.NET Framework 4

**Ngen.exe (Native Image Generator)**

Updated: April 2011

The Native Image Generator (Ngen.exe) is a tool that improves the performance of managed applications. Ngen.exe creates native images, which are files containing compiled processor-specific machine code, and installs them into the native image cache on the local computer. The runtime can use native images from the cache instead of using the just-in-time (JIT) compiler to compile the original assembly.

Changes to Ngen.exe in the .NET Framework version 4:

* Ngen.exe now compiles assemblies with full trust, and code access security (CAS) policy is no longer evaluated.
* Native images that are generated with Ngen.exe can no longer be loaded into applications that are running in partial trust.

Changes to Ngen.exe in the .NET Framework version 2.0:

* Installing an assembly also installs its dependencies, simplifying the syntax of Ngen.exe.
* Native images can now be shared across application domains.
* A new action, **update**, re-creates images that have been invalidated.
* Actions can be deferred for execution by a service that uses idle time on the computer to generate and install images.
* Some causes of image invalidation have been eliminated.

For additional information on using Ngen.exe and the native image service, see [Native Image Service](http://msdn.microsoft.com/en-us/library/ms165074.aspx).

|  |
| --- |
| **Note** |
| Ngen.exe syntax for versions 1.0 and 1.1 of the .NET Framework can be found in [Native Image Generator (Ngen.exe) Legacy Syntax](http://msdn.microsoft.com/en-us/library/ms165073.aspx). |

This tool is automatically installed with Visual Studio and with the Windows SDK. To run the tool, we recommend that you use the Visual Studio Command Prompt or the Windows SDK Command Prompt (CMD Shell). These utilities enable you to run the tool easily, without navigating to the installation folder. For more information, see [Visual Studio and Windows SDK Command Prompts](http://msdn.microsoft.com/en-us/library/ms229859.aspx).

* If you have Visual Studio installed on your computer: On the taskbar, click **Start**, click **All Programs**, click **Visual Studio**, click **Visual Studio Tools**, and then click **Visual Studio Command Prompt**.

-or-

If you have the Windows SDK installed on your computer: On the taskbar, click **Start**, click **All Programs**, click the folder for the Windows SDK, and then click **Command Prompt** (or **CMD Shell**).

* At the command prompt, type the following:

ngen action [options]

ngen /? | /help

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifActions

The following table shows the syntax of each *action*. For descriptions of the individual parts of an *action*, see the [Arguments](http://msdn.microsoft.com/en-us/library/6t9t5wcf.aspx#ArgumentTable), [Priority Levels](http://msdn.microsoft.com/en-us/library/6t9t5wcf.aspx#PriorityTable), [Scenarios](http://msdn.microsoft.com/en-us/library/6t9t5wcf.aspx#ScenarioTable), and [Config](http://msdn.microsoft.com/en-us/library/6t9t5wcf.aspx#ConfigTable) tables. The [Options](http://msdn.microsoft.com/en-us/library/6t9t5wcf.aspx#OptionTable) table describes the *options* and the help switches.

|  |  |
| --- | --- |
| **Action** | **Description** |
| **install** [*assemblyName* | *assemblyPath*] [*scenarios*] [*config*] [**/queue**[**:**{**1**|**2**|**3**}]] | Generate native images for an assembly and its dependencies and install the images in the native image cache.  If **/queue** is specified, the action is queued for the native image service. The default priority is 3. See the [Priority Levels](http://msdn.microsoft.com/en-us/library/6t9t5wcf.aspx#PriorityTable) table. |
| **uninstall** [*assemblyName* | *assemblyPath*] [*scenarios*] [*config*] | Delete the native images of an assembly and its dependencies from the native image cache.  To uninstall a single image and its dependencies, use the same command-line arguments that were used to install the image.  **Note**  In the .NET Framework version 4, the action **uninstall** \* is no longer supported. |
| **update** [**/queue**] | Update native images that have become invalid.  If **/queue** is specified, the updates are queued for the native image service. Updates are always scheduled at priority 3, so they run when the computer is idle. |
| **display** [*assemblyName* | *assemblyPath*] | Display the state of the native images for an assembly and its dependencies.  If no argument is supplied, everything in the native image cache is displayed. |
| **executeQueuedItems** [**1***|***2***|***3**]  -or-  **eqi** [1|2|3] | Execute queued compilation jobs.  If a priority is specified, compilation jobs with greater or equal priority are executed. If no priority is specified, all queued compilation jobs are executed. |
| **queue** {**pause** | **continue** | **status**} | Pause the native image service, allow the paused service to continue, or query the status of the service. |

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifArguments

|  |  |
| --- | --- |
| **Argument** | **Description** |
| *assemblyName* | The full display name of the assembly. For example, "myAssembly, Version=2.0.0.0, Culture=neutral, PublicKeyToken=0038abc9deabfle5".  **Note**  You can supply a partial assembly name, such as myAssembly, for the **display** and **uninstall** actions.  Only one assembly can be specified per Ngen.exe command line. |
| *assemblyPath* | The explicit path of the assembly. You can specify a full or relative path.  If you specify a file name without a path, the assembly must be located in the current directory.  Only one assembly can be specified per Ngen.exe command line. |

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifPriority Levels

|  |  |
| --- | --- |
| **Priority** | **Description** |
| **1** | Native images are generated and installed immediately, without waiting for idle time. |
| **2** | Native images are generated and installed without waiting for idle time, but after all priority 1 actions (and their dependencies) have completed. |
| **3** | Native images are installed when the native image service detects that the computer is idle. See [Native Image Service](http://msdn.microsoft.com/en-us/library/ms165074.aspx). |

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifScenarios

|  |  |
| --- | --- |
| **Scenario** | **Description** |
| **/Debug** | Generate native images that can be used under a debugger. |
| **/Profile** | Generate native images that can be used under a profiler. |
| **/NoDependencies** | Generate the minimum number of native images required by the specified scenario options. |

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifConfig

|  |  |
| --- | --- |
| **Configuration** | **Description** |
| **/ExeConfig:***exePath* | Use the configuration of the specified executable assembly.  Ngen.exe needs to make the same decisions as the loader when binding to dependencies. When a shared component is loaded at run time, using the [Load](http://msdn.microsoft.com/en-us/library/system.reflection.assembly.load.aspx) method, the application's configuration file determines the dependencies that are loaded for the shared component — for example, the version of a dependency that is loaded. The **/ExeConfig** switch gives Ngen.exe guidance on which dependencies would be loaded at run time. |
| **/AppBase:***directoryPath* | When locating dependencies, use the specified directory as the application base. |

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifOptions

|  |  |
| --- | --- |
| **Option** | **Description** |
| **/nologo** | Suppress the Microsoft startup banner display. |
| **/silent** | Suppress the display of success messages. |
| **/verbose** | Display detailed information for debugging.  **Note**  Due to operating system limitations, this option does not display as much additional information on Windows 98 and Windows Millennium Edition. |
| **/help**, **/?** | Display command syntax and options for the current release. |

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifRemarks

To run Ngen.exe, you must have administrative privileges.

|  |
| --- |
| **Caution** |
| Do not run Ngen.exe on assemblies that are not fully trusted. Starting with the .NET Framework version 4, Ngen.exe compiles assemblies with full trust, and code access security (CAS) policy is no longer evaluated. |

Starting with the .NET Framework 4, the native images that are generated with Ngen.exe can no longer be loaded into applications that are running in partial trust. Instead, the just-in-time (JIT) compiler is invoked.

Ngen.exe generates native images for the specified assembly and all its dependencies. Dependencies are determined from references in the assembly manifest. The only scenario in which you need to install a dependency separately is when the application loads it using reflection, for example by calling the [Assembly..::.Load](http://msdn.microsoft.com/en-us/library/system.reflection.assembly.load.aspx) method.

|  |
| --- |
| **Important** |
| Do not use the [Assembly..::.LoadFrom](http://msdn.microsoft.com/en-us/library/system.reflection.assembly.loadfrom.aspx) method with native images. An image loaded with this method cannot be used by other assemblies in the execution context. |

Ngen.exe maintains a count on dependencies. For example, suppose MyAssembly.exe and YourAssembly.exe are both installed in the native image cache, and both have references to OurDependency.dll. If MyAssembly.exe is uninstalled, OurDependency.dll is not uninstalled. It is only removed when YourAssembly.exe is also uninstalled.

If you are generating a native image for an assembly in the global assembly cache, specify its display name. See [Assembly..::.FullName](http://msdn.microsoft.com/en-us/library/system.reflection.assembly.fullname.aspx).

The native images that Ngen.exe generates can be shared across application domains. This means you can use Ngen.exe in application scenarios that require assemblies to be shared across application domains. To specify domain neutrality:

* Apply the [LoaderOptimizationAttribute](http://msdn.microsoft.com/en-us/library/system.loaderoptimizationattribute.aspx) attribute to your application.
* Set the [AppDomainSetup..::.LoaderOptimization](http://msdn.microsoft.com/en-us/library/system.appdomainsetup.loaderoptimization.aspx) property when you create setup information for a new application domain.

Always use domain-neutral code when loading the same assembly into multiple application domains. If a native image is loaded into a nonshared application domain after having been loaded into a shared domain, it cannot be used.

|  |
| --- |
| **Note** |
| Domain-neutral code cannot be unloaded, and performance may be slightly slower, particularly when accessing static members. |

**Generating Images for Different Scenarios**

After you have generated a native image for an assembly, the runtime automatically attempts to locate and use this native image each time it runs the assembly. Multiple images can be generated, depending on usage scenarios.

For example, if you run an assembly in a debugging or profiling scenario, the runtime looks for a native image that was generated with the **/Debug** or **/Profile** options. If it is unable to find a matching native image, the runtime reverts to standard JIT compilation. The only way to debug native images is to create a native image with the **/Debug** option.

The **uninstall** action also recognize scenarios, so you can uninstall all scenarios or only selected scenarios.

**Determining When to Use Native Images**

Native images can provide performance improvements in two areas: improved memory use and reduced startup time.

|  |
| --- |
| **Note** |
| Performance of native images depends on a number of factors that make analysis difficult, such as code and data access patterns, how many calls are made across module boundaries, and how many dependencies have already been loaded by other applications. The only way to determine whether native images benefit your application is by careful performance measurements in your key deployment scenarios. |

**Improved Memory Use**

Native images can significantly improve memory use when code is shared between processes. Native images are Windows PE files, so a single copy of a .dll file can be shared by multiple processes; by contrast, native code produced by the JIT compiler is stored in private memory and cannot be shared.

Applications that are run under terminal services can also benefit from shared code pages.

In addition, not loading the JIT compiler saves a fixed amount of memory for each application instance.

**Faster Application Startup**

Precompiling assemblies with Ngen.exe can improve the startup time for some applications. In general, gains can be made when applications share component assemblies because after the first application has been started the shared components are already loaded for subsequent applications. Cold startup, in which all the assemblies in an application must be loaded from the hard disk, does not benefit as much from native images because the hard disk access time predominates.

Hard binding can affect startup time, because all images that are hard bound to the main application assembly must be loaded at the same time.

|  |
| --- |
| **Note** |
| Before the .NET Framework version 3.5 Service Pack 1, you should put shared, strong-named components in the global assembly cache, because the loader performs extra validation on strong-named assemblies that are not in the global assembly cache, effectively eliminating any improvement in startup time gained by using native images. Optimizations that were introduced in the .NET Framework 3.5 SP1 removed the extra validation. |

**Importance of Assembly Base Addresses**

Because native images are Windows PE files, they are subject to the same rebasing issues as other executable files. The performance cost of relocation is even more pronounced if hard binding is employed.

To set the base address for a native image, use the appropriate option of your compiler to set the base address for the assembly. Ngen.exe uses this base address for the native image.

|  |
| --- |
| **Note** |
| Native images are larger than the managed assemblies from which they were created. Base addresses must be calculated to allow for these larger sizes. |

You can use a tool such as dumpbin.exe to view the preferred base address of a native image.

**Summary of Usage Considerations**

The following general considerations and application considerations may assist you in deciding whether to undertake the effort of evaluating native images for your application:

* Native images load faster than MSIL because they eliminate the need for many startup activities, such as JIT compilation and type-safety verification.
* Native images require a smaller initial working set because there is no need for the JIT compiler.
* Native images enable code sharing between processes.
* Native images require more hard disk space than MSIL assemblies and may require considerable time to generate.
* Native images must be maintained.
  + Images need to be regenerated when the original assembly or one of its dependencies is serviced.
  + A single assembly may need multiple native images for use in different applications or different scenarios. For example, the configuration information in two applications might result in different binding decisions for the same dependent assembly.
  + Native images must be generated by an administrator; that is, from a Windows account in the Administrators group.

In addition to these general considerations, the nature of your application must be considered when determining whether native images might provide a performance benefit:

* If your application runs in an environment that uses many shared components, native images allow the components to be shared by multiple processes.
* If your application uses multiple application domains, native images allow code pages to be shared across domains.

|  |
| --- |
| **Note** |
| In the .NET Framework versions 1.0 and 1.1, native images cannot be shared across application domains. This is not the case in version 2.0 or later. |

* If your application will be run under Terminal Server, native images allow sharing of code pages.
* Large applications generally benefit from compilation to native images. Small applications generally do not benefit.
* For long-running applications, run-time JIT compilation performs slightly better than native images. (Hard binding can mitigate this performance difference to some degree.)

**Hard Binding**

Hard binding increases throughput and reduces working set size for native images. The disadvantage of hard binding is that all the images that are hard bound to an assembly must be loaded when the assembly is loaded. This can significantly increase startup time for a large application.

Hard binding is appropriate for dependencies that are loaded in all your application's performance-critical scenarios. As with any aspect of native image use, careful performance measurements are the only way to determine whether hard binding improves your application's performance.

The [DependencyAttribute](http://msdn.microsoft.com/en-us/library/system.runtime.compilerservices.dependencyattribute.aspx) and [DefaultDependencyAttribute](http://msdn.microsoft.com/en-us/library/system.runtime.compilerservices.defaultdependencyattribute.aspx) attributes allow you to provide hard binding hints to Ngen.exe.

|  |
| --- |
| **Note** |
| These attributes are hints to Ngen.exe, not commands. Using them does not guarantee hard binding. The meaning of these attributes may change in future releases. |

**Specifying a Binding Hint for a Dependency**

Apply the [DependencyAttribute](http://msdn.microsoft.com/en-us/library/system.runtime.compilerservices.dependencyattribute.aspx) to an assembly to indicate the likelihood that a specified dependency will be loaded. LoadHint..::.Always indicates that hard binding is appropriate, Default indicates that the default for the dependency should be used, and Sometimes indicates that hard binding is not appropriate.

The following code shows the attributes for an assembly that has two dependencies. The first dependency (Assembly1) is an appropriate candidate for hard binding, and the second (Assembly2) is not.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl100_ctl00_ctl19_code');" \o "Copy Code)

Imports System.Runtime.CompilerServices

<Assembly:DependencyAttribute("Assembly1", LoadHint.Always)>

<Assembly:DependencyAttribute("Assembly2", LoadHint.Sometimes)>

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl100_ctl00_ctl20_code');" \o "Copy Code)

using System.Runtime.CompilerServices;

[assembly:DependencyAttribute("Assembly1", LoadHint.Always)]

[assembly:DependencyAttribute("Assembly2", LoadHint.Sometimes)]

Visual C++

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl100_ctl00_ctl21_code');" \o "Copy Code)

using namespace System::Runtime::CompilerServices;

[assembly:DependencyAttribute("Assembly1", LoadHint.Always)];

[assembly:DependencyAttribute("Assembly2", LoadHint.Sometimes)];

The assembly name does not include the file name extension. Display names can be used.

**Specifying a Default Binding Hint for an Assembly**

Default binding hints are only needed for assemblies that will be used immediately and frequently by any application that has a dependency on them. Apply the [DefaultDependencyAttribute](http://msdn.microsoft.com/en-us/library/system.runtime.compilerservices.defaultdependencyattribute.aspx) with LoadHint..::.Always to such assemblies to specify that hard binding should be used.

|  |
| --- |
| **Note** |
| There is no reason to apply [DefaultDependencyAttribute](http://msdn.microsoft.com/en-us/library/system.runtime.compilerservices.defaultdependencyattribute.aspx) to .dll assemblies that do not fall into this category, because applying the attribute with any value other than LoadHint..::.Always has the same effect as not applying the attribute at all. |

Microsoft uses the [DefaultDependencyAttribute](http://msdn.microsoft.com/en-us/library/system.runtime.compilerservices.defaultdependencyattribute.aspx) to specify that hard binding is the default for a very small number of assemblies in the .NET Framework, such as mscorlib.dll.

**Troubleshooting**

To confirm that native images are being used by your application, you can use the [Fuslogvw.exe (Assembly Binding Log Viewer)](http://msdn.microsoft.com/en-us/library/e74a18c4.aspx). Select **Native Images** in the **Log Categories** box on the binding log viewer window. Fuslogvw.exe provides information about why a native image was rejected.

You can use the [jitCompilationStart MDA](http://msdn.microsoft.com/en-us/library/fw872k46.aspx) managed debugging assistant (MDA) to determine when the JIT compiler starts to compile a function.

**Deferred Processing**

Generation of native images for a very large application can take considerable time. Similarly, changes to a shared component or changes to computer settings might require many native images to be updated. The **install** and **update** actions have a **/queue** option that queues the operation for deferred execution by the native image service. In addition, Ngen.exe has **queue** and **executeQueuedItems** actions that provide some control over the service. For more information, see [Native Image Service](http://msdn.microsoft.com/en-us/library/ms165074.aspx).

**Native Images and JIT Compilation**

If Ngen.exe encounters any methods in an assembly that it cannot generate, it excludes them from the native image. When the runtime executes this assembly, it reverts to JIT compilation for the methods that were not included in the native image.

In addition, native images are not used if the assembly has been upgraded, or if the image has been invalidated for any reason.

**Invalid Images**

When you use Ngen.exe to create a native image of an assembly, the output depends upon the command-line options that you specify and certain settings on your computer. These settings include the following:

* The version of the .NET Framework.
* The version of the operating system, if the change is from the Windows 9x family to the Windows NT family.
* The exact identity of the assembly (recompilation changes identity).
* The exact identity of all assemblies that the assembly references (recompilation changes identity).
* Security factors.

Ngen.exe records this information when it generates a native image. When you execute an assembly, the runtime looks for the native image generated with options and settings that match the computer's current environment. The runtime reverts to JIT compilation of an assembly if it cannot find a matching native image. The following changes to a computer's settings and environment cause native images to become invalid:

* The version of the .NET Framework.

If you apply an update to the .NET Framework, all native images that you have created using Ngen.exe become invalid. For this reason, all updates of the .NET Framework execute the Ngen Update command, to ensure that all native images are regenerated. The .NET Framework automatically creates new native images for the .NET Framework libraries that it installs.

* The version of the operating system, if the change is from the Windows 9x family to the Windows NT family.

For example, if the version of the operating system running on a computer changes from Windows 98 to Windows XP, all native images stored in the native image cache become invalid. However, if the operating system changes from Windows 2000 to Windows XP, the images are not invalidated.

* The exact identity of the assembly.

If you recompile an assembly, the assembly's corresponding native image becomes invalid.

* The exact identity of any assemblies the assembly references.

If you update a managed assembly, all native images that directly or indirectly depend on that assembly become invalid and need to be regenerated. This includes both ordinary references and hard-bound dependencies. Whenever a software update is applied, the installation program should execute an Ngen Update command to ensure that all dependent native images are regenerated.

* Security factors.

Changing machine security policy to restrict permissions previously granted to an assembly can cause a previously compiled native image for that assembly to become invalid.

For detailed information about how the common language runtime administers code access security and how to use permissions, see [Code Access Security](http://msdn.microsoft.com/en-us/library/c5tk9z76.aspx).

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifExamples

The following command generates a native image for ClientApp.exe, located in the current directory, and installs the image in the native image cache. If a configuration file exists for the assembly, Ngen.exe uses it. In addition, native images are generated for any .dll files that ClientApp.exe references.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl101_ctl00_ctl00_code');" \o "Copy Code)

ngen install ClientApp.exe

An image installed with Ngen.exe is also called a root. A root can be an application or a shared component.

The following command generates a native image for MyAssembly.exe with the specified path.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl101_ctl00_ctl01_code');" \o "Copy Code)

ngen install c:\myfiles\MyAssembly.exe

When locating assemblies and their dependencies, Ngen.exe uses the same probing logic used by the common language runtime. By default, the directory that contains ClientApp.exe is used as the application base directory, and all assembly probing begins in this directory. You can override this behavior by using the **/AppBase** option.

|  |
| --- |
| **Note** |
| This is a change from Ngen.exe behavior in the .NET Framework versions 1.0 and 1.1, where the application base is set to the current directory. |

An assembly can have a dependency without a reference, for example if it loads a .dll file by using the [Assembly..::.Load](http://msdn.microsoft.com/en-us/library/system.reflection.assembly.load.aspx) method. You can create a native image for such a .dll file by using configuration information for the application assembly, with the **/ExeConfig** option. The following command generates a native image for MyLib.dll, using the configuration information from MyApp.exe.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl101_ctl00_ctl04_code');" \o "Copy Code)

ngen install c:\myfiles\MyLib.dll /ExeConfig:c:\myapps\MyApp.exe

Assemblies installed in this way are not removed when the application is removed.

To uninstall a dependency, use the same command-line options that were used to install it. The following command uninstalls the MyLib.dll from the previous example.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl101_ctl00_ctl05_code');" \o "Copy Code)

ngen uninstall c:\myfiles\MyLib.dll /ExeConfig:c:\myapps\MyApp.exe

To create a native image for an assembly in the global assembly cache, use the display name of the assembly. For example:

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl101_ctl00_ctl06_code');" \o "Copy Code)

ngen install "ClientApp, Version=1.0.0.0, Culture=neutral,

PublicKeyToken=3c7ba247adcd2081, processorArchitecture=MSIL"

NGen.exe generates a separate set of images for each scenario you install. For example, the following commands install a complete set of native images for normal operation, another complete set for debugging, and a third for profiling:

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl101_ctl00_ctl07_code');" \o "Copy Code)

ngen install MyApp.exe

ngen install MyApp.exe /debug

ngen install MyApp.exe /profile

**Displaying the Native Image Cache**

Once native images are installed in the cache, they can be displayed using Ngen.exe. The following command displays all native images in the native image cache.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl101_ctl00_ctl08_code');" \o "Copy Code)

ngen display

The **display** action lists all the root assemblies first, followed by a list of all the native images on the computer.

Use the simple name of an assembly to display information only for that assembly. The following command displays all native images in the native image cache that match the partial name MyAssembly, their dependencies, and all roots that have a dependency on MyAssembly:

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl101_ctl00_ctl09_code');" \o "Copy Code)

ngen display MyAssembly

Knowing what roots depend on a shared component assembly is useful in gauging the impact of an **update** action after the shared component is upgraded.

If you specify an assembly's file extension, you must either specify the path or execute Ngen.exe from the directory containing the assembly:

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl101_ctl00_ctl10_code');" \o "Copy Code)

ngen display c:\myApps\MyAssembly.exe

The following command displays all native images in the native image cache with the name MyAssembly and the version 1.0.0.0.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl101_ctl00_ctl11_code');" \o "Copy Code)

ngen display "myAssembly, version=1.0.0.0"

**Updating Images**

Images are typically updated after a shared component has been upgraded. To update all native images that have changed, or whose dependencies have changed, use the **update** action with no arguments.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl101_ctl00_ctl12_code');" \o "Copy Code)

ngen update

Updating all images can be a lengthy process. You can queue the updates for execution by the native image service by using the **/queue** option. For more information on the **/queue** option and installation priorities, see [Native Image Service](http://msdn.microsoft.com/en-us/library/ms165074.aspx).

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl101_ctl00_ctl14_code');" \o "Copy Code)

ngen update /queue

**Uninstalling Images**

Ngen.exe maintains a list of dependencies, so that shared components are removed only when all assemblies that depend on them have been removed. In addition, a shared component is not removed if it has been installed as a root.

The following command uninstalls all scenarios for the root ClientApp.exe:

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl101_ctl00_ctl15_code');" \o "Copy Code)

ngen uninstall ClientApp

The **uninstall** action can be used to remove specific scenarios. The following command uninstalls all debug scenarios for ClientApp.exe:

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl101_ctl00_ctl16_code');" \o "Copy Code)

ngen uninstall ClientApp /debug

|  |
| --- |
| **Note** |
| Uninstalling **/debug** scenarios does not uninstall a scenario that includes both **/profile** and **/debug.** |

The following command uninstalls all scenarios for a specific version of ClientApp.exe:

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl101_ctl00_ctl18_code');" \o "Copy Code)

ngen uninstall "ClientApp, Version=1.0.0.0"

The following commands uninstall all scenarios for "ClientApp, Version=1.0.0.0, Culture=neutral, PublicKeyToken=3c7ba247adcd2081, processorArchitecture=MSIL", or just the debug scenario for that assembly:

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl101_ctl00_ctl19_code');" \o "Copy Code)

ngen uninstall "ClientApp, Version=1.0.0.0, Culture=neutral,

PublicKeyToken=3c7ba247adcd2081, processorArchitecture=MSIL"

ngen uninstall "ClientApp, Version=1.0.0.0, Culture=neutral,

PublicKeyToken=3c7ba247adcd2081, processorArchitecture=MSIL" /debug

As with the **install** action, supplying an extension requires either executing Ngen.exe from the directory containing the assembly or specifying a full path.

For examples relating to the native image service, see [Native Image Service](http://msdn.microsoft.com/en-us/library/ms165074.aspx).