**How the Runtime Locates Assemblies**

**.NET Framework 4.5**

To successfully deploy your .NET Framework application, you must understand how the common language runtime locates and binds to the assemblies that make up your application. By default, the runtime attempts to bind with the exact version of an assembly that the application was built with. This default behavior can be overridden by configuration file settings.

The common language runtime performs a number of steps when attempting to locate an assembly and resolve an assembly reference. Each step is explained in the following sections. The term probing is often used when describing how the runtime locates assemblies; it refers to the set of heuristics used to locate the assembly based on its name and culture.

|  |
| --- |
| **Description: NoteNote** |
| You can view binding information in the log file using the [Assembly Binding Log Viewer (Fuslogvw.exe)](http://msdn.microsoft.com/en-us/library/e74a18c4.aspx), which is included in the Windows Software Development Kit (SDK). |

[**Initiating the Bind**](javascript:void(0))

The process of locating and binding to an assembly begins when the runtime attempts to resolve a reference to another assembly. This reference can be either static or dynamic. The compiler records static references in the assembly manifest's metadata at build time. Dynamic references are constructed on the fly as a result of calling various methods, such as [System.Reflection.Assembly.Load](http://msdn.microsoft.com/en-us/library/system.reflection.assembly.load.aspx).

The preferred way to reference an assembly is to use a full reference, including the assembly name, version, culture, and public key token (if one exists). The runtime uses this information to locate the assembly, following the steps described later in this section. The runtime uses the same resolution process regardless of whether the reference is for a static or dynamic assembly.

You can also make a dynamic reference to an assembly by providing the calling method with only partial information about the assembly, such as specifying only the assembly name. In this case, only the application directory is searched for the assembly, and no other checking occurs. You make a partial reference using any of the various methods for loading assemblies such as [System.Reflection.Assembly.Load](http://msdn.microsoft.com/en-us/library/system.reflection.assembly.load.aspx) or [System.AppDomain.Load](http://msdn.microsoft.com/en-us/library/system.appdomain.load.aspx).

Finally, you can make a dynamic reference using a method such as [System.Reflection.Assembly.Load](http://msdn.microsoft.com/en-us/library/system.reflection.assembly.load.aspx) and provide only partial information; you then qualify the reference using the [<qualifyAssembly>](http://msdn.microsoft.com/en-us/library/cd71chf0.aspx) element in the application configuration file. This element allows you to provide the full reference information (name, version, culture and, if applicable, the public key token) in your application configuration file instead of in your code. You would use this technique if you wanted to fully qualify a reference to an assembly outside the application directory, or if you wanted to reference an assembly in the global assembly cache but you wanted the convenience of specifying the full reference in the configuration file instead of in your code.

|  |
| --- |
| **Description: NoteNote** |
| This type of partial reference should not be used with assemblies that are shared among several applications. Because configuration settings are applied per application and not per assembly, a shared assembly using this type of partial reference would require each application using the shared assembly to have the qualifying information in its configuration file. |

The runtime uses the following steps to resolve an assembly reference:

1. [Determines the correct assembly version](http://msdn.microsoft.com/en-us/library/8f6988ab.aspx) by examining applicable configuration files, including the application configuration file, publisher policy file, and machine configuration file. If the configuration file is located on a remote machine, the runtime must locate and download the application configuration file first.
2. [Checks whether the assembly name has been bound to before](http://msdn.microsoft.com/en-us/library/aa98tba8.aspx) and, if so, uses the previously loaded assembly. If a previous request to load the assembly failed, the request is failed immediately without attempting to load the assembly.

|  |
| --- |
| **Description: NoteNote** |
| The caching of assembly binding failures is new in the .NET Framework version 2.0. |

1. [Checks the global assembly cache](http://msdn.microsoft.com/en-us/library/4a9t8a9a.aspx). If the assembly is found there, the runtime uses this assembly.
2. [Probes for the assembly](http://msdn.microsoft.com/en-us/library/15hyw9x3.aspx) using the following steps:
   1. If configuration and publisher policy do not affect the original reference and if the bind request was created using the [AssemblyLoadFrom](http://msdn.microsoft.com/en-us/library/system.reflection.assembly.loadfrom.aspx) method, the runtime checks for location hints.
   2. If a codebase is found in the configuration files, the runtime checks only this location. If this probe fails, the runtime determines that the binding request failed and no other probing occurs.
   3. Probes for the assembly using the heuristics described in the [probing section](http://msdn.microsoft.com/en-us/library/15hyw9x3.aspx). If the assembly is not found after probing, the runtime requests the Windows Installer to provide the assembly. This acts as an install-on-demand feature.

|  |
| --- |
| **Description: NoteNote** |
| There is no version checking for assemblies without strong names, nor does the runtime check in the global assembly cache for assemblies without strong names. |