

Exploring and Manipulating the GAC

Marudhamaran Gunasekaran

Contents

- Introduction**
- Why would you care about GAC?**
- Settings Things Straight**
- Structure of the GAC**
- Stripping the GAC Naked?**
- Deploying Assemblies to the GAC**
- GAC changes in .Net framework 4.0**
- What now?**
- References and Further Reading**

Introduction

The Global Assembly Cache (GAC) is a repository that contains assemblies intended to be shared across multiple applications on the running on the machine. This article focuses on the underlying structure of the GAC, managing assemblies in the GAC, ways to dissect the GAC, the changes GAC went through with .Net framework 4.0 and a bit of interesting things. The title might sound like the article's content is confined only to GAC; but to understand and work with the GAC we'd need a couple of other relative concepts of the CLR, assembly, etc., and I will try to discuss them as and when required. Some of the content is intended to provide an overview of the .Net framework intricacies and facts and basic understanding of the .Net framework is required.

Why would you care about GAC?

“Why would you care about GAC?” – You can ask yourself. The answer would be ‘I don't really care’ if you are writing a tool or library that does not want to expose its functionalities (types or methods or...) to other applications or if you care about ‘simple application deployment’ via *xcopy* or simple file copying via explorer. While having all referenced assemblies on the application's base directory (**Privately deployed Assemblies**) is a convenient way of deployment, when you build a class library that'd be consumed across applications on a machine (**Shared Assemblies**), GAC is what you

should be after. These two deployment types have their own pros and cons. So GAC is a special folder (?) and a well known location to the CLR where you can install assemblies when you want it to be shared by other applications on the machine. That's when you should care about GAC. Otherwise, private deployment is the way to go.

Setting Things Straight

If there are any misconceptions (perhaps more!) about the .Net framework versions, the CLR versions that shipped with it, the Base Class Library (BCL) versions, and the Visual Studio versions, let's get things straight. This section briefs the key points that'd serve as a basis for the subsequent sections. The below table summarizes the entities that we'd need to know before we move on. It's not required that you have to memorize it, pay attention to the .Net framework complete version and the CLR version shipped with it. This could be your ready-reckoner throughout the article.

.Net fw Public Version / Complete Version Number	CLR Public Version / Complete Version Number	Release Date	Visual Studio Version	Default in Windows
1.0 / 1.0.3705.0	v1.0 / v1.0.3705	2002-02-13	Visual Studio .Net	Windows XP Tablet and Media Center Editions
1.1 / 1.1.4322.573	v1.1 / v1.1.4322	2003-04-24	Visual Studio .Net 2003	Windows Server 2003
2.0 / 2.0.50727.42	v2.0 / v2.0.50727	2005-11-07	Visual Studio .Net 2005	Windows Server 2003 R2
3.0 / 3.0.4506.30	v2.0 / v2.0.50727	2006-11-06	-	Windows Vista, Windows Server 2008
3.5 / 3.5.21022.8	v2.0 / v2.0.50727	2007-11-19	Visual Studio .Net 2008	Windows 7, Windows Server 2008 R2
4.0 / 4.0.30319.1	v4.0 / v4.0.30319	2010-04-12	Visual Studio .Net 2010	<i>(May be Windows 8?)</i>

If someone needs more clarity on the beta version or the service pack versions, these links might suffice the need.

[List of .Net Framework Versions](#)

[What version of .Net framework is included in what version of the OS?](#)

A few more:

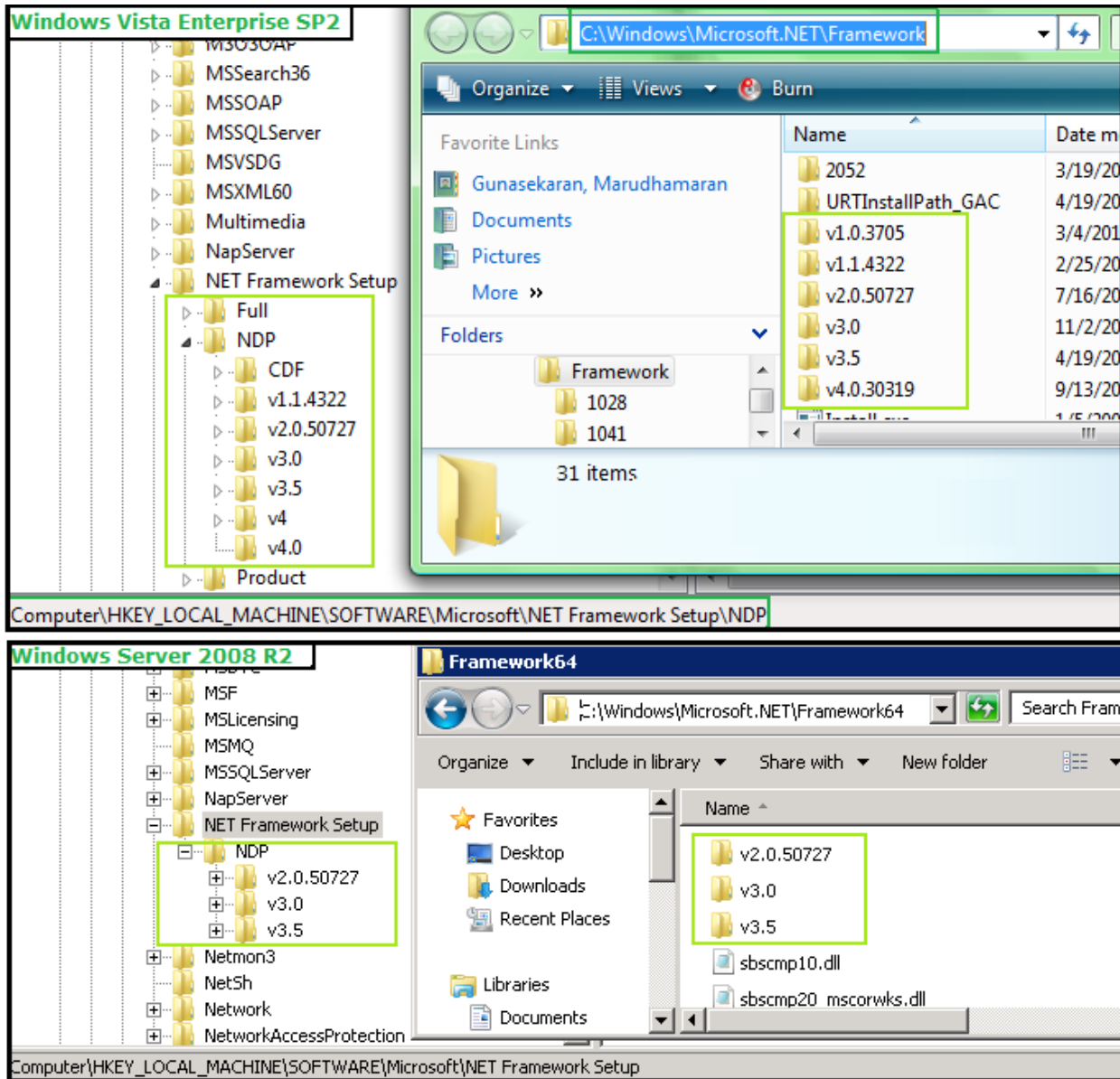
- The CLR version shipped with the .Net framework might be the same or different compared to the .Net framework version number (.Net fw v2.0, 3.0, 3.5 → CLR v2.0; .Net fw v4.0 → CLR v4.0)
- The Base Class Library assembly version does not necessarily have to be same as the Net framework version. While most of the BCLs are kept with their version numbers consistent with the .Net framework version, some BCLs have a different version in comparison to the framework version they shipped with.
- The physical location of the GAC in the file system is **C:\Windows\assembly** for .Net framework versions 2.0, 3.0, and 3.5 (CLR v2.0)
- The physical location of the GAC in the file system is **C:\Windows\Microsoft.NET\assembly** for .Net framework version 4.0 (CLR v4.0)
- **GAC Viewer (shfusion.dll)** which adorns the assembly folder and enables the drag and drop installation of an assembly to the GAC is obsolete starting with .Net framework version 4.0.
- **GacUtil.exe** is the recommended tool to be used to install/uninstall assemblies to the GAC.
- Normally, an assembly file (foo.dll for instance) contains only one assembly in it, but you might have multiple assemblies (and resource files) packed together in a single assembly file.
- The version information shown by Windows Explorer Properties dialog might not be the version of the assembly.

Identify the versions of .Net framework installed on the machine:

As we know that there could be multiple versions of the framework installed, here how you can find the versions of the .Net framework installed on a machine.

1. Navigate to the folder **C:\Windows\Microsoft.NET\Framework** and get the list of frameworks installed.
2. Alternatively, Navigate to the registry key **HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\NET Framework Setup\NDP** and examine the sub keys under it.

Here's a snapshot of the .Net frameworks installed on a Windows Vista Enterprise sp2 and a Windows Server 2008 R2.



Identify the versions of .Net framework installed on the machine:

So, different versions of the .Net framework ship with same/different version of the CLR. Targeting your assembly to the right framework version in Visual Studio (or using the right version of `csc.exe`) is important, and that takes care of emitting the right CLR version for the target framework. We will discuss how to examine the CLR version targeted by an assembly in the later part of the article.

Now to view the versions of the CLR installed on a machine, launch a VS 2010 Command Prompt

and type in `clrver`. Here's a snapshot of the CLR versions installed on my Windows Vista Enterprise sp2.

```
Administrator: Visual Studio Command Prompt (2010)
Setting environment for using Microsoft Visual Studio 2010 x86 tools.
c:\Program Files\Microsoft Visual Studio 10.0\VC>clrver
Versions installed on the machine:
v1.0.3705
v1.1.4322
v2.0.50727
v4.0.30319
c:\Program Files\Microsoft Visual Studio 10.0\VC>
```

Note: If you are using `clrver` in VS 2008 command prompt, then you won't be able to see no later versions of CLR than CLR v2.0. Using the right version of the VS tools and setting the right environment variables (which the VS command prompt automatically takes care of) is important. Some of the VS tools behave in a way that they shadow the newer versions (or they are incapable of being resilient to the newer versions of the framework)

Structure of the GAC

"GAC is a special folder" – Well, technically that's a statement you'd probably disagree if you understand the structure of the GAC. We've seen before that the physical path of the GAC is `C:\windows\assembly` (for .Net fw 2.0, 3.0, and 3.5). Open a Command prompt and change the working directory to `C:\windows\assembly` and try a `dir`. Here's a screenshot of that.

```
Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.0.6002]
Copyright (c) 2006 Microsoft Corporation. All rights reserved.

C:\Users\gunasema>cd C:\windows\assembly

C:\Windows\assembly>dir
Volume in drive C is PC COE
Volume Serial Number is 8EF0-DB5C

Directory of C:\Windows\assembly

06/08/2011  07:29 PM    <DIR>          GAC
04/25/2011  05:32 PM    <DIR>          GAC_32
09/14/2011  02:41 AM    <DIR>          GAC_MSIL
03/29/2011  03:00 PM    <DIR>          NativeImages1_v1.0.3705
03/04/2010  01:26 AM    <DIR>          NativeImages1_v1.1.4322
09/13/2011  01:32 PM    <DIR>          NativeImages_v2.0.50727_32
09/14/2011  12:44 AM    <DIR>          NativeImages_v4.0.30319_32
09/14/2011  02:41 AM    <DIR>          temp
09/14/2011  01:01 AM    <DIR>          tmp
                0 File(s)      0 bytes
                9 Dir(s)  67,292,164,096 bytes free

C:\Windows\assembly>
```

Looking at the screenshot above, it does not look like any special folder (like the windows

explorer displays – when you navigate to `C:\windows\assembly`). Windows explorer has a special view applied to this folder via the GAC viewer of the `shfusion.dll`. We will examine a couple of these subdirectories.

- **GAC** – contains the .Net 1.x assemblies.
- **GAC_32** – contains the assemblies built specifically for 32 bit Windows environment.
- **GAC_64** – contains the assemblies built specifically for 64 bit Windows environment. (not in the screenshot, you'd have to be in an x64 environment to see this folder)
- **GAC_MSIL** – contains the assemblies built CPU agnostic (they can run of both 32 & 64 bit Windows environment).
- **NativeImages_V####** - contains the native assemblies (**Ngen**-ed assemblies) targeting the specific CLR version represented by ####.

That's a bird's eye view of how the assembly is compartmentalized. If you try to drill down a folder, you'd see a folder with the `<<assemblyname>>`, further drilling down a folder, you'd see a folder with the `<<assemblyversion_publickeytoken>>`, and then finally the assembly file. Here's an example to view the `System.Data.dll` assembly.

```
C:\Windows\assembly>cd GAC_32\System.Data\2.0.0.0_b77a5c561934e089
C:\Windows\assembly\GAC_32\System.Data\2.0.0.0_b77a5c561934e089>dir
Volume in drive C is PC COE
Volume Serial Number is 8EF0-DB5C

Directory of C:\Windows\assembly\GAC_32\System.Data\2.0.0.0_b77a5c561934e089

34/11/2009  06:50 PM    <DIR>          .
34/11/2009  06:50 PM    <DIR>          -
34/11/2009  06:45 PM             2,933,760 System.Data.dll
               1 File(s)          2,933,760 bytes
               2 Dir(s)    67,289,100,288 bytes free

C:\Windows\assembly\GAC_32\System.Data\2.0.0.0_b77a5c561934e089>
```

```
cd C:\windows\assembly\GAC_32\System.Data\2.0.0.0_b77a5c561934e089
dir
```

These folder structures are created for you dynamically when you use `gacutil` or `msi` (and of course during the .Net framework installation). Creating a folder structure like this manually to install an assembly is strictly not recommended. Instead you should use the dedicated tools. We will discuss this in later sections.

Though the GAC is mapped to standard aforementioned folders, you have the flexibility to change the default GAC location, if you that is requirement (and if you really want to do that).

Note: If you have two companies developing different versions of foo.dll, obviously, they both can't reside on the same folder, one might get replaced by the other, breaking the application. Since dedicated structures are created in the GAC for each assemblies, no two assemblies with same name gets replaced (as long as they have different assembly name, version, culture, public key token combination)

Stripping the GAC Naked?

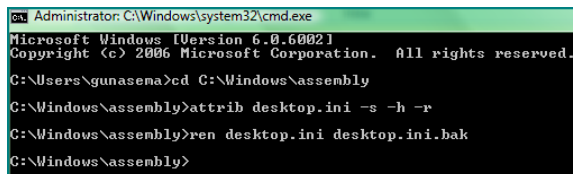
Now that we know that the GAC is just like any other folder in the file system, let's look at the techniques through which we can view it naked (just like a normal file/folder in windows explorer).

Sometimes we might want to copy an assembly from the GAC, or we might want to add a direct reference to a GAC assembly for your project in VS (there are times at which you'd end up with an assembly with no source code, and the only location you can find it is the GAC, and the VS 'Add Reference' dialog box is not guaranteed to display all the assemblies deployed to the GAC either). That's why we might want to do this step (apart from a learning perspective).

As I have mentioned earlier **shfusion.dll** is responsible for the GAC's pretty face in explorer and that is taken care by the **desktop.ini** file located at the same folder. OK! That's that, time to go to the flipside.

Rename the Desktop.ini file:

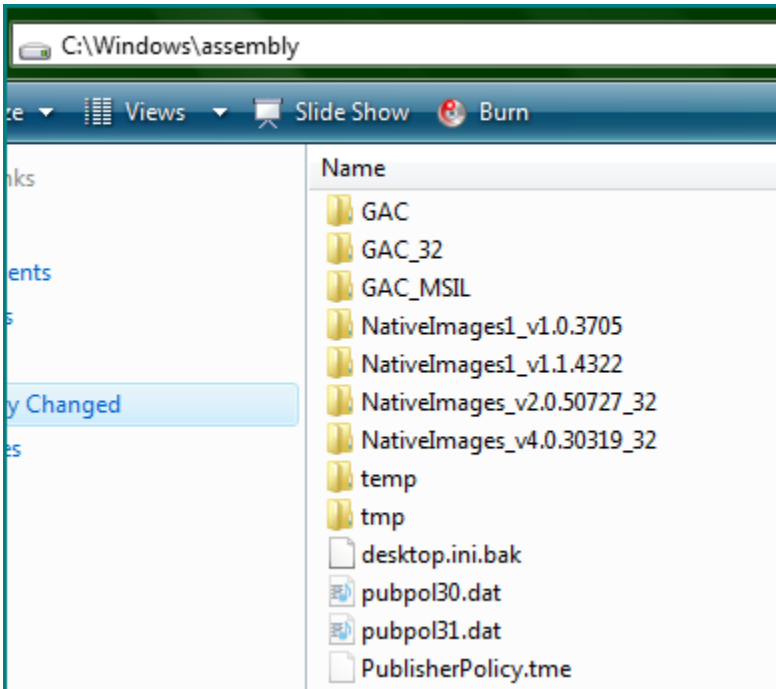
Open a command prompt and type the following commands in order.



```
Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.0.6002]
Copyright (c) 2006 Microsoft Corporation. All rights reserved.
C:\Users\gunasena>cd C:\Windows\assembly
C:\Windows\assembly>attrib desktop.ini -s -h -r
C:\Windows\assembly>ren desktop.ini desktop.ini.bak
C:\Windows\assembly>
```

```
cd C:\Windows\assembly
attrib desktop.ini -s -h -r
ren desktop.ini desktop.ini.bak
```

Now Navigate to *C:\Windows\assembly* via Windows Explorer and see if for yourself. Here's a screenshot.



To revert the changes, type the following commands in order

```
Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.0.6002]
Copyright (c) 2006 Microsoft Corporation. All rights reserved.

C:\Users\gunasema>cd c:\windows\assembly
c:\Windows\assembly>ren desktop.ini.bak desktop.ini
c:\Windows\assembly>attrib desktop.ini +s +h +r
c:\Windows\assembly>
```

```
cd c:\windows\assembly
ren desktop.ini.bak desktop.ini
attrib desktop.ini +s +h +r
```

Use substitute command:

This is perhaps the easy way. Run the following commands in order and open my computer to view another drive that's been created (that's reference the GAC folder).

Open a command prompt. Type in

```
subst z: c:\windows\assembly
```

[Note: 'z:' should be a driver letter that is not currently used by any network shares or other Drives/partitions. You can use any available drive letter]

To revert, type in


```
subst /d z:
```

This deletes the driver letter z: mapped to `c:\windows\assembly`

Unregister shfusion.dll

Unregister the shfusion.dll using regsvr32.

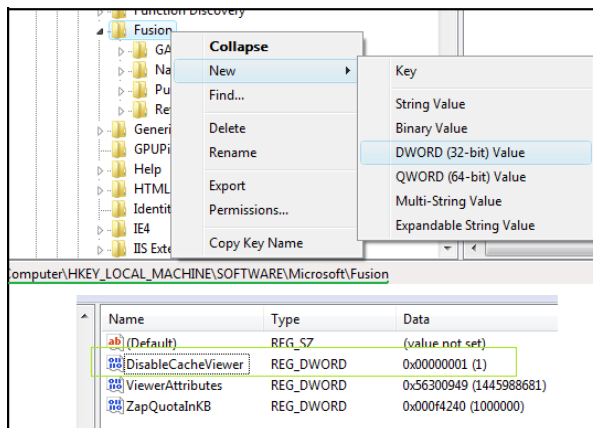
Start → Run → `regsvr32 -u "C:\Windows\Microsoft.NET\Framework\v2.0.50727\shfusion.dll"`

To revert,

Start → run → `regsvr32 -I "C:\Windows\Microsoft.NET\Framework\v2.0.50727\shfusion.dll"`

Edit registry:

Navigate to the registry (start→Run→`regedit`) key at `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Fusion` and add a new **DWORD** named **DisableCacheViewer** with the value set to **1**



Note: Take a look at [How to display an assembly in the "Add Reference" dialog box](#)

Deploying Assemblies to the GAC

Well, having understood the concepts above, we should be able to deploy an assembly to the GAC. There are various techniques to achieve this. I'd discuss the two widely used and simple techniques and leave the rest of them in the hyperlinks.

Is your assembly Strongly named?

Before we move on; “All assemblies that are meant for GAC deployment or assemblies that you are building for a product should be strongly named which helps the CLR to uniquely identify an assembly besides a couple of other verifications like checksum for tamper-resistance”.

Strong naming an assembly means, you sign the assembly with a ‘public-private key pair’. There are a couple of algorithms to generate this public-private key pair and the hash for ‘public key token’. All you have to know at this point of time is that when you sign an assembly, the assembly gets embedded with a ‘public-private key pair’, and then obtains a 64 bit hash of the public key which becomes the public key token for that assembly. This process makes an assembly strong named. And the **Assembly Name**, the **Assembly Version**, **Assembly Culture**, **Assembly Public Key Token** together uniquely identifies the assembly.

In VS, Right click on the Project properties, Hit **signing** → **Sign the Assembly** and select a key file or generate a new key file. And when you build the project, the assembly in the project gets signed and thus becomes strongly named.

Sn.exe is responsible for generating the public-private key pair. You can manually generate an public-private key pair and then sign the assembly using **csc.exe**.

Using GacUtil.exe

First things first, Here’s how you use gacutil.

```
Gacutil -i "C:\Projects\Foo\bar.dll"  
Gacutil -u "bar, Version=1.0.0.0, Culture=neutral, PublicKeyToken=b4567au8a2hu223"  
Gacutil -L
```

As the above command line usage might sound obvious, let’s discuss it.

`Gacutil -i "C:\Projects\Foo\bar.dll"` → installs assembly named bar located at `C:\Projects\Foo`

`Gacutil -u "bar, Version=1.0.0.0, Culture=neutral, PublicKeyToken=b4567au8a2hu223"` → uninstalls an assembly named bar having Version=1.0.0.0, Culture=neutral, and PublicKeyToken=b4567au8a2hu223.

This is the way to uninstall an assembly from the GAC. If you just pass a partial name of the assembly like `gacutil -u bar`, then that would uninstall all the assemblies matching the name bar irrespective of the Version, Culture and PublicKeyToken. You don’t want that happening especially if you have multiple versions of an assembly installed to the GAC.

That’s the warm up. The above ways of deploying an assembly to the GAC would work

beautifully if you have VS or the Windows SDK installed on the machine. Most of the times, the production servers would not have VS, neither the Windows SDK. A couple of ways to handle this scenario would be to provide an MSI that would install the assembly to GAC (perhaps doing other things that you programmed the msi for) or you can copy the gacutil.exe from the Windows SDK's/VS's physical location to the target production environment and use a batch file to invoke it. The important thing if you resort to the later technique is to identify the right version of the gacutil.exe to use. We will discuss that shortly.

One more guideline when you use gacutil in production environment is to specify the `/r` switch like below

```
Gacutil -i -r bar.dll UNINSTALL_KEY FooBarApp "This is a FooBar Application"
```

The above statement associates the assembly 'bar' to the Windows Install and Uninstall Engine. It tells the computer that 'FooBarApp' requires the assembly bar.dll.

```
Gacutil -i -r bar.dll FILEPATH "C:\Projects\Foo\barapp.exe" BarApp  
Gacutil -i -r bar.dll FILEPATH "C:\Projects\Greet\greet.exe" Greet
```

The above two commands associates the assembly 'bar' with the two applications named 'BarApp' And 'Greet' located at their respective locations.

```
Gacutil -u -r bar.dll FILEPATH "C:\Projects\Greet\greet.exe" Greet
```

The above command removes the installer reference of Greet from bar.dll assembly. If there are installer references for bar.dll assembly, then gacutil is faithful enough to not uninstall bar.dll from the GAC, ensuring the other application that consume this bar.dll are not left broken. (If the above uninstall command is executed and if that Greet was the last reference for that assembly bar.dll, then the assembly is uninstalled from the GAC). More? Here [Gacutil.exe](#)

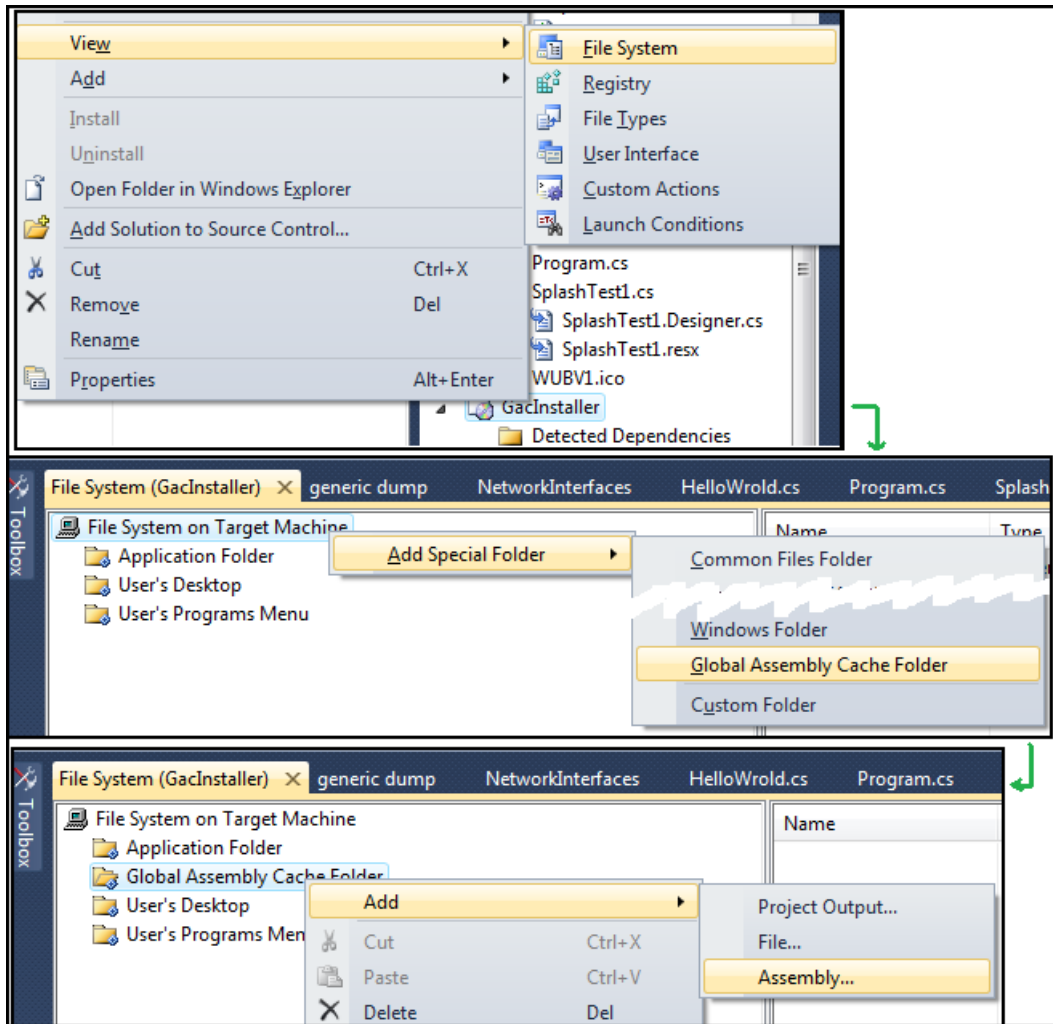
Using MSI:

Let's open up VS 2010. Create a new set up project, let's name it Gaclnstaller. Right click the Gaclnstaller project in the **Solution Explorer** and select **View** → **File System**.

Right click on **File System on Target Machine** and select **Add Special Folder** → **Global Assembly Cache Folder**.

Right click on **Global Assembly Cache Folder** and select **Add** → **Assembly**.

Here's you are free to add any .Net assemblies (as long as they are strong named) targeted for any version of .Net framework. The msi built eventually takes care of deploying them in the GAC (in their respective folders – `C:\Windows\assembly\GAC` for CLR v1.1; `C:\Windows\assembly\GAC_32` for CLR v2.0 x86; `C:\Windows\Microsoft.Net\assembly` for CLR V4.0).



Once you are done adding the assemblies, build the GacInstaller project, fetch the MSI and try installing it in any machine. If you want to check if the assemblies are successfully placed in the GAC, open VS 2010 command prompt and run

```
gacutil -l
```

(Or you can navigate to the above mentioned folders in command prompt or windows explorer to view them)

Remember, if you had to do this deployment of multiple assemblies targeting different CLR versions via a batch file, then you have to identify the respective gacutil.exe from the file system and then use the appropriate gacutil.exe against the appropriate assemblies. Or if you have the latest version of VS (2010) or Windows SDK (7.5) then you can pick that **gacutil.exe** and use it across all

assemblies built for different CLR versions.

Still if you prefer the use this GacInstaller.msi on a batch file, make use of the **/quiet** or **/passive** switches provided by **msiexec.exe**.

Other ways to deploy assemblies to the GAC. (Just a couple of them below, not a complete list.)

[Installing Assemblies for Runtime and Design-time Use](#)

[How can I install a file to the GAC and the local file system using WiX syntax?](#)

[Sample Managed GAC API Wrappers](#)

GAC changes in .Net framework 4.0

We have seen before that .Net framework 4.0 shipped with CLR v4.0. What this means is that, if you build assemblies for .Net fw 4.0, they are eventually targeted to be run by CLR v4.0. If you want to install this assembly targeting the .Net fw 4.0, you should be better off using the gacutil in VS 2010 command prompt.

Now, GAC has its new location for assemblies targeted for .Net framework 4.0. The new GAC location is *C:\Windows\Microsoft.Net\Assembly*. And the *GAC Viewer (shfusion.dll)* is obsolete starting with .Net framework 4.0.

What about Drag and Drop?

Gone! The drop and drop assembly installation was facilitated by shfusion.dll, which should be considered obsolete now. And it is pretty easy to install an assembly to the GAC by simply dragging and dropping to the assembly folder. (I know of a couple of bugs that shfusion.dll has, may be that is the reason why shfusion.dll is discontinued). So there is no drag and drop assembly installation for .Net 4.0 assemblies. If you try to drop a .Net 4.0 assembly in to the GAC, you won't be shown an error; you won't be able to see the assembly in the GAC either. It just won't go there and get installed! If you try to drag and drop a .Net 4.0 assembly assembly to the new GAC at *C:\windows\Microsoft.Net\assembly*, the assembly will be copied but that makes no difference (the assembly is not installed to the new GAC, because the GAC structure expects the subdirectories with version numbers and public key token – we have seen this before in the Structure of the GAC section)

Why do we need another GAC anyway?

If you consider the way GAC organized structurally, and the way an assembly is differentiated from another assembly by the Assembly Name, Version, Culture, public key token - (to avoid DllHell) - you would not need another GAC. But the information from communities and MSDN outlines a problem.

In GAC, if you have Foo.dll (assembly version 1.0.0.0) targeted for CLR v.2.0, and another Foo.dll (assembly version 2.0.0.0) targeted for CLR v.4.0, there exists a scenario when an application expecting the Foo.dll V 1.0.0.0 might load the newer version available Foo.dll v 2.0.0.0 and thus might result in unexpected behavior. So the assembly is split for each runtime. ^[3]

You can view the CLR version of an assembly, using **corflags.exe**.

```
c:\Program Files\Microsoft Visual Studio 10.0\VC>corflags "C:\Maran\Foo\bin\Debug\Foo.dll"
Microsoft (R) .NET Framework CorFlags Conversion Tool. Version 4.0.30319.1
Copyright (c) Microsoft Corporation. All rights reserved.

Version : v4.0.30319
CLR Header: 2.5
PE : PE32
CorFlags : 9
ILOONLY : 1
32BIT : 0
Signed : 1

c:\Program Files\Microsoft Visual Studio 10.0\VC>corflags "C:\Maran\ENSS\web_ENSS_Demo\Bin\ENSS_a2.dll"
Microsoft (R) .NET Framework CorFlags Conversion Tool. Version 4.0.30319.1
Copyright (c) Microsoft Corporation. All rights reserved.

Version : v2.0.50727
CLR Header: 2.5
PE : PE32
CorFlags : 9
ILOONLY : 1
32BIT : 0
Signed : 1
```

Or you can use **ildasm.exe** and view the Version in the manifest tables. Or examine the CLR Header using **dumpbin.exe**. [What CLR Version?](#)

Finding the right gacutil:

(The information in this section is useful when VS or Windows SDK is not available in the production environment or you don't choose to use the VS command prompt for some reason or you choose to install assemblies to the GAC using a batch file)

With various .Net framework installations and VS installations side-by-side, you'd have to get a clear idea of the CLR version that you are targeting. Microsoft stopped shipping gacutil.exe as a part of the .Net framework (.net framework 1.1 shipped with gacutil). Now gacutil is a part of VS and Windows SDK.

For assemblies targeting .Net fw 2.0, 3.0, and 3.5 (CLR v2.0), use the gacutil at

<C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\Bin\gacutil.exe>

Or

<C:\Program Files\Microsoft SDKs\Windows\v6.0A\bin\gacutil.exe>

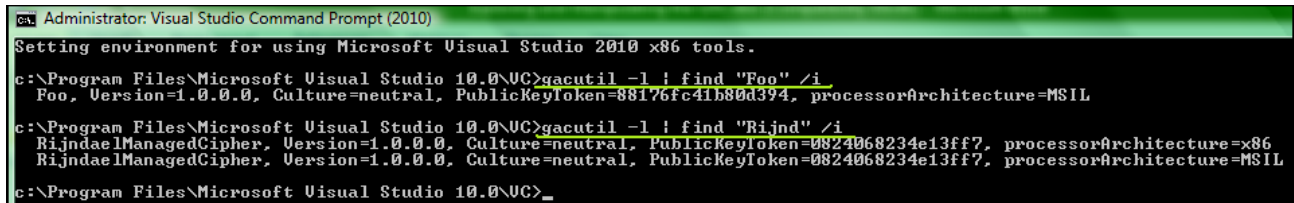
For assemblies targeting .Net fw 4.0 (CLR v4.0) or earlier versions (1.0, 1.1, 3.0, 3.5) use the gacutil at

<C:\Program Files\Microsoft SDKs\Windows\v7.0A\bin\gacutil.exe>

Search for an assembly in the GAC:

So with the *GACViewer (shfusion.dll)* obsolete, if you are looking for ways to quickly find if an assembly is installed to the GAC, you can use the `gacutil -l` switch and filter the output by a find (with ignore case switch)

Here's how to do that. `gacutil -l | find "<<assemblyname>> /I`



```
Administrator: Visual Studio Command Prompt (2010)
Setting environment for using Microsoft Visual Studio 2010 x86 tools.
c:\Program Files\Microsoft Visual Studio 10.0\UC>gacutil -l | find "Foo" /i
Foo, Version=1.0.0.0, Culture=neutral, PublicKeyToken=88176fc41b80d394, processorArchitecture=MSIL
c:\Program Files\Microsoft Visual Studio 10.0\UC>gacutil -l | find "Rijnd" /i
RijndaelManagedCipher, Version=1.0.0.0, Culture=neutral, PublicKeyToken=0824068234e13ff7, processorArchitecture=x86
RijndaelManagedCipher, Version=1.0.0.0, Culture=neutral, PublicKeyToken=0824068234e13ff7, processorArchitecture=MSIL
c:\Program Files\Microsoft Visual Studio 10.0\UC>_
```

Important:

If you use `gacutil` with the VS2005 (which internally uses `C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\Bin\gacutil.exe`) or VS2008 (which internally uses `C:\Program Files\Microsoft SDKs\Windows\v6.0A\bin\gacutil.exe`) command prompt, then you won't be able to see the assemblies that are targeted for the .Net fw 4.0 (the assemblies installed at the location `C:\windows\Microsoft.Net\assembly`). The reason being... <<identify yourself>>.

So if you are writing apps targeting .Net framework 4.0, then you should (must) use VS 2010 command prompt. For deployment, you should pick the `gacutil` under the folder `C:\Program Files\Microsoft SDKs\Windows\v7.0A\bin\gacutil.exe`.

[If you are still wondering the reason, the `gacutil.exe` shipped before .Net fw 4.0 knows GAC as a folder located at `C:\Windows\assembly` and it can't look/search/traverse any other folders. But the `gacutil` shipped with .Net fw 4.0 knows `C:\windows\Microsoft.Net` as well as `C:\Windows\assembly` and thus handles the assembly installation process appropriately].

What now?

That's pretty much it. Those are some of the phenomena of GAC, .Net framework and its associated tools. Try to apply them at work. Some of the topics are left intentionally vague due to lack of time and space. If you find any deviation in the content, please bring it to my attention. I can be reached at <http://renouncedthoughts.wordpress.com/contact/>.

References and Further Reading

1. CLR via C# 2rd edition and 3rd edition, by Jeffrey Richter
2. Inside C# 2nd edition, by Tom Archer, Andrew Whitechapel
3. [Understanding the CLR Binder](http://msdn.microsoft.com/en-us/magazine/dd727509.aspx), by Aarthi Ramamurthy and Mark Miller - <http://msdn.microsoft.com/en-us/magazine/dd727509.aspx>

4. [Global Assembly Cache](http://msdn.microsoft.com/en-us/library/yf1d93sz.aspx) - <http://msdn.microsoft.com/en-us/library/yf1d93sz.aspx>
5. [Global Assembly Cache Tool](http://msdn.microsoft.com/en-us/library/ex0ss12c.aspx) - <http://msdn.microsoft.com/en-us/library/ex0ss12c.aspx>
6. [Assembly Cache Viewer](http://msdn.microsoft.com/en-us/library/34149zk3.aspx) - <http://msdn.microsoft.com/en-us/library/34149zk3.aspx>
7. [Assembly.LoadWithPartialName Method](http://msdn.microsoft.com/en-us/library/system.reflection.assembly.loadwithpartialname.aspx) - <http://msdn.microsoft.com/en-us/library/system.reflection.assembly.loadwithpartialname.aspx>
8. [Strong Name Tool](http://msdn.microsoft.com/en-us/library/k5b5tt23(v=VS.100).aspx) - [http://msdn.microsoft.com/en-us/library/k5b5tt23\(v=VS.100\).aspx](http://msdn.microsoft.com/en-us/library/k5b5tt23(v=VS.100).aspx)