

Version Policy in .Net

Agenda

- Version Policy
- Resolving Names to Locations

Assembly Version Number

- Each assembly has a four-part version number as part of its identity
 - that is, version 1.0.0.0 of some assembly and version 2.1.0.2 are completely different identities as far as the class loader is concerned.
- Including the version as part of the identity is essential to distinguish different versions of an assembly for the purposes of side-by-side execution.
- There are no semantics applied to the parts of the version number!
 - That is, the CLR does not infer compatibility or any other characteristic of an assembly based on how the version number is assigned.
- As a developer you are free to change any portion of this number as you see fit.
- Even though there are no semantics applied to the format of the version number, individual organizations will likely find it useful to establish conventions around how the version number is changed.

The parts of the Assembly version number

- The parts of the version number are:

2.5.719.2

major.minor.build.revision

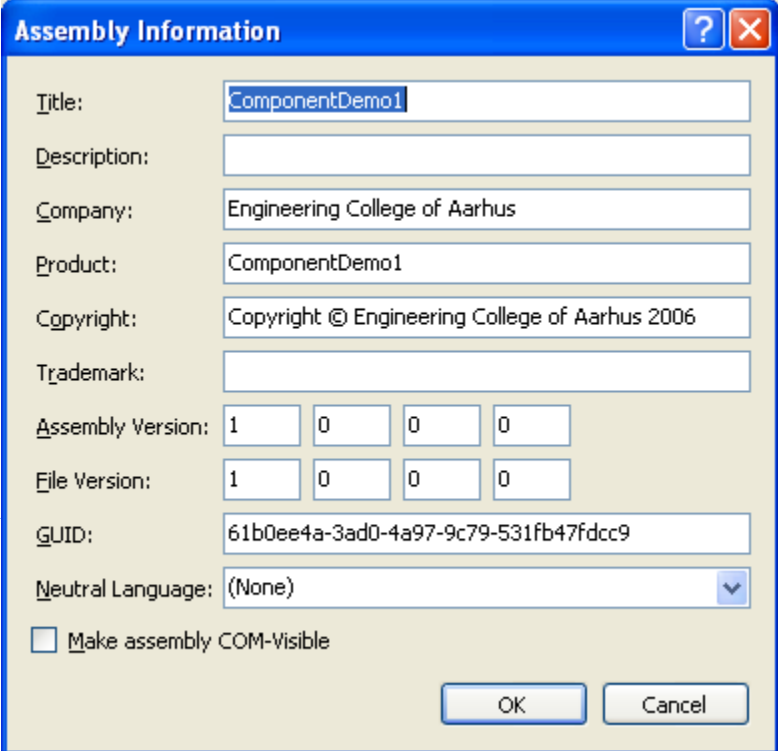
- **Major or minor.** Changes to the major or minor portion of the version number indicate an incompatible change.
 - Under this convention then, version 2.0.0.0 would be considered incompatible with version 1.0.0.0.
 - Examples of an incompatible change would be a change to the types of some method parameters or the removal of a type or method altogether.
- **Build.** The Build number is typically used to distinguish between daily builds or smaller compatible releases.
- **Revision.** Changes to the revision number are typically reserved for an incremental build needed to fix a particular bug. You'll sometimes hear this referred to as the "emergency bug fix" number in that the revision is what is often changed when a fix to a specific bug is shipped to a customer.

Entering Version Number

During development the version number is kept in the file AssemblyInfo.cs (created by the Wizard).

But typically you enter the number by use of the assembly properties, and open the application tab. Click the Assembly information button to display this dialog


You can get VS to update the build and revision part automatically by entering “”*



```
// AssemblyInfo.cs
using System.Reflection;

...
[assembly: AssemblyVersion("1.0.0.0")]
[assembly: AssemblyFileVersion("1.0.0.0")]
```

Assembly Deployment Models

- .Net support two assembly deployment models:
- **Private assemblies:**
 - Each application has its own local copy of the assembly.
 - Private assemblies must be in same folder as the exe-file (or in a sub folder if probing is applied).
- **Shared assemblies:**
 - Can be used by multiple applications.
 - Must be installed in a special folder called the **global assembly cache (GAC)**, which have this path: C:\WINDOWS\assembly on a PC running Windows.
 - Must have a ***Strong name*** 

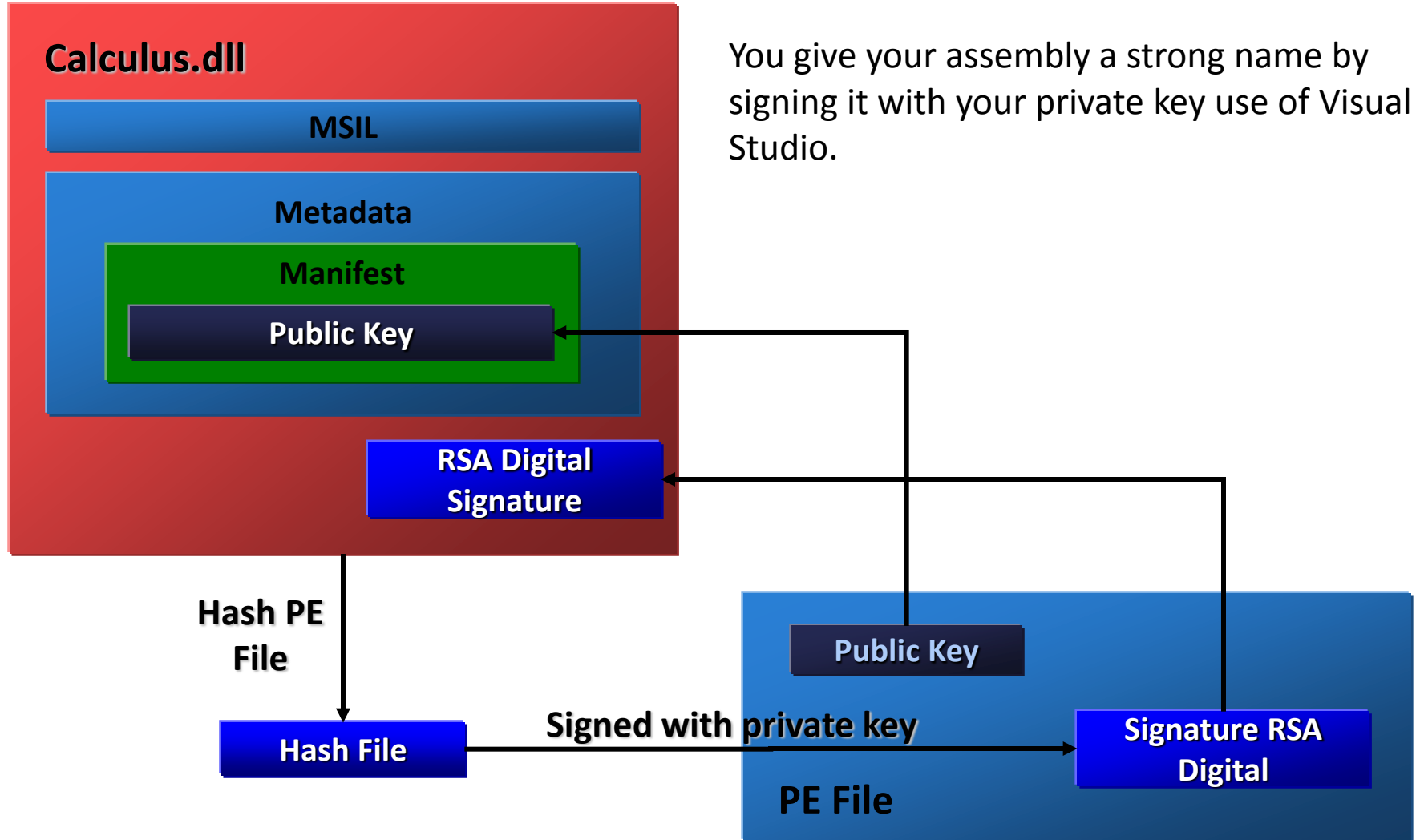
Assembly Names

- Each assembly has a four-part name that **uniquely** identifies it:
friendly name=MyAssembly, **Version**=1.2.3.4, **Culture**=en-US, **PublicKeyToken**=a169ca1231b23456
- The four parts are:
 - A friendly name
 - This is the name of the file with the assembly manifest
 - Version
 - All assemblies has a four-part version number. If you don't set it, the compiler will set in to 0.0.0.0
 - Culture
 - Two-part string indicating spoken language and region as specified in RFC 1766. For non localized assemblies the culture is set to neutral
 - Public Key
 - **Is used to sign the assembly with a strong name.**
 - Is set to null if not used.
 - This key indicate the producer of this assembly.

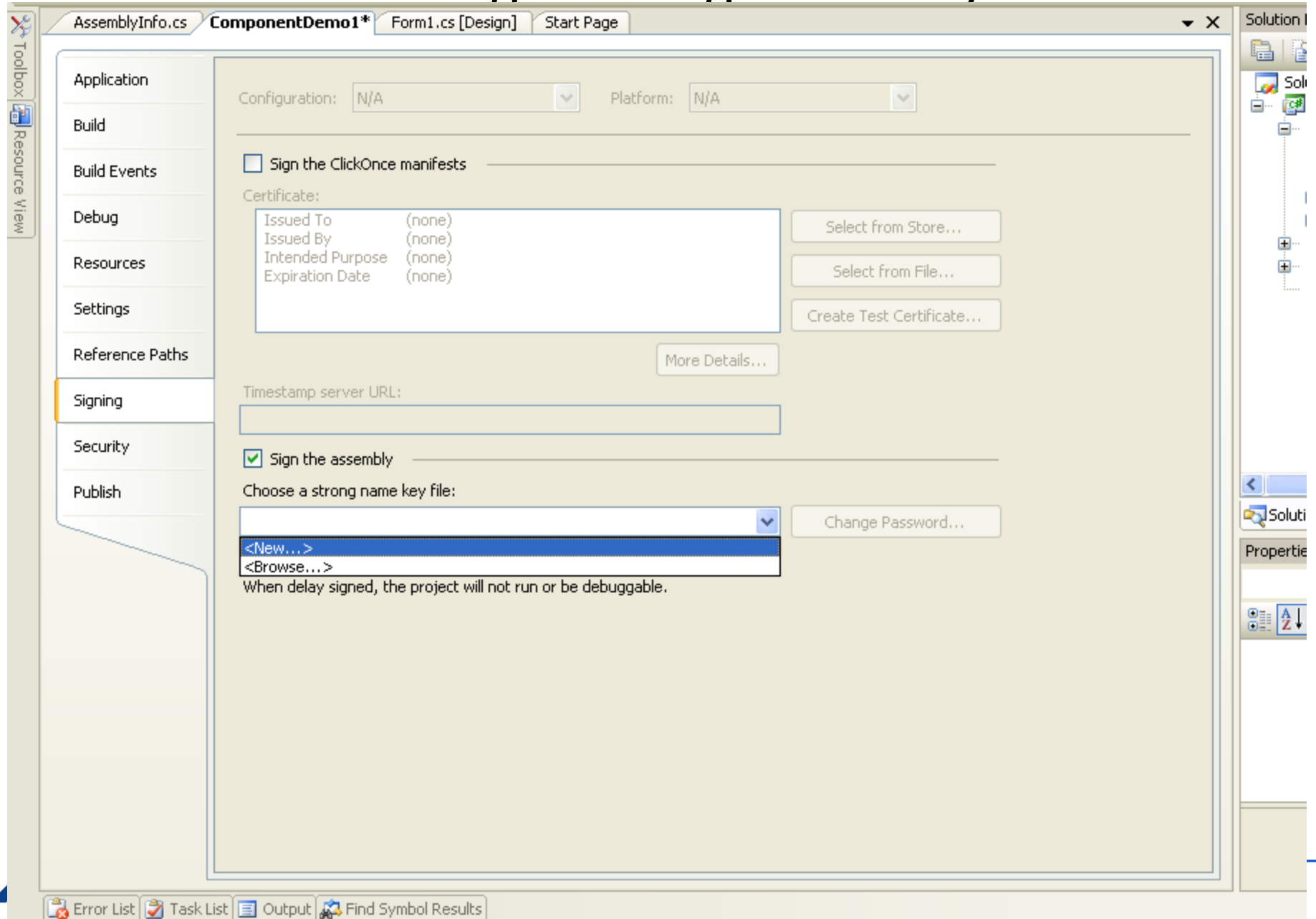
Strong Names

- Simple name accompanied by:
 - Public key != null
 - Digital signature
 - Generated from assembly (hash of file) and private key
- To give your assembly a strong name you **Sign** your assembly
 - se next slide.
- Prevent others from „taking over your namespace“.
- Protect version lineage.
- Assemblies with same strong name are identical!
- **Versioning only works with strong named assemblies!**

Signing Assemblies



Generating a strong name key file



Delaying Strong Name Assignment

- Access to private key might be restricted
- Delayed (or partial) signing reserves space in file
 - Actual signing is deferred
- Process works as follows:
 - Developer works with public key file only
[assembly:AssemblyKeyFile("pubKey.snk")]
[assembly:AssemblyDelaySign(true)]
 - Verification must be switched off
sn -Vr myAssembly.dll
 - Full signing must be applied later
sn -R myAssembly.dll fullKey.snk

Culture

- What is Culture?
 - Used to develop similar assemblies for different languages
 - Identified via a string that contains a primary and secondary tag
 - If no culture string is explicitly assigned, the assembly is considered “Culture-Neutral”
- Example

```
// Set assembly's culture to Swiss German
[assembly:AssemblyCulture("de-ch")]
```

Version Numbers

- What Is Versioning All About?
 - **Runtime only applies version policy to strongly named assemblies**
 - Runtime uses whatever private assemblies it can find, regardless of its version information
- Example

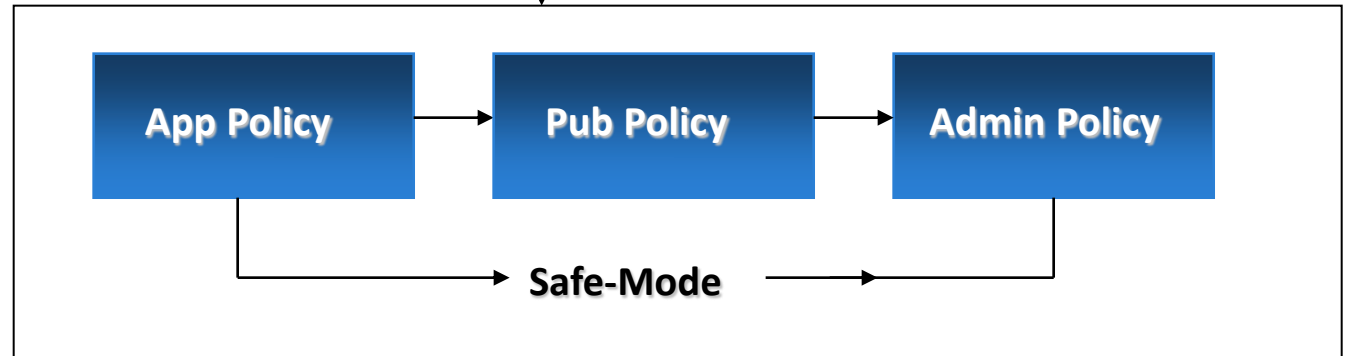
2.5.719.2
major.minor.build.revision

Versioning overview

1) Original
Assembly
Reference

Ex: foo,Ver=1.0.0.0, PK= 23 43 ...

2) Apply
Policy



3) Post-Policy
Reference

Ex: foo,Ver=2.0.2.0, PK= 23 43 ...

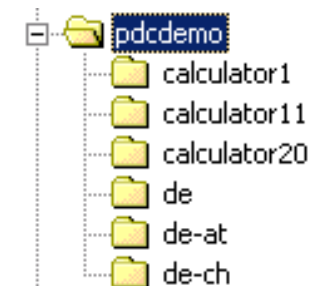
4) Find
Assembly

Global Assembly Cache

App Directory

The screenshot shows a Windows Explorer window with the address bar set to 'OSDisk (C:) > Windows > assembly'. The main pane displays a table of assemblies in the Global Assembly Cache.

Assembly Name	Version	Culture	Public Key Token	Processor Architecture
Microsoft.Office.Tools.v9.0.resources	9.0.0.0	da	b03f5f7f11d50a3a	MSIL
Microsoft.Office.Tools.Word.v9.0	9.0.0.0		b03f5f7f11d50a3a	MSIL
Microsoft.Office.Tools.Word.v9.0.resources	9.0.0.0	da	b03f5f7f11d50a3a	MSIL



Sharing, Culture and Type References

App.exe

AssemblyRef 1:

Name: Calculus.dll

Version: 1.2.3.4

Culture: ""

PublicKeyToken: 22acab57c8682eac

AssemblyRef 2:

Name: AdvMath.dll (private assembly)

Calculus.dll

Version: 1.2.3.4

Culture: ""

PublicKey: 22acab57c8682eac

AdvMath.dll

AssemblyRef 1:

Name: Calculus.dll

Version: 2.0.0.0

Culture: ""

PublicKeyToken: 22acab57c8682eac

Calculus.dll

Version: 2.0.0.0

Culture: ""

PublicKey: 22acab57c8682eac

Side-By-Side Execution of different versions of the same assembly is possible - even in the same process.

Default Assembly Version Policy

- When resolving references to **strong named** assemblies, the CLR determines which version of the dependency to load when it comes across a reference to that assembly in code.
- The default version policy in .NET for shared assemblies is extremely straightforward:
 - When resolving a reference, the CLR takes the version from the calling assembly's manifest and loads the version of the dependency with the **exact same version number**.
 - In this way, the caller gets the exact version that he was built and tested against.
- This default policy has the property that it protects applications from the scenario where a different application installs a new version of a shared assembly that an existing application depends on.
- *For private assemblies NO version policy is applied!*

Custom Version Policy

- There may be times when binding to the exact version the application was shipped with isn't what you want.
- For example, an administrator may deploy a critical bug fix to a shared assembly and want all applications to use this new version regardless of which version they were built with.
- Also, the vendor of a shared assembly may have shipped a service release to an existing assembly and would like all applications to begin using the service release instead of the original version.
- These scenarios and others are supported in the .NET Framework through version policies.

Version Policies Files

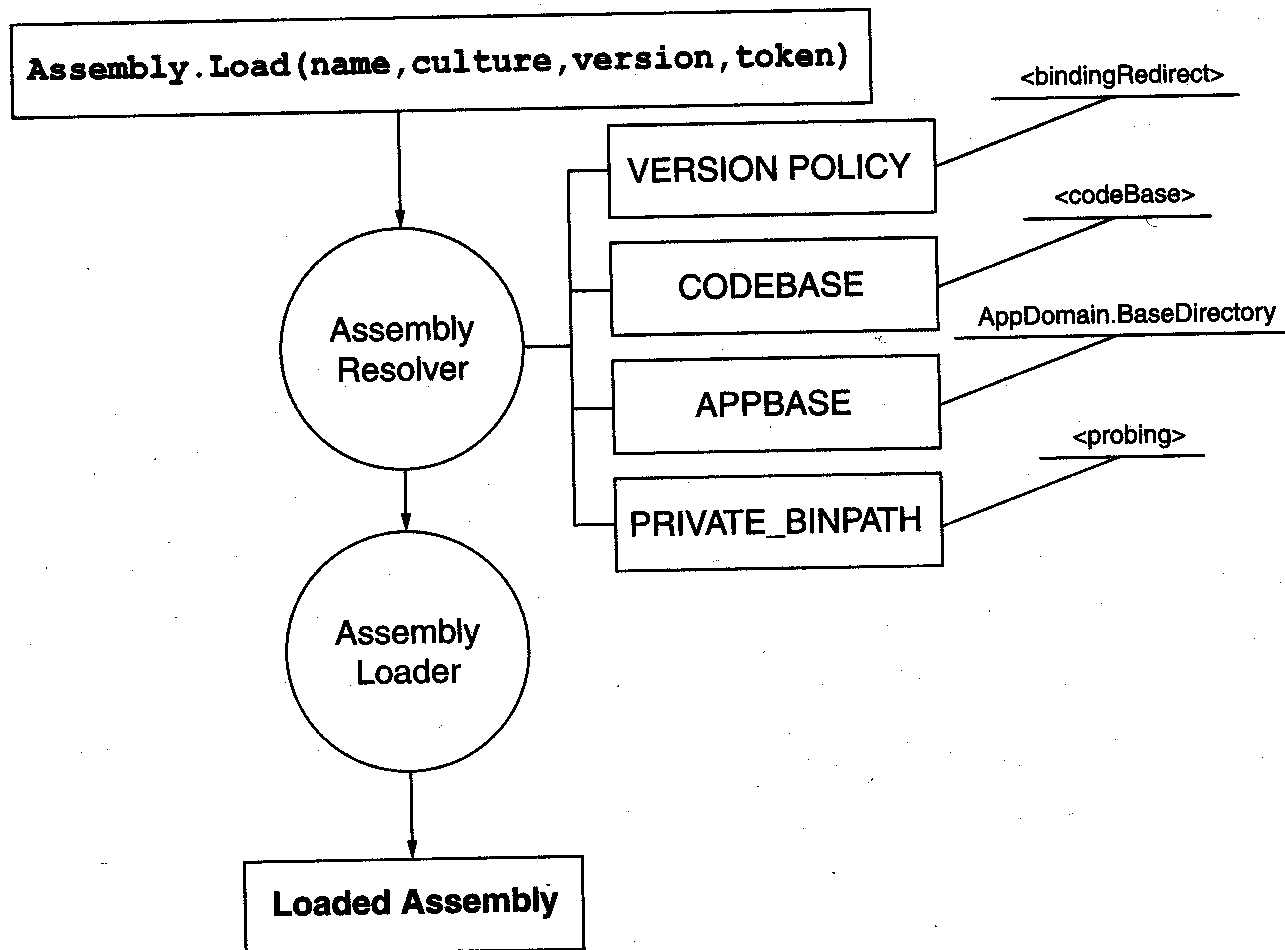
- Version policies are stated in XML files and are simply a request to load one version of an assembly instead of another.
 - For example, the following version policy directs the CLR to load version 5.0.0.1 instead of version 5.0.0.0 of an assembly called MarineCtrl:

```
<assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
  <dependentAssembly>
    <assemblyIdentity name="MarineCtrl" publicKeyToken="9335a2124541cfb9" />
    <bindingRedirect oldVersion="5.0.0.0" newVersion="5.0.0.1" />
  </dependentAssembly>
</assemblyBinding>
```

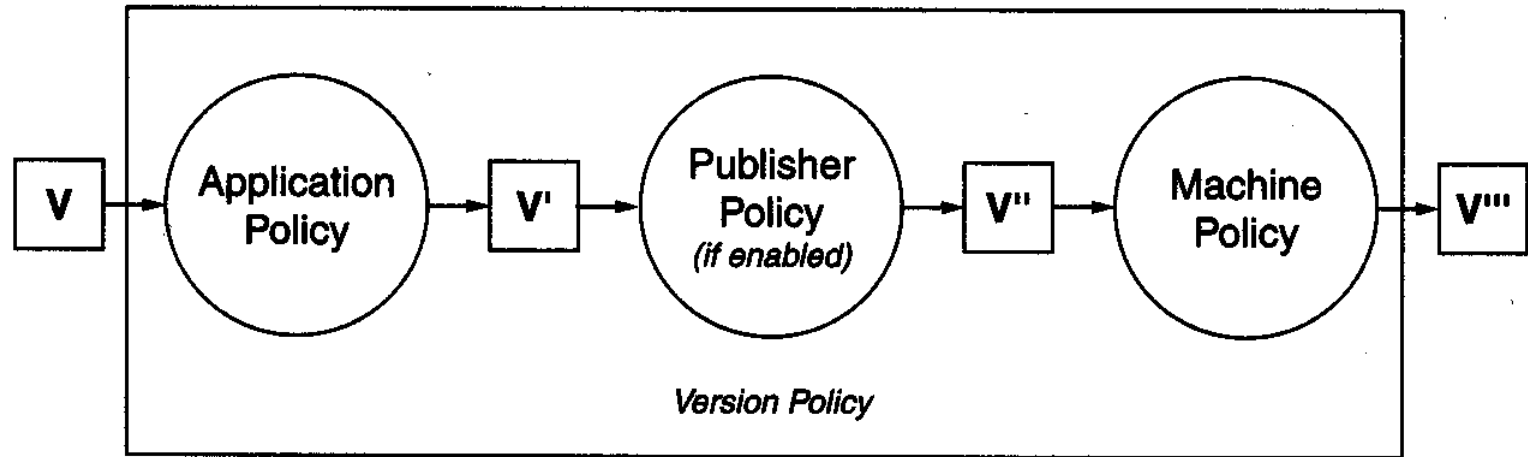
- In addition to redirecting from a specific version number to another, you can also redirect from a range of versions to another version.
 - For example, the following policy redirects all versions from 0.0.0.0 through 5.0.0.0 of MarineCtrl to version 5.0.0.1:

```
...
<bindingRedirect oldVersion="0.0.0.0-5.0.0.0" newVersion="5.0.0.1" />
...
```

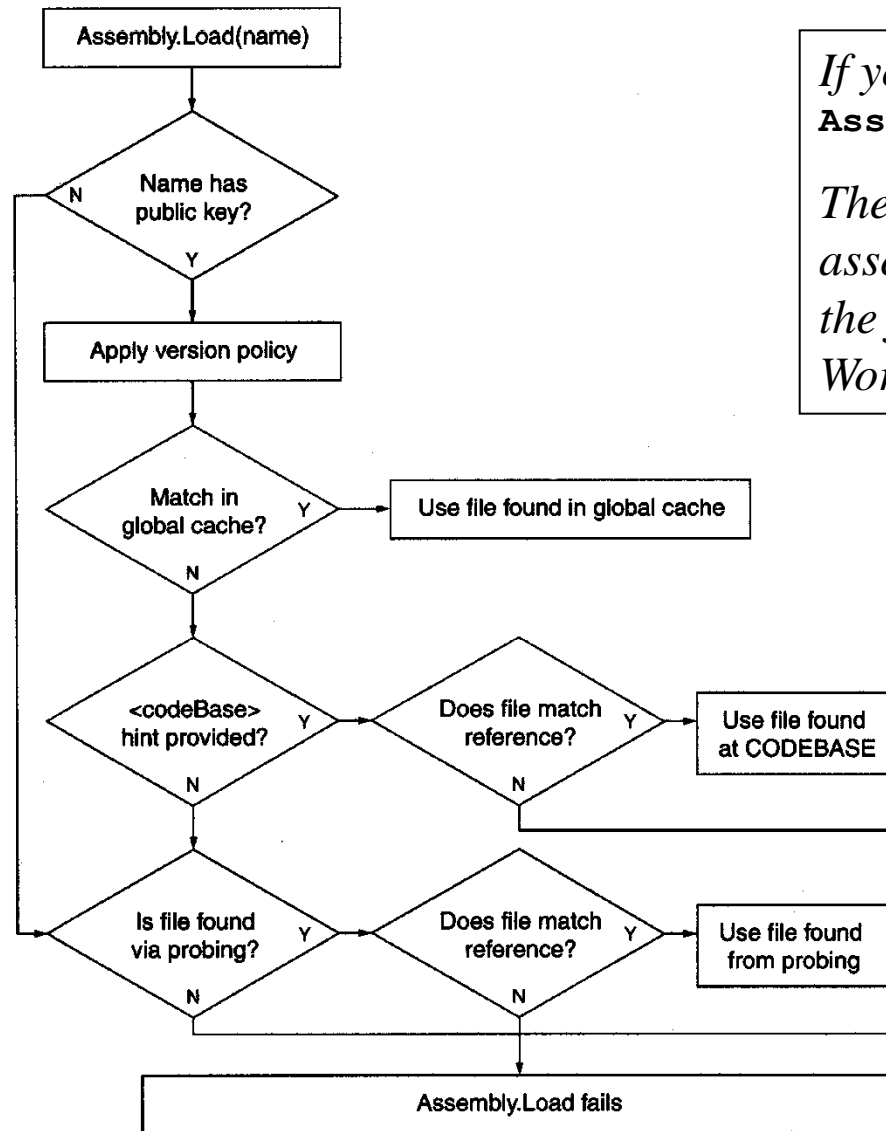
Loader Mechanics



Version Policy Order



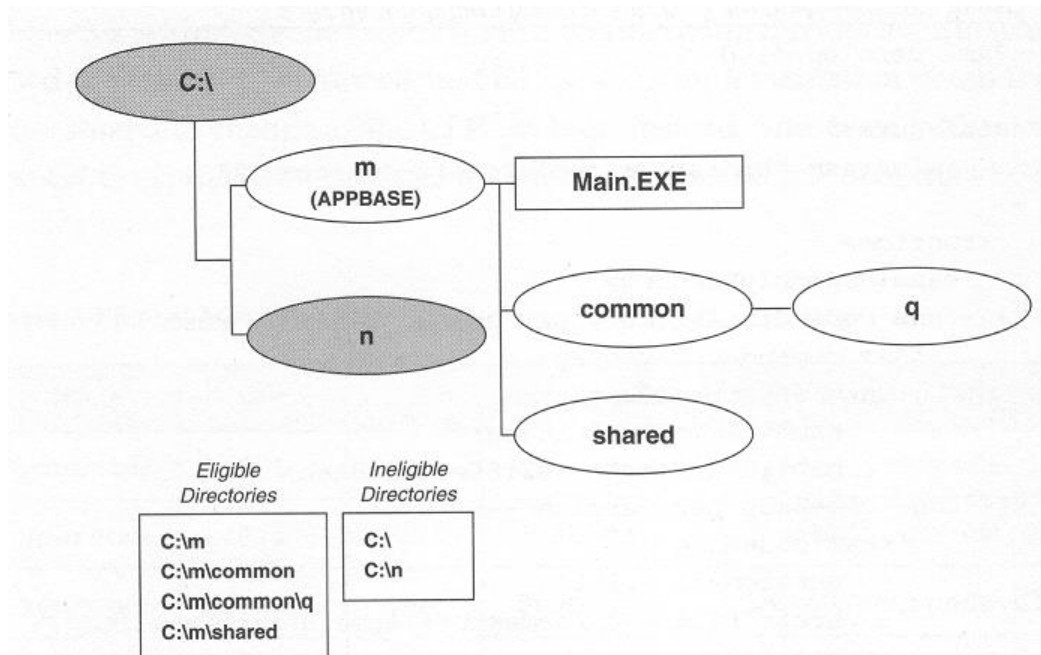
Resolving Names to Locations



*If you use
`Assembly.LoadFrom(string)`
Then you can load an
assembly from any location in
the file system, or on the
World Wide Web.*

Probing

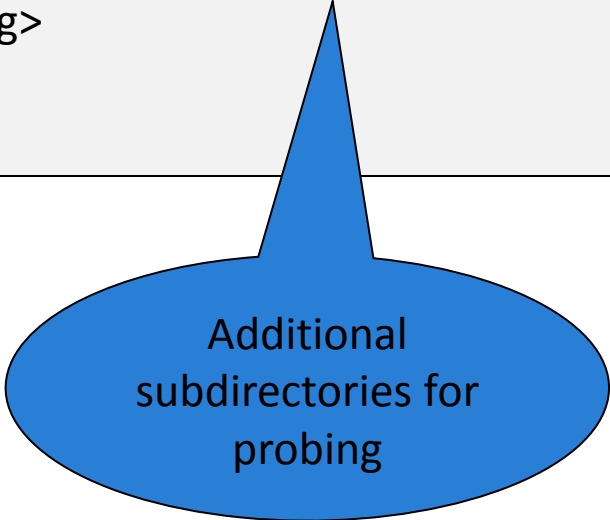
- Probing: the assembly resolver will only look for assemblies in the application directory (where the exe-file is), unless you tell it to look in additional directories in the application configuration file.
- It's only possible to search in directories located below the application directory.



Application Configuration File

- An application configuration file must have the same name as the assembly + .config as an extra extension.
 - If the application assembly is in the file **MyApp.exe**
 - Then the application configuration file must have the name **MyApp.exe.config**
- The application configuration file is an xml-file with the following structure:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration xmlns:asm="urn:schemas-microsoft-com:asm.v1">
  <runtime>
    <asm:assemblyBinding>
      <asm:probing privatePath="Shared;common" />
    </asm:assemblyBinding>
  </runtime>
</configuration>
```



Additional
subdirectories for
probing