.Net basics

1. For pre .net applications the respective compilers generate native/machine code (0’s & 1’s)

For .net applications:

The respective source code is compiled by the respective language compilers which generates the intermediate language code or assembly code (msil or managed code) called as (dll’s or exe’s) these intermediate language code is then passed to the JIT in CLR-common language runtime.

CLR-takes the intermediate language and passes to the JIT (just in time) compiler which then generate the native code.

CLR is dependent on the operating system.

Advantages of CLR:

1. Portability

2. CLR does garbage collection.

When we install .net framework two things happen

1. Respective framework will be installed.
2. CLR for the respective OS will be installed.

2. ILDASM: ildasm (“intermediate language disassembler”) is used to de-assemble the assembly and shows its internal functionality.

To run the ildasm use visual studio developer command prompt

The assembly contains the two things:

1. Manifest file

2. Intermediate language.

Manifest file—manifest file contains the metadata of the assembly like name and version. To change the manifest file information under project properties there will be AssemblyInfo.cs change the properties over there.

The manifest file is used to which version of assembly it should load.

Intermediate language – intermediate language contains the high language code. Like (“LDSTR”)

We can store the intermediate language with “.il” extension using ildasm tool.

ILASM—ilasm (“intermediate language assembler”) takes the “.il” code and converts into the respective assembly either (.dll or .exe)

3. In .Net the assemblies are broadly classified into two types:

1. Strong Named assemblies

2. Weak Named assemblies

The system assemblies like (“System”) are get stored in the GAC (Global Assembly cache) which will be in c:\windows\assembly\

The Strong Named assemblies should consist of 4 parts:

1. Assembly name

2. Version number

3. Culture info

4. Public key

To change the assembly version use AssemblyInfo.cs file under properties of the project and modify

[Assembly: AssemblyVersion(“2.0.0.0”)]

The Assembly Version number can be classified into 4 types

1. Major version

2. Minor Version

3. Build Number

4. Revision Number

The build number and Revision number are defaulted to zero.

To change the culture of that particular assembly use:

[Assembly: AssemblyCulture (“”)]

To assign a public key to the assembly we have to first generate the public-private key with Strong Name tool using visual studio developer command prompt.

Run:

Sn.exe –k c:\MyKeys.snk

The strong named key file will be stored with the extension of .snk

To sign the assembly with strong name use:

[Assembly: AssemblyKeyFile(“c:\\MyKeys.snk”)]

Only strong Named assemblies are deployed into the GAC.

The strong Named assemblies are guaranteed to be unique.

4. GAC—global Assembly Cache

The system assemblies like (“system”) will get stored in to the GAC

There are two GAC’s in the System

1. c:\windows\assembly—.Net framework 2.0-3.5

2. c:\windows\Microsoft.NET\assembly --.Net framework 4.0

The use of deploying the assemblies to the GAC is if there are multiple projects in the solution and if one project is being referenced in the multiple projects to improve the performance we can deploy the assembly to the GAC without creating the local copy of that referenced assembly.

To deploy the assembly to the GAC use:

gacutil.exe –i c:\Projects\classlibrary.dll

Will install into the GAC for that respective framework

To uninstall the assembly from the GAC use :( don’t use .dll or .exe at the end of the assembly for uninstall)

gacutil.exe –u classlibrary,Version=1.0.0.0,PublicKeyToken=56457272727

If we don’t mention version and publickeytoken then all the assemblies with that name are get uninstalled.

Gacutil.exe –u classlibrary

Multiple Assemblies with same name can exist in the Cache but it must differentiate with version number this is called side by side execution.

5. Net Resolution Algorithm:

1. .NET figures out what version of assembly it is needed: this information is available in the assembly manifest file. It checks the config file, machine.config file if the version information is override.

2. .NET searches GAC if it is strongly named.

3. If the assembly is not found in GAC it will search in config file path where the assembly is located or if it is not found then it will check the assembly in the same directory.

4. If the assembly is not found then the application terminates with the error.

Version Checking is not done for weak named assemblies.

5. DLL hell problem:

Dll hell is a problem in which one application downloads the latest version of shared dll which is not backward compatible with all the applications that rely on that shared dll and causes the application to crash.

6. To solve the DLL hell problem:

Strongly name the shared component and then install into the GAC even if there are any changes to the shared component the multiple versions of the shared components can exists in the GAC(side by side execution) and .net resolution algorithm finds the suitable version from the GAC thus it will solve not the application to crash.