Bong Geek - Abhinaba Basu: .NET: Loading Native (NGEN) images and its interaction with the GAC

blog.bonggeek.com/2013/12/net-loading-native-ngen-images-and-its.html

Wednesday, December 11, 2013

.NET: Loading Native (NGEN) images and its interaction with the GAC

It's common for people to think that NGEN works with strong named assemblies only and it places output files or uses GAC closely. This is mostly not true.
If you are new to this there's a quick primer on NGEN that I wrote http://blogs.msdn.com/b/abhinaba/archive/2013/12/10/net-ngen-gac-and-their-interactions.aspx.

GAC

The Global Assembly Cache or GAC is a central repository where managed assemblies can be placed either using the command like gacutil tool or programmatically using Fusion APIs. The main benefits of GAC is

1. Avoid dll hell

2. Provide for a central place to discover dependencies (place your binary in the central place and other applications will find it)
3. Allow side-by-side publication of multiple versions of the same assembly
4. Way to apply critical patches, especially security patches that will automatically flow all app using that assembly
5. Sharing of assemblies across processes. Particularly helpful for system assemblies that are used in most managed assemblies
6. Provide custom versioning mechanisms (e.g. assembly re-directs / publisher policies)

While GAC has it's uses it has its problems as well. One of the primary problem being that an assembly has to be strongly named to be placed in GAC and it's not always possible to do that. E.g. read here and here.

NIC

The NIC or **N**ative **I**mage **C**ache is the location where NGEN places native images. When NGEN is run to create a native image as in

```
c:\Projects&gt;ngen.exe install MyMathLibrary.dll
```

The corresponding MyMathLibrary.ni.dll is placed in the NIC. The NIC has a similar purpose as GAC but is not the same location. NIC is placed at <Windows Dir>\assembly\NativeImages_<CLRversion>_<arch>. E.g. a sample path is

c:\Windows\assembly\NativeImages_v4.0.30319_64\MyMathLibrary\7ed4d51aae956cce52d0809914a3afb3\MyMathLibrary.ni.dll

NGEN places the files it generates in NIC along with other metadata to ensure that it can reliably find the right native image corresponding to an IL image.

How does the .NET Binder find valid native images

The CLR module that finds assemblies for execution is called the Binder. There are various kinds of binders that CLR uses. The one used to find native images for a given assembly is called the NativeBinder.

Finding the right native image involves two steps. First the IL image and the corresponding potential native image is located on the file system and then verification is made to ensure

that the native image is indeed a valid image for that IL. E.g. the runtime gets a request to bind against an assembly MyMathLibrary.dll as another assembly program.exe has dependency on it. This is what will happen

1. First the standard fusion binder will kick in to find that assembly. It can find it either in
    a. GAC, which means it is strongly named. The way files are placed in GAC ensures that the binder can extract all the required information about the assembly without physically opening the file
    b. Find it the APPBASE (E.g. the local folder of program.exe). It will proceed to open the IL file and read the assemblies metadata
2. Native binding will proceed only in the default context (more about this in a later post)
3. The NativeBinder finds the NI file from the NIC. It reads in the NI file details and metadata
4. Verifies the NI is indeed for that very same IL assembly. For that it goes through a rigorous matching process which includes (but not limited to) full assembly name match (same name, version, public key tokens, culture), time stamp matching (NI has to be newer than IL), MVID (see below)
5. Also verifies that the NI has been generated for the same CLR under which it is going to be run (exact .NET version, processor type, etc…) .
6. Also ensures that the NI's dependencies are also valid. E.g. when the NI was generated it bound against a particular version of mscorlib. If that mscorlib native image is not valid then this NI image is also rejected

The question is what happens if the assembly is not strongly named? The answer is in that case MVID is used to match instead of relying on say the signing key tokens. MVID is a guid that is embedded in an IL file when a compiler compiles it. If you compile an assembly multiple times, each time the IL file is generated it has an unique MVID. If you open any managed assembly using ildasm and double lick on it's manifest you can see the MVID

.module MyMathLibrary.dll
// MVID: {**EEEBEA21-D58F-44C6-9FD2-22B57F4D0193**}

If you re-compile and re-open you should see a new id. This fact is used by the NativeBinder as well. NGEN stores the mvid of the IL file for which a NI is generated. Later the native binder ensures that the MVID of the IL file matches with the MVID of the IL file for which the NI file was generated. This step ensures that if you have multiple common.dll in your PC and all of which has version 0.0.0.0 and is not signed, even then NI for one of the common.dll will not get used for another common.dll.

The Double Loading Problem

In early version of .NET when a NI file was opened the corresponding IL file was also opened. I found a 2003 post from Jason Zander on this. However, currently this is partially fixed. In the above steps look at step 1. To match NI with its IL a bunch of information is required from the IL file. So if that IL file comes from the GAC then the IL file need not be opened to get those information. Hence no double loading happens. However, if the IL file comes from outside the GAC then it is indeed opened and kept open. This causes significant memory overhead in large applications. This is something which the CLR team needs to fix in the future.

Summary

1. Unsigned (non strong-named) assemblies can also be NGEN'd
2. Assemblies need not be placed in GAC to ngen them or to consume the ngen images
3. However, GAC'd files provide better startup performance and memory utilization while using NI images because it avoids double loading
4. NGEN captures enough metadata on an IL image to ensure that if its native image has become stale (no longer valid) it will reject the NI and just use the IL