderID="MainContent">
11. <h3>Your Claims</h3>
12. <p>
13. <asp:GridView ID="ClaimsGridView" runat="server" CellPadding="3">
14. <AlternatingRowStyle BackColor="White" />
15. <HeaderStyle BackColor="#7AC0DA" ForeColor="White" />
16. </asp:GridView>
17. </p>
18. </asp:Content>
19. Open the code-behind file named Default.aspx.cs. Replace the existing code with the following, then save the file:

C#

using System;

using System.Web.UI;

using System.Security.Claims;

namespace TestApp

{

public partial class \_Default : Page

{

protected void Page\_Load(object sender, EventArgs e)

{

ClaimsPrincipal claimsPrincipal = Page.User as ClaimsPrincipal;

this.ClaimsGridView.DataSource = claimsPrincipal.Claims;

this.ClaimsGridView.DataBind();

}

}

}

[**Step 2 – Implement Claims Transformation Using a Custom ClaimsAuthenticationManager**](javascript:void(0))

In this step you will override default functionality in the [ClaimsAuthenticationManager](https://msdn.microsoft.com/en-us/library/system.security.claims.claimsauthenticationmanager(v=vs.110).aspx) class to add an Administrator role to the incoming Principal.

**To implement claims transformation using a custom ClaimsAuthenticationManager**

1. In Visual Studio, right-click the on the solution, click **Add**, and then click **New Project**.
2. In the **Add New Project** window, select **Class Library** from the **Visual C#** templates list, enter ClaimsTransformation, and then press **OK**. The new project will be created in your solution folder.
3. Right-click on **References** under the **ClaimsTransformation** project, and then click **Add Reference**.
4. In the **Reference Manager** window, select **System.IdentityModel**, and then click **OK**.
5. Open **Class1.cs**, or if it doesn’t exist, right-click **ClaimsTransformation**, click **Add**, then click **Class…**
6. Add the following using directives to the code file:

C#

using System.Security.Claims;

using System.Security.Principal;

1. Add the following class and method in the code file.

|  |
| --- |
| **Caution note Caution** |
| The following code is for demonstration purposes only; make sure that you verify your intended permissions in production code. |

1. C#
2. public class ClaimsTransformationModule : ClaimsAuthenticationManager
3. {
4. public override ClaimsPrincipal Authenticate(string resourceName, ClaimsPrincipal incomingPrincipal)
5. {
6. if (incomingPrincipal != null && incomingPrincipal.Identity.IsAuthenticated == true)
7. {
8. ((ClaimsIdentity)incomingPrincipal.Identity).AddClaim(new Claim(ClaimTypes.Role, "Admin"));
9. }
11. return incomingPrincipal;
12. }
13. }
14. Save the file and build the **ClaimsTransformation** project.
15. In your **TestApp** ASP.NET project, right-click on References, and then click **Add Reference**.
16. In the **Reference Manager** window, select **Solution** from the left menu, select **ClaimsTransformation** from the populated options, and then click **OK**.
17. In the root **Web.config** file, navigate to the **<system.identityModel>** entry. Within the **<identityConfiguration>** elements, add the following line and save the file:
18. <claimsAuthenticationManager type="ClaimsTransformation.ClaimsTransformationModule, ClaimsTransformation" />

[**Step 3 – Test Your Solution**](javascript:void(0))

In this step you will test your ASP.NET Web Forms application, and verify that claims are presented when a user signs in with Forms authentication.

**To test your ASP.NET Web Forms application for claims using Forms authentication**

1. Press **F5** to build and run the application. You should be presented with Default.aspx.
2. On the Default.aspx page, you should see a table beneath the **Your Claims** heading that includes the **Issuer**, **OriginalIssuer**, **Type**,**Value**, and **ValueType** claims information about your account. The last row should be presented in the following way:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| LOCAL AUTHORITY | LOCAL AUTHORITY | http://schemas.microsoft.com/ws/2008/06/identity/claims/role | Admin | http://www.w3.org/2001/XMLSchema#string |

## WIF Guidelines

**.NET Framework 4.6 and 4.5**

* [Guidelines for Migrating an Application Built Using WIF 3.5 to WIF 4.5](https://msdn.microsoft.com/en-us/library/jj157089(v=vs.110).aspx)
* [Namespace Mapping between WIF 3.5 and WIF 4.5](https://msdn.microsoft.com/en-us/library/jj157091(v=vs.110).aspx)

### Guidelines for Migrating an Application Built Using WIF 3.5 to WIF 4.5

**.NET Framework 4.6 and 4.5**

[**Applies To**](javascript:void(0))

* Microsoft® Windows® Identity Foundation (WIF) 3.5 and 4.5.

[**Overview**](javascript:void(0))

Windows Identity Foundation (WIF) was originally released in the .NET 3.5 SP1 timeframe. That version of WIF is referred to as WIF 3.5. It was released as a separate runtime and SDK, which meant that every computer on which a WIF-enabled application ran had to have the WIF runtime installed and developers had to download and install the WIF SDK to get the Visual Studio templates and tooling that enabled development of WIF-enabled applications. Beginning with .NET 4.5, WIF has been fully integrated into the .NET Framework. A separate runtime is no longer needed and the WIF tooling can be installed in Visual Studio 2012 by using the Visual Studio Extensions Manager. This version of WIF is referred to as WIF 4.5.

The integration of WIF into .NET necessitated several changes in the WIF API surface. WIF 4.5 includes new namespaces, some changes to configuration elements, and new tooling for Visual Studio. This topic provides guidance that you can use to help you migrate applications built using WIF 3.5 to WIF 4.5. There may be scenarios in which you need to run legacy applications built using WIF 3.5 on computers that are running Windows 8 or Windows Server 2012. This topic also provides guidance about how to enable WIF 3.5 for these operating systems.

[**Changes Required for WIF 4.5**](javascript:void(0))

This section describes the changes that are required to migrate a WIF 3.5 application to WIF 4.5.

[**Assembly and Namespace Changes**](javascript:void(0))

In WIF 3.5, all of the WIF classes were contained in the **Microsoft.IdentityModel** assembly (microsoft.identitymicrosoft.identitymodel.dll). In WIF 4.5, the WIF classes have been split across the following assemblies: **mscorlib**(mscorlib.dll), **System.IdentityModel** (System.IdentityModel.dll), **System.IdentityModel.Services** (System.IdentityModel.Services.dll), and **System.ServiceModel** (System.ServiceModel.dll).

The WIF 3.5 classes were all contained in one of the **Microsoft.IdentityModel** namespaces; for example, **Microsoft.IdentityModel**,**Microsoft.IdentityModel.Tokens**, **Microsoft.IdentityModel.Web**, and so on. In WIF 4.5, the WIF classes are now spread across the[System.IdentityModel](http://go.microsoft.com/fwlink/?LinkId=272004) namespaces, the [System.Security.Claims](https://msdn.microsoft.com/en-us/library/system.security.claims(v=vs.110).aspx) namespace, and the [System.ServiceModel.Security](https://msdn.microsoft.com/en-us/library/system.servicemodel.security(v=vs.110).aspx) namespace. In addition to this reorganization, some WIF 3.5 classes have been dropped in WIF 4.5.

The following table shows some of the more important WIF 4.5 namespaces and the kind of classes they contain. For more detailed information about how namespaces map between WIF 3.5 and WIF 4.5 and about namespaces and classes that have been dropped in WIF 4.5, see [Namespace Mapping between WIF 3.5 and WIF 4.5](https://msdn.microsoft.com/en-us/library/jj157091(v=vs.110).aspx).

|  |  |  |
| --- | --- | --- |
| **WIF 4.5 Namespace** | **Description** | |
| [System.IdentityModel](https://msdn.microsoft.com/en-us/library/system.identitymodel(v=vs.110).aspx) | Contains classes that represent cookie transforms, security token services, and specialized XML dictionary readers. Contains classes from the following WIF 3.5 namespaces:**Microsoft.IdentityModel**, **Microsoft.IdentityModel.SecurityTokenService**, and**Microsoft.IdentityModel.Threading**. | |
| [System.Security.Claims](https://msdn.microsoft.com/en-us/library/system.security.claims(v=vs.110).aspx) | Contains classes that represent claims, claims-based identities, claims based principals, and other claims based identity model artifacts. Contains classes from the **Microsoft.IdentityModel.Claims**namespace. | |
| [System.IdentityModel.Tokens](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens(v=vs.110).aspx) | Contains classes that represent security tokens, security token handlers, and other security token artifacts. Contains classes from the following WIF 3.5 namespaces:**Microsoft.IdentityModel.Tokens**, **Microsoft.IdentityModel.Tokens.Saml11**, and**Microsoft.IdentityModel.Tokens.Saml2**. | |
| [System.IdentityModel.Services](https://msdn.microsoft.com/en-us/library/system.identitymodel.services(v=vs.110).aspx) | Contains classes that are used in passive (WS-Federation) scenarios. Contains classes from the**Microsoft.IdentityModel.Web** namespace. | |
| [System.ServiceModel.Security](https://msdn.microsoft.com/en-us/library/system.servicemodel.security(v=vs.110).aspx) | Classes that represent WCF contracts, channels, service hosts and other artifacts that are used in active (WS-Trust) scenarios are now in this namespace. In WIF 3.5 , these classes were in the**Microsoft.IdentityModel.Protocols.WSTrust** namespace. | |
| **Important note Important** | |
| The following **System.IdentityModel** namespaces contain classes that implement the WCF claims-based identity model:[System.IdentityModel.Claims](https://msdn.microsoft.com/en-us/library/system.identitymodel.claims(v=vs.110).aspx), [System.IdentityModel.Policy](https://msdn.microsoft.com/en-us/library/system.identitymodel.policy(v=vs.110).aspx), and [System.IdentityModel.Selectors](https://msdn.microsoft.com/en-us/library/system.identitymodel.selectors(v=vs.110).aspx). The WCF claims-based identity model is superseded by WIF. You should not use classes in these three namespaces when building solutions based on WIF. | |

[**Changes Due to .NET Integration**](javascript:void(0))

WIF is now integrated into the .NET Framework. Most .NET identity and principal classes now derive from[System.Security.Claims.ClaimsIdentity](https://msdn.microsoft.com/en-us/library/system.security.claims.claimsidentity(v=vs.110).aspx) and [System.Security.Claims.ClaimsPrincipal](https://msdn.microsoft.com/en-us/library/system.security.claims.claimsprincipal(v=vs.110).aspx). The results in the following changes in WIF 4.5:

* WIF classes that represent claims, identities, and principals now exist in the [System.Security.Claims](https://msdn.microsoft.com/en-us/library/system.security.claims(v=vs.110).aspx) namespace.

|  |
| --- |
| **Important note Important** |
| The [System.IdentityModel.Claims](https://msdn.microsoft.com/en-us/library/system.identitymodel.claims(v=vs.110).aspx) namespace contains classes that represent artifacts in the WCF claims-based identity model. Many of these classes have names that are the same as WIF classes; for example, **Claims**. Do not use these classes when building solutions based on WIF. |

* .NET identity and principal classes now derive directly from [System.Security.Claims.ClaimsIdentity](https://msdn.microsoft.com/en-us/library/system.security.claims.claimsidentity(v=vs.110).aspx) and[System.Security.Claims.ClaimsPrincipal](https://msdn.microsoft.com/en-us/library/system.security.claims.claimsprincipal(v=vs.110).aspx), which represent claims-based identities and principals. For this reason, the WIF 3.5 interfaces **IClaimsIdentity** and **IClaimsPrincipal** are no longer needed and are not available in WIF 4.5.
* Because.NET identity and principal classes such as [System.Security.Principal.WindowsIdentity](https://msdn.microsoft.com/en-us/library/system.security.principal.windowsidentity(v=vs.110).aspx) and[System.Security.Principal.WindowsPrincipal](https://msdn.microsoft.com/en-us/library/system.security.principal.windowsprincipal(v=vs.110).aspx) now derive from [ClaimsIdentity](https://msdn.microsoft.com/en-us/library/system.security.claims.claimsidentity(v=vs.110).aspx) and [ClaimsPrincipal](https://msdn.microsoft.com/en-us/library/system.security.claims.claimsprincipal(v=vs.110).aspx), they have claims functionality built-in. For this reason, claims-specific identity and principal classes such as **WindowsClaimsIdentity** and**WindowsClaimsPrincipal** that were present in WIF 3.5 are no longer needed and do not exist in WIF 4.5.

[**Other Changes to WIF Functionality**](javascript:void(0))

In addition to the namespace changes and the changes due to integration with .NET, the following general changes to WIF functionality occur in WIF 4.5.

* The **Microsoft.IdentityModel.ExceptionMapper** class, which provided functionality that enabled you to map exceptions to specific SOAP Faults, is removed.
* The exception surface has been greatly reduced.
* Many of the classes that defined protocol or WS-\* specific constants have been removed; for example, the**Microsoft.IdentityModel.WSAddressing10Constants** class, which defined constants related to WS-Addressing 1.0.
* The Claims to Windows Token Service (c2wts) and its associated classes in the **Microsoft.IdentityModel.WindowsTokenService**namespace are removed.

[**WIF Configuration Changes**](javascript:void(0))

Many of the changes in the configuration file have been caused by namespace updates in WIF 4.5. For example, consider the following WIF 3.5 entry in the <httpModules> section to add the WS-Federation Authentication Manager to an application:

<add name="WSFederationAuthenticationModule" type="Microsoft.IdentityModel.Web.WSFederationAuthenticationModule, Microsoft.IdentityModel, Version=3.5.0.0, Culture=neutral, PublicKeyToken=abcd … 5678" />

This entry has been updated in WIF 4.5 to include the new namespaces and assembly version:

<add name="WSFederationAuthenticationModule" type="System.IdentityModel.Services.WSFederationAuthenticationModule, System.IdentityModel.Services, Version=4.0.0.0, Culture=neutral, PublicKeyToken=abcd … 5678"/>

The following list enumerates the major changes to the configuration file for WIF 4.5.

* The **<microsoft.identityModel>** section is now the [<system.identityModel>](https://msdn.microsoft.com/en-us/library/hh568638(v=vs.110).aspx) section.
* The **<service>** element is now the [<identityConfiguration>](https://msdn.microsoft.com/en-us/library/hh568637(v=vs.110).aspx) element.
* A new section, [<system.identityModel.services>](https://msdn.microsoft.com/en-us/library/hh568674(v=vs.110).aspx), has been added to specify settings that control behavior in passive (WS-Federation) scenarios.
* The [<federationConfiguration>](https://msdn.microsoft.com/en-us/library/hh568657(v=vs.110).aspx) element and its child elements have been moved from the **<service>** element in WIF 3.5 to the new **<system.identityModel.services>** element .
* Several elements that could be specified at the service-level directly under the **<service>** element in WIF 3.5 have been restricted to being specified under the [<securityTokenHandlerConfiguration>](https://msdn.microsoft.com/en-us/library/hh568639(v=vs.110).aspx) element. (They may still be specified under the[<identityConfiguration>](https://msdn.microsoft.com/en-us/library/hh568637(v=vs.110).aspx) element in WIF 4.5 for backward compatibility.)

For a complete list of the WIF 4.5 configuration elements, see [Windows Identity Foundation Configuration Schema](https://msdn.microsoft.com/en-us/library/hh598930(v=vs.110).aspx).

[**Visual Studio Tooling Changes**](javascript:void(0))

The WIF 3.5 SDK offered a stand-alone federation utility, FedUtil.exe (FedUtil), that you could use to outsource identity management in WIF-enabled applications to a security token service (STS). This tool added WIF settings to the application configuration file to enable the application to get security tokens from one or more STSs, and was surfaced in Visual Studio through the **Add STS Service Reference**button. FedUtil does not ship with WIF 4.5. Instead, WIF 4.5 supports a new Visual Studio extension named the Identity and Access Tool for Visual Studio 2012 that you can use to modify your application’s configuration file with the WIF settings required to outsource identity management to an STS. The Identity and Access Tool also implements an STS called Local STS that you can use to test your WIF-enabled applications. In many cases, this feature obviates the need to build custom STSs that were often necessary in WIF 3.5 to test solutions under development. For this reason, the STS templates are no longer supported in Visual Studio 2012; however, the classes that support the development of STSs are still available in WIF 4.5.

You can install the Identity and Access Tool from the Extensions and Updates Manager in Visual Studio or you can download it from the following page on Code Gallery: [Identity and Access Tool for Visual Studio 2012 on Code Gallery](http://go.microsoft.com/fwlink/?LinkID=245849). The Visual Studio tooling changes are summarized in the following list:

* The Add STS Service Reference functionality is removed. The replacement is the Identity and Access Tool.
* The Visual Studio STS templates are removed. You can use the Local STS that is available through the Identity and Access Tool to test identity solutions that you develop with WIF. The Local STS configuration can be modified to customize the claims that it serves.
* The stand-alone Federation Utility (FedUtil) is not available in WIF 4.5. You can use the Identity and Access Tool to modify your configuration files to outsource identity management to an STS.

For more information about the Identity and Access Tool, see [Identity and Access Tool for Visual Studio 2012](https://msdn.microsoft.com/en-us/library/hh545418(v=vs.110).aspx)

[**Passive (WS-Federation) Scenarios:**](javascript:void(0))

* Classes that support passive scenarios have been moved to the [System.IdentityModel.Services](https://msdn.microsoft.com/en-us/library/system.identitymodel.services(v=vs.110).aspx) namespace. In WIF 3.5 these classes were in the **Microsoft.IdentityModel.Web** namespace.
* The classes in the **Microsoft.IdentityModel.Web.Configuration** namespace have been moved to[System.IdentityModel.Services.Configuration](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.configuration(v=vs.110).aspx). These classes represent objects specific to configuration in passive scenarios.
* The **FederatedPasssiveSignInControl** is no longer supported; all classes in the **Microsoft.IdentityModel.Web.Controls**namespace have been removed from WIF 4.5.
* The WS-Federation Authentication Module ([WSFederationAuthenticationModule](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.wsfederationauthenticationmodule(v=vs.110).aspx)) sign-out functionality is different than WIF 3.5. For more details about how sign-out works in WIF 4.5, see the [WSFederationAuthenticationModule](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.wsfederationauthenticationmodule(v=vs.110).aspx) class topic.

[**Active (WCF/WS-Trust) Scenarios:**](javascript:void(0))

* The **Microsoft.IdentityModel.Protocols.WSTrust** namespace has been split mainly into two namespaces in WIF 4.5. Classes that represent artifacts that are specific to the WS-Trust protocol are now in [System.IdentityModel.Protocols.WSTrust](https://msdn.microsoft.com/en-us/library/system.identitymodel.protocols.wstrust(v=vs.110).aspx). This includes classes like [RequestSecurityToken](https://msdn.microsoft.com/en-us/library/system.identitymodel.protocols.wstrust.requestsecuritytoken(v=vs.110).aspx). Classes that represent service contracts, channels, service hosts and other artifacts that are involved in using WS-Trust in WCF applications have been moved to [System.ServiceModel.Security](https://msdn.microsoft.com/en-us/library/system.servicemodel.security(v=vs.110).aspx); for example, the[IWSTrust13AsyncContract](https://msdn.microsoft.com/en-us/library/system.servicemodel.security.iwstrust13asynccontract(v=vs.110).aspx) interface.
* Configuring a WCF application to use WIF has been greatly simplified. Previously the**Microsoft.IdentityModel.Configuration.ConfigureServiceHostBehaviorExtensionElement** had to be added as a behavior extension and this functionality was then used to wedge WIF into the service behavior by specifying a**<federatedServiceHostConfiguration>** element. WIF 4.5 has been more tightly integrated with WCF. Now you enable WIF on a WCF service by specifying the **useIdentityConfiguration** attribute on the**<system.serviceModel>**/**<behaviors>**/**<serviceBehaviors>**/**<serviceCredentials>** element as in the following XML:

Xml

<serviceCredentials useIdentityConfiguration="true">

<!--Certificate added by Identity And Access VS Package. Subject='CN=localhost', Issuer='CN=localhost'. Make sure you have this certificate installed. The Identity and Access tool does not install this certificate.-->

<serviceCertificate findValue="CN=localhost" storeLocation="LocalMachine" storeName="My" x509FindType="FindBySubjectDistinguishedName" />

</serviceCredentials>

* In WIF 4.5 the service certificate used by an active (WCF) service must be specified on the service host. In configuration, you can do this through the **<system.serviceModel>**/**<behaviors>**/**<serviceBehaviors>**/**<serviceCredentials>**/**<serviceCertificate>**element as shown in the preceding example. In WIF 3.5 the service certificate could be specified through the**<serviceCertificate>** child element of the WIF **<service>** element. This functionality does not exist in WIF 4.5; specify the**<serviceCertificate>** element under the **<serviceCredentials>** element instead.
* The classes used to wedge WIF 3.5 into WCF no longer exist. This includes the following classes in the**Microsoft.IdentityModel.Tokens** namespace: **FederatedSecurityTokenManager**, **FederatedServiceCredentials**, and**IdentityModelServiceAuthorizationManager**, as well as the**Microsoft.IdentityModel.Configuration.ConfigureServiceHostBehaviorExtensionElement** class.

[**Enabling WIF 3.5 in Windows 8**](javascript:void(0))

Because WIF 4.5 is part of .NET 4.5, it is automatically enabled on computers running Windows 8 and Windows Server 2012 and applications that are built using WIF 4.5 will run under Windows 8 or Windows Server 2012 by default. However, you may need to run applications that were built using WIF 3.5 on a computer that is running Windows 8 or Windows Server 2012. In this case, you need to enable WIF 3.5 on the target computer. On a computer running Windows 8, you can do this by using the Deployment Image Servicing and Management (DISM) tool. On a computer running Windows Server 2012, you can use the DISM tool or you can use the Windows PowerShell **Add-WindowsFeature** cmdlet. In both cases, the appropriate commands can be invoked either at the command line or from inside the Windows PowerShell environment.

The following commands show how to use the DISM tool from either the command line or from inside the Windows PowerShell environment. By default, the DISM PowerShell module is included in Windows 8 and Windows Server 2012 and does not need to be imported. For more information about using DISM with Windows PowerShell, see [How to Use DISM in Windows PowerShell](http://go.microsoft.com/fwlink/?LinkId=254419).

To enable WIF 3.5 using DISM:

dism /online /enable-feature:windows-identity-foundation

To disable WIF 3.5 using DISM:

dism /online /disable-feature:windows-identity-foundation

To check which features are enabled or disabled using DISM:

dism /online /get-features

Alternatively, on computers running Windows Server 2012, you can enable WIF 3.5 by using the Windows PowerShell **Add-WindowsFeature** cmdlet. To do so from the command line, you can enter the following command:

powershell "add-windowsfeature windows-identity-foundation"

From inside the Windows PowerShell environment, you can enter the command directly:

add-windowsfeature windows-identity-foundation

|  |
| --- |
| **NoteNote** |
| Because many of the classes in WIF 3.5 and WIF 4.5 share the same names, when you are using both WIF 3.5 and WIF 4.5 together, be sure to either use fully qualified class names or use namespace aliases to distinguish between classes in WIF 3.5 and WIF 4.5. |

### Namespace Mapping between WIF 3.5 and WIF 4.5

**.NET Framework 4.6 and 4.5**

Beginning with .NET 4.5, Windows Identity Foundation (WIF) has been fully integrated into the .NET Framework. This integration engendered name changes and some consolidation of the WIF namespaces and API surface. This topic provides some guidance and a general mapping between the WIF 3.5 namespaces and the WIF 4.5 namespaces. It is not intended to be exhaustive, but rather provide some general information about where to find familiar WIF 3.5 classes in WIF 4.5. For more detailed information about the differences between WIF 3.5 and WIF 4.5, see [What's New in Windows Identity Foundation 4.5](https://msdn.microsoft.com/en-us/library/hh873305(v=vs.110).aspx). For guidance about how to migrate an applications built using WIF 3.5 to WIF 4.5, see [Guidelines for Migrating an Application Built Using WIF 3.5 to WIF 4.5](https://msdn.microsoft.com/en-us/library/jj157089(v=vs.110).aspx).

[**WIF 3.5 to WIF 4.5 Namespace Map**](javascript:void(0))

The WIF classes, which were collected under the **Microsoft.IdentityModel** namespaces in WIF 3.5, are now distributed among the following namespaces: **System.Security.Claims**, **System.ServiceModel.Security**, and the **System.IdentityModel** namespaces in WIF 4.5. In addition some WIF 3.5 namespaces were consolidated or dropped entirely in WIF 4.5.

|  |
| --- |
| **Important note Important** |
| The following **System.IdentityModel** namespaces contain classes that implement the WCF claims-based identity model:[System.IdentityModel.Claims](https://msdn.microsoft.com/en-us/library/system.identitymodel.claims(v=vs.110).aspx), [System.IdentityModel.Policy](https://msdn.microsoft.com/en-us/library/system.identitymodel.policy(v=vs.110).aspx), and [System.IdentityModel.Selectors](https://msdn.microsoft.com/en-us/library/system.identitymodel.selectors(v=vs.110).aspx). The WCF claims-based identity model is superseded by WIF. You should not use classes in these three namespaces when building solutions based on WIF. |

The following table provides information about where WIF 3.5 classes can be found in WIF 4.5.

|  |  |  |
| --- | --- | --- |
| **WIF 3.5 Namespace** | **WIF 4.5 Namespace** | **Comments** |
| **Microsoft.IdentityModel** | [System.IdentityModel](https://msdn.microsoft.com/en-us/library/system.identitymodel(v=vs.110).aspx) | * Most of the classes that represent constants are not implemented. * The classes that are used to build security token services have been moved from**Microsoft.IdentityModel.SecurityTokenService** to[System.IdentityModel](https://msdn.microsoft.com/en-us/library/system.identitymodel(v=vs.110).aspx). * The classes in **Microsoft.IdentityModel.Threading** have been moved to [System.IdentityModel](https://msdn.microsoft.com/en-us/library/system.identitymodel(v=vs.110).aspx). * The **ExceptionMapper** and **MruSecurityTokenCache**classes are not implemented. |
| **Microsoft.IdentityModel.Claims** | [System.Security.Claims](https://msdn.microsoft.com/en-us/library/system.security.claims(v=vs.110).aspx) | * The **IClaimsPrincipal** and **IClaimsIdentity** interfaces are not implemented in WIF 4.5. Instead[System.Security.Claims.ClaimsPrincipal](https://msdn.microsoft.com/en-us/library/system.security.claims.claimsprincipal(v=vs.110).aspx) and[System.Security.Claims.ClaimsIdentity](https://msdn.microsoft.com/en-us/library/system.security.claims.claimsidentity(v=vs.110).aspx) are now the base classes from which most .NET principal and identity classes derive. This means there is no need for specialized claims principal and identity classes like**Microsoft.IdentityModel.Claims.WindowsClaimsPrincipal**and**Microsoft.IdentityModel.Claims.WindowsClaimsIdentity**in WIF 4.5, use [System.Security.Principal.WindowsPrincipal](https://msdn.microsoft.com/en-us/library/system.security.principal.windowsprincipal(v=vs.110).aspx)and [System.Security.Principal.WindowsIdentity](https://msdn.microsoft.com/en-us/library/system.security.principal.windowsidentity(v=vs.110).aspx) instead. The same is true for other for the other specialized claims principal and identity classes that existed in WIF 3.5. * The **Microsoft.IdentityModel.Claims.ClaimsCollection**class is not implemented in WIF 4.5. Instead, collections of claims are exposed as enumerable collections of type[System.Security.Claims.Claim](https://msdn.microsoft.com/en-us/library/system.security.claims.claim(v=vs.110).aspx). * [System.Security.Claims.ClaimsPrincipal](https://msdn.microsoft.com/en-us/library/system.security.claims.claimsprincipal(v=vs.110).aspx) and[System.Security.Claims.ClaimsIdentity](https://msdn.microsoft.com/en-us/library/system.security.claims.claimsidentity(v=vs.110).aspx) provide methods that now fully support LINQ. |
| **Microsoft.IdentityModel.Configuration** | [System.IdentityModel.Configuration](https://msdn.microsoft.com/en-us/library/system.identitymodel.configuration(v=vs.110).aspx) | Some elements and classes have undergone name changes and some have been dropped in WIF 4.5; for example**Microsoft.IdentityModel.Configuraiton.ServiceConfiguration** is now [System.IdentityModel.Configuration.IdentityConfiguration](https://msdn.microsoft.com/en-us/library/system.identitymodel.configuration.identityconfiguration(v=vs.110).aspx). |
| **Microsoft.IdentityModel.Protocols** | [System.IdentityModel.Services](https://msdn.microsoft.com/en-us/library/system.identitymodel.services(v=vs.110).aspx) | - |
| **Microsoft.IdentityModel.Protocols.WSFederation** | [System.IdentityModel.Services](https://msdn.microsoft.com/en-us/library/system.identitymodel.services(v=vs.110).aspx) | - |
| **Microsoft.IdentityModel.Protocols.WSFederation.Metadata** | [System.IdentityModel.Metadata](https://msdn.microsoft.com/en-us/library/system.identitymodel.metadata(v=vs.110).aspx) | - |
| **Microsoft.IdentityModel.Protocols.WSIdentity** | Not Implemented in WIF 4.5 | In WIF 3.5 contained classes to support CardSpace, not implemented in WIF 4.5. |
| **Microsoft.IdentityModel.Protocols.WSTrust** | Split between the[System.IdentityModel.Protocols.WSTrust](https://msdn.microsoft.com/en-us/library/system.identitymodel.protocols.wstrust(v=vs.110).aspx)and [System.ServiceModel.Security](https://msdn.microsoft.com/en-us/library/system.servicemodel.security(v=vs.110).aspx)namespaces. | Classes that represents WS-Trust artifacts are in the[System.IdentityModel.Protocols.WSTrust](https://msdn.microsoft.com/en-us/library/system.identitymodel.protocols.wstrust(v=vs.110).aspx) namespace; for example, the [RequestSecurityToken](https://msdn.microsoft.com/en-us/library/system.identitymodel.protocols.wstrust.requestsecuritytoken(v=vs.110).aspx) class. Classes that represent WCF service contracts, service hosts, and channels that enable a WCF service to communicate using the WS-Trust protocol are in the[System.ServiceModel.Security](https://msdn.microsoft.com/en-us/library/system.servicemodel.security(v=vs.110).aspx) namespace; for example, the[WSTrustServiceHost](https://msdn.microsoft.com/en-us/library/system.servicemodel.security.wstrustservicehost(v=vs.110).aspx) class. |
| **Microsoft.IdentityModel.Protocols.WSTrust.Bindings** | Not Implemented in WIF 4.5 | - |
| **Microsoft.IdentityModel.Protocols.XmlEncryption** | Not Implemented in WIF 4.5 | Contained classes that represent XML Encryption constants in WIF 3.5. These constants are not implemented in WIF 4.5. |
| **Microsoft.IdentityModel.Protocols.XmlSignature** | [System.IdentityModel](https://msdn.microsoft.com/en-us/library/system.identitymodel(v=vs.110).aspx) | The **EnvelopingSignature** class and classes that represent constants are not implemented. |
| **Microsoft.IdentityModel.SecurityTokenService** | Split between the [System.IdentityModel](https://msdn.microsoft.com/en-us/library/system.identitymodel(v=vs.110).aspx),[System.IdentityModel.Protocols.WSTrust](https://msdn.microsoft.com/en-us/library/system.identitymodel.protocols.wstrust(v=vs.110).aspx), and [System.IdentityModel.Tokens](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens(v=vs.110).aspx)namespaces. | - |
| **Microsoft.IdentityModel.Threading** | [System.IdentityModel](https://msdn.microsoft.com/en-us/library/system.identitymodel(v=vs.110).aspx) | - |
| **Microsoft.IdentityModel.Tokens** | [System.IdentityModel.Tokens](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens(v=vs.110).aspx) | - |
| **Microsoft.IdentityModel.Tokens.Saml11** | [System.IdentityModel.Tokens](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens(v=vs.110).aspx) | - |
| **Microsoft.IdentityModel.Tokens.Saml2** | [System.IdentityModel.Tokens](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens(v=vs.110).aspx) | - |
| **Microsoft.IdentityModel.Web** | [System.IdentityModel.Services](https://msdn.microsoft.com/en-us/library/system.identitymodel.services(v=vs.110).aspx) | - |
| **Microsoft.IdentityModel.Web.Configuration** | [System.IdentityModel.Services.Configuration](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.configuration(v=vs.110).aspx) | Classes that provide configuration for passive (WS-Federation) scenarios have largely been moved to[System.IdentityModel.Services.Configuration](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.configuration(v=vs.110).aspx); however, some of these classes are in [System.IdentityModel.Services](https://msdn.microsoft.com/en-us/library/system.identitymodel.services(v=vs.110).aspx). |
| **Microsoft.IdentityModel.Web.Controls** | Not Implemented in WIF 4.5 | The classes in **Microsoft.IdentityModel.Web.Controls**implemented the Federated Passive Sign-In Control, which does not exist in WIF 4.5. |
| **Microsoft.IdentityModel.WindowsTokenService** | Not Implemented in WIF 4.5 | - |

## WIF Code Sample Index

**.NET Framework 4.6 and 4.5**

The following are code samples for Windows Identity Foundation 4.5:

* [ClaimsAwareWebApp](http://go.microsoft.com/fwlink/?LinkID=248405) - this sample demonstrates basic use of authentication externalization (to the local test Security Token Service from the Identity and Access Tool for Visual Studio 11) on a classic ASP.NET application (as opposed to a web site).
* [ClaimsAwareWebService](http://go.microsoft.com/fwlink/?LinkID=248406) - this sample demonstrates basic use of authentication externalization on a classic WCF service.
* [ClaimsAwareMvcApplication](http://go.microsoft.com/fwlink/?LinkID=248407) - this sample demonstrates how to integrate WIF with MVC, including non-blanket protection and code which honors the forms authentication redirects out of the LogOn controller.
* [ClaimsAwareWebFarm](http://go.microsoft.com/fwlink/?LinkID=248408) - this sample demonstrates a farm ready session cache (as opposed to a tokenreplycache) so that you can use sessions by reference instead of exchanging big cookies. It also demonstrates an easier way of securing cookies in a farm.
* [ClaimsAwareFormsAuthentication](http://go.microsoft.com/fwlink/?LinkID=248409) - this very simple sample demonstrates that in .NET 4.5 you get claims in your principals regardless of how you authenticate your users.
* [ClaimsBasedAuthorization](http://go.microsoft.com/fwlink/?LinkID=248410)- this samples shows how to use your CLaimsAuthorizationManager class and the ClaimsAuthorizationModule for applying your own authorization policies.
* [FederationMetadata](http://go.microsoft.com/fwlink/?LinkID=248411) – this sample demonstrates both dynamic generation (on a custom STS) and dynamic consumption (on a relying party application) of metadata documents.
* [CustomToken](http://go.microsoft.com/fwlink/?LinkID=248412) – this sample demonstrates how to build a custom Simple Web Token (SWT) token type.

## WIF Extensions

**.NET Framework 4.6 and 4.5**

This section describes the extensions for Windows Identity Foundation.

* [JSON Web Token Handler](https://msdn.microsoft.com/en-us/library/dn205065(v=vs.110).aspx)
* [Validating Issuer Name Registry](https://msdn.microsoft.com/en-us/library/dn205067(v=vs.110).aspx)

### JSON Web Token Handler

**.NET Framework 4.6 and 4.5**

The JSON Web Token Handler extension for Windows Identity Foundation enables you to create and validate JSON Web Tokens (JWT) in your applications. The JWT Token Handler can be configured to run in the WIF pipeline like other built-in security token handlers, but it can also be used independently to perform token validation in lightweight applications. The JWT Token Handler is particularly useful when using an OAuth 2.0 bearer token scheme, such as authenticating to Windows Azure Active Directory.

The JWT Token Handler is available as a NuGet package. See [Downloading the JSON Web Token Handler Package](https://msdn.microsoft.com/en-us/library/dn205064(v=vs.110).aspx) for more information.

[**Scenarios**](javascript:void(0))

The JWT Token Handler enables the following key scenarios:

* **Validate a JWT Token in a Server Application**: In this scenario, a company named Litware has a server application that uses WIF to handle web sign-on requests. Litware wants to enable their application to use JWT tokens for authentication. The application is updated with the JWT Token Handler, and then the application configuration is updated to add the JWT Token Handler in the WIF pipeline. After the updates have been made and a new request enters the WIF pipeline, the JWT token is validated using the new handler and successful authentication occurs.
* **Validate a JWT Token in a REST Web Service**: In this scenario, a company named Litware has a REST web service that is secured by Windows Azure Active Directory. Requests to the web service must be authenticated by Windows Azure AD, which issues a JWT token upon successful authentication. Litware has a client application that needs to access the web service. The client makes a request to the web service and presents its JWT token from Windows Azure AD, which is then validated by the web service using the JWT Token Handler. After the JWT Token Handler has validated the token, the desired resource is returned to the client by the web service.

[**Features**](javascript:void(0))

The JWT Token Handler offers the following features:

* **Validate a JWT Token**: JWT tokens can be easily validated by the token handler’s validation logic, either as a part of the application’s WIF pipeline or called independently of WIF
* **Create a JWT Token**: The JWT Token Handler can be used to create JWT tokens for authorization in downstream services

#### Downloading the JSON Web Token Handler Package

**.NET Framework 4.6 and 4.5**

This topic discusses how to download and use the JSON Web Token Handler in your project.

[**Downloading the JSON Web Token Handler**](javascript:void(0))

The JSON Web Token Handler extension is available as a NuGet package, which adds the necessary assemblies and references to your project. If you do not already have NuGet installed, go to [nuget.org](http://nuget.org/) to install it. You can see the versioning history for the extension by visiting its page on NuGet: [JSON Web Token Handler on NuGet](http://www.nuget.org/packages/System.IdentityModel.Tokens.Jwt/)

Downloading the JSON Web Token Handler by using the Package Manager GUI

1. In Visual Studio, right-click your project in **Solution Explorer**, and then select **Manage NuGet Packages**.
2. In the **Manage NuGet Packages** window, click the search box and enter JWT Token Handler and press **Enter**.
3. From the results pane, click the **Install** button for the first result.
4. The package will begin downloading. Before it is added to your project, the License Acceptance dialog will appear. If you agree to the license terms, click **I Accept**.
5. The latest JSON Web Token Handler assemblies will be downloaded and added to your project.

Downloading the JSON Web Token Handler by using the Package Manager Console

1. In Visual Studio, click **Tools**, **Library Package Manager**, and then **Package Manager Console**.
2. The **Package Manager Console** appears. Enter the following text and press **Enter**:
3. Install-Package System.IdentityModel.Tokens.Jwt
4. The latest JSON Web Token Handler assemblies will be downloaded and added to your project.

#### JSON Web Token Handler API

This section contains the API Reference for the JSON Web Token Handler WIF Extension.

##### System.IdentityModel.Tokens Namespace

[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635736413806036745

The System.IdentityModel.Tokens namespace contains classes that represent security tokens, security token handlers, key identifier clauses and other artifacts used in token generation and processing. The namespace contains base classes such as [SecurityToken](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.securitytoken(v=vs.114).aspx),[SecurityTokenHandler](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.securitytokenhandler(v=vs.114).aspx), and [SecurityKeyIdentifierClause](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.securitykeyidentifierclause(v=vs.114).aspx), as well as classes that derive from these classes and represent several of the token types, artifacts, and handlers for which the Windows Identity Foundation (WIF) has built in support. This includes classes that contain support for SAML v1.1 and v2.0 tokens, such as: [SamlSecurityToken](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.samlsecuritytoken(v=vs.114).aspx), [SamlSecurityTokenHandler](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.samlsecuritytokenhandler(v=vs.114).aspx), [Saml2SecurityToken](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.saml2securitytoken(v=vs.114).aspx), and [Saml2SecurityTokenHandler](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.saml2securitytokenhandler(v=vs.114).aspx).

[**Classes**](javascript:void(0))

|  |  |  |
| --- | --- | --- |
|  | **Class** | **Description** |
| Public class | [AsymmetricSignatureProvider](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.asymmetricsignatureprovider(v=vs.114).aspx) | A class that provides signing and verifying operations when working with an[AsymmetricSecurityKey](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.asymmetricsecuritykey(v=vs.114).aspx). |
| Public class | [JwtHeader](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.jwtheader(v=vs.114).aspx) | The [JwtHeader](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.jwtheader(v=vs.114).aspx) class contains JSON objects that represent the cryptographic operations applied to the JSON Web Token (JWT) and optionally any additional properties of the JWT. |
| Public class | [JwtPayload](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.jwtpayload(v=vs.114).aspx) | The [JwtPayload](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.jwtpayload(v=vs.114).aspx) class contains JSON objects representing the claims contained in the JWT. Each claim is a JSON object in the form of { Name, Value }. |
| Public class | [JwtSecurityToken](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.jwtsecuritytoken(v=vs.114).aspx) | A class that represents a JSON Web Token (JWT). |
| Public class | [JwtSecurityTokenHandler](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.jwtsecuritytokenhandler(v=vs.114).aspx) | A [SecurityTokenHandler](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.securitytokenhandler(v=vs.114).aspx) designed for creating and validating JSON Web Tokens (JWT). See <http://tools.ietf.org/html/draft-ietf-oauth-json-web-token-07> for more information about the JWT specification. |
| Public class | [JwtSecurityTokenRequirement](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.jwtsecuritytokenrequirement(v=vs.114).aspx) | Provides a location for settings that control how the [JwtSecurityTokenHandler](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.jwtsecuritytokenhandler(v=vs.114).aspx)validates or creates a [JwtSecurityToken](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.jwtsecuritytoken(v=vs.114).aspx). |
| Public class | [NamedKeyIssuerTokenResolver](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.namedkeyissuertokenresolver(v=vs.114).aspx) | [NamedKeyIssuerTokenResolver](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.namedkeyissuertokenresolver(v=vs.114).aspx) represents a collection of named sets of[SecurityKey](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.securitykey(v=vs.114).aspx)(s) that can be matched by a [NamedKeySecurityKeyIdentifierClause](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.namedkeysecuritykeyidentifierclause(v=vs.114).aspx)and return a [NamedKeySecurityToken](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.namedkeysecuritytoken(v=vs.114).aspx) that contains [SecurityKey](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.securitykey(v=vs.114).aspx)(s). |
| Public class | [NamedKeySecurityKeyIdentifierClause](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.namedkeysecuritykeyidentifierclause(v=vs.114).aspx) | A [SecurityKeyIdentifierClause](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.securitykeyidentifierclause(v=vs.114).aspx) that can be used to match[NamedKeySecurityToken](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.namedkeysecuritytoken(v=vs.114).aspx). |
| Public class | [NamedKeySecurityToken](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.namedkeysecuritytoken(v=vs.114).aspx) | A [SecurityToken](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.securitytoken(v=vs.114).aspx) that contains multiple [SecurityKey](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.securitykey(v=vs.114).aspx) that have a name. |
| Public class | [SignatureProvider](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.signatureprovider(v=vs.114).aspx) | This class defines the object model for types that provide signature services. |
| Public class | [SignatureProviderFactory](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.signatureproviderfactory(v=vs.114).aspx) | Creates [SignatureProvider](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.signatureprovider(v=vs.114).aspx)s by specifying a [SecurityKey](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.securitykey(v=vs.114).aspx) and algorithm.Supports both [AsymmetricSecurityKey](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.asymmetricsecuritykey(v=vs.114).aspx) and [SymmetricSecurityKey](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.symmetricsecuritykey(v=vs.114).aspx). |
| Public class | [SymmetricSignatureProvider](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.symmetricsignatureprovider(v=vs.114).aspx) | Provides signing and verifying operations using a [SymmetricSecurityKey](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.symmetricsecuritykey(v=vs.114).aspx) and specifying an algorithm. |
| Public class | [TokenValidationParameters](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.tokenvalidationparameters(v=vs.114).aspx) | Contains a set of parameters that are used by [SecurityTokenHandler](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.securitytokenhandler(v=vs.114).aspx) when validating a [SecurityToken](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.securitytoken(v=vs.114).aspx). |

### Validating Issuer Name Registry

**.NET Framework 4.6 and 4.5**

The Validating Issuer Name Registry (VINR) for Windows Identity Foundation enables multi-tenant applications to ensure that an incoming token has been issued by a trusted tenant and identity provider. This functionality is particularly useful for multi-tenant applications that use Windows Azure Active Directory because all tokens issued by Windows Azure AD are signed using the same certificate. In order to differentiate between requests from multiple tenants that use the same certificate – and consequently have the same thumbprint – your application must persist the issuer name for each tenant to perform validation logic. The VINR provides this functionality, and it also enables you to add custom validation logic or to store the issuer registry data in locations other than a configuration file. The extension can be added to your application’s WIF pipeline or it can be used independently.

The VINR is available as a NuGet package. See [Downloading the Validating Issuer Name Registry Package](https://msdn.microsoft.com/en-us/library/dn205069(v=vs.110).aspx) for more information.

[**Scenarios**](javascript:void(0))

The VINR enables the following key scenario:

* **Validate a Token in a Multi-Tenant Application**: In this scenario, a company named Litware has developed a multi-tenant application that uses an identity provider such as Windows Azure AD. This application serves two customers: Contoso and Fabrikam. When a user from Fabrikam authenticates to Litware’s application, the resulting token from Windows Azure AD is signed using its standard certificate and the request is issued by Fabrikam. The application needs to verify that both the issuer name and the token is valid, and needs to differentiate requests from Contoso that are signed using the same certificate from Windows Azure AD. The VINR makes it easy for Litware’s application to differentiate and validate requests from multiple tenants such as Contoso and Fabrikam.

[**Features**](javascript:void(0))

The VINR offers the following features:

* **Issuer Name and Token Validation for Multi-Tenant Applications**: Validates the incoming token by verifying the issuer name (tenant) and whether the token was signed using a valid certificate from the identity provider.
* **Extensibility for Custom Validation Logic and Data Stores**: Provides extensibility to inject your own validation logic and to specify a data store other than the default configuration file.

#### Downloading the Validating Issuer Name Registry Package

**.NET Framework 4.6 and 4.5**

This topic discusses how to download and use the Validating Issuer Name Registry (VINR) in your project.

[**Downloading the Validating Issuer Name Registry**](javascript:void(0))

The VINR is available as a NuGet package, which adds the necessary assemblies and references to your project. If you do not already have NuGet installed, go to [nuget.org](http://nuget.org/) to install it. You can see the versioning history for the extension by visiting its page on NuGet: [Microsoft Validating Issuer Name Registry on NuGet](https://nuget.org/packages/System.IdentityModel.Tokens.ValidatingIssuerNameRegistry/)

Downloading the Validating Issuer Name Registry by using the Package Manager GUI

1. In Visual Studio, right-click your project in **Solution Explorer**, and then select **Manage NuGet Packages**.
2. In the **Manage NuGet Packages** window, click the search box and enter ValidatingIssuerNameRegistry and press **Enter**.
3. From the results pane, click the **Install** button for the first result.
4. The package will begin downloading. Before it is added to your project, the License Acceptance dialog will appear. If you agree to the license terms, click **I Accept**.
5. The latest VINR assemblies will be downloaded and added to your project.

Downloading the Validating Issuer Name Registry by using the Package Manager Console

1. In Visual Studio, click **Tools**, **Library Package Manager**, and then **Package Manager Console**.
2. The **Package Manager Console** appears. Enter the following text and press **Enter**:
3. Install-Package System.IdentityModel.Tokens.ValidatingIssuerNameRegistry
4. The latest VINR assemblies will be downloaded and added to your project.

#### Validating Issuer Name Registry API

This section contains the API Reference for the Validating Issuer Name Registry WIF Extension.

## WIF API Reference

**.NET Framework 4.6 and 4.5**

Windows Identity Foundation (WIF) classes are split across the following assemblies: **mscorlib** (mscorlib.dll), **System.IdentityModel**(System.IdentityModel.dll), **System.IdentityModel.Services** (System.IdentityModel.Services.dll), and **System.ServiceModel**(System.ServiceModel.dll). This topic provides links to the WIF namespaces and brief explanations of the classes that each namespace contains.

|  |
| --- |
| **Important note Important** |
| The following **System.IdentityModel** namespaces contain classes that implement the WCF claims-based identity model:[System.IdentityModel.Claims](https://msdn.microsoft.com/en-us/library/system.identitymodel.claims(v=vs.110).aspx), [System.IdentityModel.Policy](https://msdn.microsoft.com/en-us/library/system.identitymodel.policy(v=vs.110).aspx), and [System.IdentityModel.Selectors](https://msdn.microsoft.com/en-us/library/system.identitymodel.selectors(v=vs.110).aspx). Starting with .NET Framework 4.5, the WCF claims-based identity model is superseded by WIF. You should not use classes in these three namespaces when building solutions based on WIF. |

[System.IdentityModel](https://msdn.microsoft.com/en-us/library/system.identitymodel(v=vs.110).aspx)

Contains classes that represent cookie transforms, security token services, and specialized XML dictionary readers.

[System.IdentityModel.Configuration](https://msdn.microsoft.com/en-us/library/system.identitymodel.configuration(v=vs.110).aspx)

Contains classes that provide configuration for applications and services built using the Windows Identity Foundation (WIF). The classes in this namespace represent settings under the [<identityConfiguration>](https://msdn.microsoft.com/en-us/library/hh568637(v=vs.110).aspx) element.

[System.IdentityModel.Metadata](https://msdn.microsoft.com/en-us/library/system.identitymodel.metadata(v=vs.110).aspx)

Contains classes that represent elements in a Federation Metadata document.

[System.IdentityModel.Protocols.WSTrust](https://msdn.microsoft.com/en-us/library/system.identitymodel.protocols.wstrust(v=vs.110).aspx)

Contains classes that represent WS-Trust artifacts.

[System.IdentityModel.Services](https://msdn.microsoft.com/en-us/library/system.identitymodel.services(v=vs.110).aspx)

Contains classes that are used in passive (WS-Federation) scenarios. Also contains some classes that represent settings under the[<system.identityModel.services>](https://msdn.microsoft.com/en-us/library/hh568674(v=vs.110).aspx) element. Settings under this element configure WS-Federation for applications. The**System.IdentityModel.Services.Configuration** namespace contains most of the classes that are used to configure WS-Federation.

[System.IdentityModel.Services.Configuration](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.configuration(v=vs.110).aspx)

Contains classes that provide configuration for WIF applications that use the WS-Federation protocol. The classes in this namespace represent settings under the [<system.identityModel.services>](https://msdn.microsoft.com/en-us/library/hh568674(v=vs.110).aspx) element. The **System.IdentityModel.Services** namespace also contains some classes that are used to configure WS-Federation.

[System.IdentityModel.Services.Tokens](https://msdn.microsoft.com/en-us/library/system.identitymodel.services.tokens(v=vs.110).aspx)

Contains specialized security token handlers for Web farm scenarios.

[System.IdentityModel.Tokens](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens(v=vs.110).aspx)

Contains classes that represent security tokens, security token handlers, and other security token artifacts.

[System.Security.Claims](https://msdn.microsoft.com/en-us/library/system.security.claims(v=vs.110).aspx)

Contains classes that represent claims, claims-based identities, claims-based principals, and other claims-based identity model artifacts.

[System.ServiceModel.Security](https://msdn.microsoft.com/en-us/library/system.servicemodel.security(v=vs.110).aspx)

Contains classes that represent WCF contracts, channels, service hosts and other artifacts that are used in active (WS-Trust) scenarios. This namespace also contains classes that are specific to Windows Communication Foundation (WCF) and that are not used by WIF.

## WIF Configuration Reference

**.NET Framework 4.6 and 4.5**

You can configure Windows Identity Foundation (WIF) in your applications by adding elements to a configuration file. This topic contains links to reference topics for the WIF configuration elements.

[Windows Identity Foundation Configuration Schema](https://msdn.microsoft.com/en-us/library/hh598930(v=vs.110).aspx)

The reference for the WIF configuration elements.

[WIF Configuration Schema Conventions](https://msdn.microsoft.com/en-us/library/hh598928(v=vs.110).aspx)

Contains information about general attributes and formats used by the WIF configuration elements.

### WIF Configuration Schema Conventions

**.NET Framework 4.6 and 4.5**

This topic discusses conventions used throughout the Windows Identity Foundation (WIF) configuration topics and describes some common features and attributes used in the [<system.identityModel>](https://msdn.microsoft.com/en-us/library/hh568638(v=vs.110).aspx) and the [<system.identityModel.services>](https://msdn.microsoft.com/en-us/library/hh568674(v=vs.110).aspx) sections.

[**Modes**](javascript:void(0))

Many of the WIF configuration elements have a **mode** attribute. This attribute typically controls which class is used to do a particular part of the processing and which configuration elements are allowed as child elements of the current element. A configuration error will be raised if elements that are included in the configuration file are ignored because of the mode settings.

[**Timespan Values**](javascript:void(0))

Where [TimeSpan](https://msdn.microsoft.com/en-us/library/system.timespan(v=vs.110).aspx) is used as the type of an attribute, see the [Parse(String)](https://msdn.microsoft.com/en-us/library/se73z7b9(v=vs.110).aspx) method to see the allowed format. This format conforms to the following specification.

[ws][-]{ d | [d.]hh:mm[:ss[.ff]] }[ws]

For example, "30", "30.00:00", "30.00:00:00" all mean 30 days; and "00:05", "00:05:00", "0.00:05:00.00" all mean 5 minutes.

[**Certificate References**](javascript:void(0))

Several elements take references to certificates using the **<certificateReference>** element. When referencing a certificate, the following attributes are available.

|  |  |
| --- | --- |
| **storeLocation** | A value of the [StoreLocation](https://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.storelocation(v=vs.110).aspx) enumeration: **CurrentUser** or **CurrentMachine**. |
| **storeName** | A value of the [StoreName](https://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.storename(v=vs.110).aspx) enumeration; the most useful for this context are **My** and **TrustedPeople**. |
| **x509FindType** | A value of the [X509FindType](https://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.x509findtype(v=vs.110).aspx) enumeration; the most useful for this context are **FindBySubjectName** and**FindByThumbprint**. To eliminate the chance of error, it is recommended that the **FindByThumbprint** type be used in production environments. |
| **findValue** | The value used to find the certificate, based on the **x509FindType** attribute. To eliminate the chance of error, it is recommended that the **FindByThumbprint** type be used in production environments. When **FindByThumbPrint** is specified, this attribute takes a value that is the hexadecimal-string form of the certificate thumbprint; for example, "97249e1a5fa6bee5e515b82111ef524a4c91583f". |

[**Custom Type References**](javascript:void(0))

Several elements reference custom types using the **type** attribute. This attribute should specify the name of the custom type. To reference a type from the Global Assembly Cache (GAC), a strong name should be used. To reference a type from an assembly in the Bin/ directory, a simple assembly-qualified reference may be used. Types defined in the App\_Code/ directory may also be referenced by simply specifying the type name with no qualifying assembly.

Custom types must be derived from the type specified and they must provide a **public** default (0 argument) constructor.

# Classes

## PrincipalPermission Class

<https://msdn.microsoft.com/en-us/library/system.security.permissions.principalpermission(v=vs.110).aspx>

## IPrincipal Interface

[**https://msdn.microsoft.com/en-us/library/system.security.principal.iprincipal(v=vs.71).aspx**](https://msdn.microsoft.com/en-us/library/system.security.principal.iprincipal(v=vs.71).aspx)

## IIdentity Interface

**https://msdn.microsoft.com/en-us/library/system.security.principal.iidentity(v=vs.71).aspx**