This page is specific to

Microsoft Visual Studio 2010/.NET Framework 4

.NET Framework 4

**Application Domains**

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.

The topics in this section of the documentation explain how to use application domains to provide isolation between assemblies.

This overview contains the following sections:

* [The Benefits of Isolating Applications](http://msdn.microsoft.com/en-us/library/2bh4z9hs.aspx#benefits)
* [Related Topics](http://msdn.microsoft.com/en-us/library/2bh4z9hs.aspx#related_topics)
* [Reference](http://msdn.microsoft.com/en-us/library/2bh4z9hs.aspx#reference)

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifThe Benefits of Isolating Applications

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.

|  |
| --- |
| **Description: NoteNote** |
| 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 **System.IO.FileNotFound**. See [Remote Objects](http://msdn.microsoft.com/en-us/library/72x4h507.aspx) for more details. The mechanism for determining how objects can be accessed across domains is determined by the object. For more information, see [MarshalByRefObject Class](http://msdn.microsoft.com/en-us/library/system.marshalbyrefobject.aspx).
* 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.