Application domains - .NET Framework

By gewarren

Archived: 2026-04-06 00:40:44 UTC

Note

This article is specific to .NET Framework. It doesn't apply to newer implementations of .NET, including .NET 6 and later versions.

Operating systems and runtime environments typically provide some form of isolation between applications. For example, Windows uses processes to isolate applications. This isolation is necessary to ensure that code running in one application cannot adversely affect other, unrelated applications.

Application domains provide an isolation boundary for security, reliability, and versioning, and for unloading assemblies. Application domains are typically created by runtime hosts, which are responsible for bootstrapping the common language runtime before an application is run.

Historically, process boundaries have been used to isolate applications running on the same computer. Each application is loaded into a separate process, which isolates the application from other applications running on the same computer.

The applications are isolated because memory addresses are process-relative; a memory pointer passed from one process to another cannot be used in any meaningful way in the target process. In addition, you cannot make direct calls between two processes. Instead, you must use proxies, which provide a level of indirection.

Managed code must be passed through a verification process before it can be run (unless the administrator has granted permission to skip the verification). The verification process determines whether the code can attempt to access invalid memory addresses or perform some other action that could cause the process in which it is running to fail to operate properly. Code that passes the verification test is said to be type-safe. The ability to verify code as type-safe enables the common language runtime to provide as great a level of isolation as the process boundary, at a much lower performance cost.

Application domains provide a more secure and versatile unit of processing that the common language runtime can use to provide isolation between applications. You can run several application domains in a single process with the same level of isolation that would exist in separate processes, but without incurring the additional overhead of making cross-process calls or switching between processes. The ability to run multiple applications within a single process dramatically increases server scalability.

Isolating applications is also important for application security. For example, you can run controls from several Web applications in a single browser process in such a way that the controls cannot access each other's data and resources.

The isolation provided by application domains has the following benefits:

- Faults in one application cannot affect other applications. Because type-safe code cannot cause memory faults, using application domains ensures that code running in one domain cannot affect other applications in the process.

- Individual applications can be stopped without stopping the entire process. Using application domains enables you to unload the code running in a single application.

  Note

  You cannot unload individual assemblies or types. Only a complete domain can be unloaded.

- Code running in one application cannot directly access code or resources from another application. The common language runtime enforces this isolation by preventing direct calls between objects in different application domains. Objects that pass between domains are either copied or accessed by proxy. If the object is copied, the call to the object is local. That is, both the caller and the object being referenced are in the same application domain. If the object is accessed through a proxy, the call to the object is remote. In this case, the caller and the object being referenced are in different application domains. Cross-domain calls use the same remote call infrastructure as calls between two processes or between two machines. As such, the metadata for the object being referenced must be available to both application domains to allow the method call to be JIT-compiled properly. If the calling domain does not have access to the metadata for the object being called, the compilation might fail with an exception of type FileNotFoundException. For more information, see Remote Objects. The mechanism for determining how objects can be accessed across domains is determined by the object. For more information, see System.MarshalByRefObject.

- The behavior of code is scoped by the application in which it runs. In other words, the application domain provides configuration settings such as application version policies, the location of any remote assemblies it accesses, and information about where to locate assemblies that are loaded into the domain.

- Permissions granted to code can be controlled by the application domain in which the code is running.

This section describes the relationship between application domains and assemblies. You must load an assembly into an application domain before you can execute the code it contains. Running a typical application causes several assemblies to be loaded into an application domain.

The way an assembly is loaded determines whether its just-in-time (JIT) compiled code can be shared by multiple application domains in the process, and whether the assembly can be unloaded from the process.

- If an assembly is loaded domain-neutral, all application domains that share the same security grant set can share the same JIT-compiled code, which reduces the memory required by the application. However, the assembly can never be unloaded from the process.

- If an assembly is not loaded domain-neutral, it must be JIT-compiled in every application domain in which it is loaded. However, the assembly can be unloaded from the process by unloading all the application domains in which it is loaded.

The runtime host determines whether to load assemblies as domain-neutral when it loads the runtime into a process. For managed applications, apply the LoaderOptimizationAttribute attribute to the entry-point method for the process, and specify a value from the associated LoaderOptimization enumeration. For unmanaged applications that host the common language runtime, specify the appropriate flag when you call the CorBindToRuntimeEx Function method.

There are three options for loading domain-neutral assemblies:

- LoaderOptimization.SingleDomain loads no assemblies as domain-neutral, except Mscorlib, which is always loaded domain-neutral. This setting is called single domain because it is commonly used when the host is running only a single application in the process.
- LoaderOptimization.MultiDomain loads all assemblies as domain-neutral. Use this setting when there are multiple application domains in the process, all of which run the same code.
- LoaderOptimization.MultiDomainHost loads strong-named assemblies as domain-neutral, if they and all their dependencies have been installed in the global assembly cache. Other assemblies are loaded and JIT-compiled separately for each application domain in which they are loaded, and thus can be unloaded from the process. Use this setting when running more than one application in the same process, or if you have a mixture of assemblies that are shared by many application domains and assemblies that need to be unloaded from the process.

JIT-compiled code cannot be shared for assemblies loaded into the load-from context, using the LoadFrom method of the Assembly class, or loaded from images using overloads of the Load method that specify byte arrays.

Assemblies that have been compiled to native code by using the Ngen.exe (Native Image Generator) can be shared between application domains, if they are loaded domain-neutral the first time they are loaded into a process.

JIT-compiled code for the assembly that contains the application entry point is shared only if all its dependencies can be shared.

A domain-neutral assembly can be JIT-compiled more than once. For example, when the security grant sets of two application domains are different, they cannot share the same JIT-compiled code. However, each copy of the JIT-compiled assembly can be shared with other application domains that have the same grant set.

When you decide whether to load assemblies as domain-neutral, you must make a tradeoff between reducing memory use and other performance factors.

- Access to static data and methods is slower for domain-neutral assemblies because of the need to isolate assemblies. Each application domain that accesses the assembly must have a separate copy of the static data, to prevent references to objects in static fields from crossing domain boundaries. As a result, the runtime contains additional logic to direct a caller to the appropriate copy of the static data or method. This extra logic slows down the call.
- All the dependencies of an assembly must be located and loaded when the assembly is loaded domain-neutral, because a dependency that cannot be loaded domain-neutral prevents the assembly from being

loaded domain-neutral.

An application domain forms an isolation boundary for security, versioning, reliability, and unloading of managed code. A thread is the operating system construct used by the common language runtime to execute code. At runtime, all managed code is loaded into an application domain and is run by one or more managed threads.

There is not a one-to-one correlation between application domains and threads. Several threads can execute in a single application domain at any given time, and a particular thread is not confined to a single application domain. That is, threads are free to cross application domain boundaries; a new thread is not created for each application domain.

At any given time, every thread executes in an application domain. Zero, one, or multiple threads might be executing in any given application domain. The runtime keeps track of which threads are running in which application domains. You can locate the domain in which a thread is executing at any time by calling the Thread.GetDomain method.

Culture, which is represented by a CultureInfo object, is associated with threads. You can get the culture that is associated with the currently executing thread by using the CultureInfo.CurrentCulture property, and you can get or set the culture that is associated with the currently executing thread by using the Thread.CurrentCulture property. If the culture that is associated with a thread has been explicitly set by using the Thread.CurrentCulture property, it continues to be associated with that thread when the thread crosses application domain boundaries. Otherwise, the culture that is associated with the thread at any given time is determined by the value of the CultureInfo.DefaultThreadCurrentCulture property in the application domain in which the thread is executing:

- If the value of the property is not `null`, the culture that is returned by the property is associated with the thread (and therefore returned by the Thread.CurrentCulture and CultureInfo.CurrentCulture properties).
- If the value of the property is `null`, the current system culture is associated with the thread.

Application domains are usually created and manipulated programmatically by runtime hosts. However, sometimes an application program might also want to work with application domains. For example, an application program could load an application component into a domain to be able to unload the domain (and the component) without having to stop the entire application.

The AppDomain is the programmatic interface to application domains. This class includes methods to create and unload domains, to create instances of types in domains, and to register for various notifications such as application domain unloading. The following table lists commonly used AppDomain methods.

AppDomain Method	Description
CreateDomain	Creates a new application domain. It is recommended that you use an overload of this method that specifies an AppDomainSetup object. This is the preferred way to set the properties of a new domain, such as the application base, or root directory for the application; the location of the configuration file for the domain; and the search path that the common language runtime is to use to load assemblies into the domain.
ExecuteAssembly and ExecuteAssemblyByName	Executes an assembly in the application domain. This is an instance method, so it can be used to execute code in another application domain to which you have a reference.
CreateInstanceAndUnwrap	Creates an instance of a specified type in the application domain, and returns a proxy. Use this method to avoid loading the assembly containing the created type into the calling assembly.
Unload	Performs a graceful shutdown of the domain. The application domain is not unloaded until all threads running in the domain have either stopped or are no longer in the domain.

Note

The common language runtime does not support serialization of global methods, so delegates cannot be used to execute global methods in other application domains.

The unmanaged interfaces described in the common language runtime Hosting Interfaces Specification also provide access to application domains. Runtime hosts can use interfaces from unmanaged code to create and gain access to the application domains within a process.

An environment variable that sets the default loader optimization policy of an executable application.

```
COMPLUS_LoaderOptimization = 1
```

A typical application loads several assemblies into an application domain before the code they contain can be executed.

The way the assembly is loaded determines whether its just-in-time (JIT) compiled code can be shared by multiple application domains in the process.

- If an assembly is loaded domain-neutral, all application domains that share the same security grant set can share the same JIT-compiled code. This reduces the memory required by the application.
- If an assembly is not loaded domain-neutral, it must be JIT-compiled in every application domain in which it is loaded and the loader must not share internal resources across application domains.

When set to 1, the COMPLUS_LoaderOptimization environment flag forces the runtime host to load all assemblies in non-domain-neutral way known as SingleDomain. SingleDomain loads no assemblies as domain-neutral, except Mscorlib, which is always loaded domain-neutral. This setting is called single domain because it is commonly used when the host is running only a single application in the process.

Caution

The COMPLUS_LoaderOptimization environment flag was designed to be used in diagnostic and test scenarios. Having the flag turned on can cause severe slow-down and increase in memory usage.

To force all assemblies not to be loaded as domain-neutral for the IISADMIN service can be achieved by appending `COMPLUS_LoaderOptimization=1` to the Environment's Multi-String Value in the HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\IISADMIN key.

```
 Key = HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\IISADMIN
Name = Environment
Type = REG_MULTI_SZ
Value (to append) = COMPLUS_LoaderOptimization=1
```

- System.AppDomain
- System.MarshalByRefObject
- Programming with Application Domains and Assemblies
- Using Application Domains
