[what is the difference between Response.Redirect and server.Transfer?](http://www.codeproject.com/Questions/178831/what-is-the-difference-between-Response-Redirect-a)

Response.Redirect(“newpage.aspx”) and Server.Transfer(“newpage.aspx”)  
   
      Response.Redirect involves a roundtrip to the server whereas Server.Transfer conserves server resources by avoiding the roundtrip. It just changes the focus of the webserver to a different page and transfers the page processing to a different page.  
   
Roundtrip means in case of Response.Redirect it first sends the request for the new page to the browser then browser sends the request for the new page to the webserver then after your page changes But in case of Server.Transfer it directly communicate with the server to change the page hence it saves a roundtrip in the whole process.  
   
      If you are using Server.Transfer then you can directly access the values, controls and properties of the previous page which you can’t do with Response.Redirect.  
   
Suppose you are currently on the Page1.aspx and now you are transferring the user to the Page2.aspx using Response.Redirect then When the Page2 page is requested, Page1 has been flushed from the server’s memory and no information can be retrieved about it unless the developer explicitly saved the information using some technique like session, cookie, application, cache etc. But in case of Server.Transfer variables can stay in scope and Page2 can read properties directly from Page1 because it’s still in memory, as you know the Server.Transfer just changes the focus from page1 to page2 So in this case browser doesn’t know that any change is happen there that’s why with this method you can access the information about the previous page.  
   
      Response.Redirect changes the URL in the browser’s address bar. So they can be bookmarked. Whereas Server.Transfer retains the original URL in the browser’s address bar. It just replaces the contents of the previous page with the new one.  
   
Actually in case of Server.Transfer it directly contact with the webserver for the new page request, it doesn’t involve the browser, so that browser doesn’t know that there is any change happen. But in case of Response.Redirect as you know it first send the request (for new page) to the browser then further processing will be performed, so here browser knows that yes there is some change in the browser window that’s why it changes the URL in the address bar.  
So, the matter “No change of address in the browser address bar” This is a good thing if you see it from security point of view but it creates problem in case if you refresh your page or in case you want to add bookmark of that page. In case of refresh and bookmark it will add perform both the action with the URL currently present in the address bar, but as you know Server.Transfer doesn’t changes the URL, so sometimes it creates problem.  
   
      Response.Redirect can be used for both .aspx and html pages whereas Server.Transfer can be used only for .aspx pages and is specific to ASP and ASP.NET.  
   
With Response.Redirect you can redirect the user to the both type of pages .html or .aspx like below,  
Response.Redirect(“mypage.html”) OR Response.Redirect(“OtherPage.aspx”)  
But in case of Server.Transfer you can only work with .asp or .aspx page like below  
Server.Transfer(“mypage.asp”) OR Server.Transfer(“OtherPage.aspx”)  
   
      Response.Redirect can be used to redirect a user to an external websites. Server.Transfer can be used only on sites running on the same server. You cannot use Server.Transfer to redirect the user to a page running on a different server.  
   
Suppose on some action on the webpage I want to redirect my user to the http://www.yahoo.com so with Response.Redirect you can redirect your user to the external site, but in case of Server.Transfer you can only work with the .asp or .aspx pages that are present in your site.  
   
Now the question is which to use and when to use? Mostly the Server.Transfer method is preferable to use because Server.Transfer is faster since there is one less roundtrip, but in some people say that Server.Transfer is not recommended since the operations typically flow through several different pages due to which you lose the correct URL of the page, but again all depends on your requirement.  
Server.Transfer also allows for more flexibility since you can use HTTPContext.Items to pass variables between pages so, use Server.Transfer when you need to pass context items. Otherwise use Response.Redirect so the user will always see the correct URL in the address bar.

**Language independence**

Language independence means that you can execute your code, written in any language , in any other different language.  
whereas

**Platform independence**

Platform independence means that you can execute/run your application on any machine without being concerned with issues like what is the operating system of the machine etc.Platform independence refers in particular to the ability of the application to run on any machine having any operating system installed on it.

platform independent means you can run the application on any operating system. C is platform independent because all most operating systems are done in C. But to run java, yo need the JVM running on your machine.

### [Static and Dynamic web pages](http://aspdotnetbank-kartik.blogspot.in/2011/11/static-and-dynamic-web-pages.html)

We can broadly classify web sites and web pages into two categories:  
  
1. Static web pages  
2. Dynamic web pages

## Static Web Pages

A static web page is a page which has the same content always.   
  
In case of static web pages, content is written in the page itself as plain html. Until the author of the web page updates the content, the content remains the same in the static pages.   
  
Static web pages are meant for providing information which does not change often. For example, visit<http://www.google.com/intl/en/about.html>. This page is a static page. The content is always the same (until they update the content by uploading a new html file to the web server).  
  
HTML files are used to create static web pages.

## Dynamic Web Pages

Dynamic web pages get content from database. Content is NOT hard-coded in the page itself.   
  
Dynamic pages are created using "serverside code" when the page is loaded every time.  
  
An example for a dynamic page is this tutorial page itself. See the file name in the URL. The file name is "Tutorial8.aspx". We have used only one file to display any tutorial chapter. The chapter number is there as part of the URL (the chapter number is 8 for this chapter).   
  
When you type the URL in the browser, this page is dynamically created from database, based on the chapter id in the URL. The content of each chapter is stored in our database, not in the file itself. When you access the page, our server side code will check what is the TutorialId in the URL. Based on the TutorialId, it will retrieve appropriate chapter content from the database and dynamically create the web page. (You can try to change the TutorialId in the URL to some very large number and try. The page will give an error because our code will fail to get the corresponding chapter content from the database). This entire site has only very few files (like index.aspx, Tutorial.aspx etc)  
  
Dynamic web pages are created using technologies like ASP, ASP.NET, PHP etc.   
  
HTML pages cannot be dynamic. All HTML files are static pages. If you want to write dynamic pages, you must use some technologies like ASP.NET.

### What Are Meta Tags?

HTML meta tags are officially page data tags that lie between the open and closing head tags in the HTML code of a document.

The text in these tags is not displayed, but parsable and tells the browsers (or other web services) specific information about the page. Simply, it “explains” the page so a browser can understand it.

Here's a code example of meta tags:

<head> <title>Not a Meta Tag, but required anyway </title> <meta name="description" content="Awesome Description Here"> <meta http-equiv="content-type" content="text/html;charset=UTF-8"> </head>

For more on the history of meta tags, see our post “[Death of a Meta Tag](http://searchenginewatch.com/article/2066825/Death-Of-A-Meta-Tag)”.

### The Title Tag

Although the title tag appears in the head block of the page, it isn't actually a meta tag. What's the difference? The title tag is a required page “element” according to the W3C. Meta tags are optional page descriptors.

To learn more about best practices for title tag element, our post “[How to Write Title Tags For Search Engine Optimization](http://searchenginewatch.com/article/2154469/How-to-Write-Title-Tags-For-Search-Engine-Optimization)” tells you everything you need to know.

### The Description Meta Tag

This is what the description tag looks like:

<meta name="description" content="Awesome Description Here">

Ideally, your description should be no longer than 155 characters (including spaces). However, check the search engine results page (SERP) of choice to confirm this. Some are longer and some are shorter. This is only a rule of thumb, not a definite “best practice” anymore.

# Introduction to the IDE (Visual C#)

The Visual C# integrated development environment (IDE) is a collection of development tools exposed through a common user interface. Some of the tools are shared with other Visual Studio languages, and some, such as the C# compiler, are unique to Visual C#. The documentation in this section provides an overview of how to use the most important Visual C# tools as you work in the IDE in various phases of the development process.

|  |
| --- |
| **NoteNote:** |
| If you are developing an ASP.NET 2.0 Web application, you will be using the Visual Web Developer IDE,  which is a fully integrated part of Visual Studio. However, if your code-behind pages are in Visual C#,  then you will be using the Visual C# Code Editor within Visual Web Developer. Therefore,  some topics in this section, such as [Designing a User Interface (Visual C#)](http://msdn.microsoft.com/en-us/library/ms173080(v=vs.90).aspx), might not be applicable to Web applications. |

## [Visual C# Tools](javascript:void(0))

The following are the most important tools and windows in Visual C#. The windows for most of these tools can be opened from the **View** menu.

* The Code Editor, for writing source code.
* The C# compiler, for converting C# source code into an executable program.
* The Visual Studio debugger, for testing your program.
* The **Toolbox** and **Designer**, for rapid development of user interfaces by using the mouse.
* **Solution Explorer**, for viewing and managing project files and settings.
* **Project Designer**, for configuring compiler options, deployment paths, resources, and more.
* **Class View**, for navigating through source code according to types, not files.
* **Properties**Window, for configuring properties and events on controls in your user interface.
* **Object Browser**, for viewing the methods and classes available in dynamic link libraries including .NET Framework assemblies and COM objects.
* **Document Explorer**, for browsing and searching product documentation on a local computer and on the Internet.

# Microsoft Intermediate Language - MSIL

MSIL stands for Microsoft Intermediate Language. We can call it as Intermediate Language (IL) or Common Intermediate Language (CIL). During the compile time , the compiler convert the source code into Microsoft Intermediate Language (MSIL) .Microsoft Intermediate Language (MSIL) is a CPU-independent set of instructions that can be efficiently converted to the native code. During the runtime the [Common Language Runtime](http://vb.net-informations.com/framework/common_language_runtime.htm) (CLR)'s [Just In Time](http://vb.net-informations.com/framework/just_in_time_compiler.htm) (JIT) compiler converts the Microsoft Intermediate Language (MSIL) code into native code to the Operating System.

* **IL Code :**The word IL Code stands for Intermediate Language Code. It is a CPU independent partially compiled code. When we develop our .NET  application, we don’t know in what kind of environment our code will run i.e. on which operating system it will be finally hosted, what will be the CPU configuration, etc. So for this purpose we have IL code which is compiled according to machine configuration. IL code is given by Language compiler which is different for different languages for example csc.exe compiler for C#, vbc.exe compiler for VB.NET, etc.
* **CLR :**CLR stands for Common Language Runtime. It is the heart of our .NET Framework. CLR performs following tasks :

1. **Garbage Collection :**When we run our .NET application, many objects are created. Garbage collection is a background process which deletes the objects which are not in use by the application and frees memory.
2. **Code Access Security(CAS) and Code Verification(CV) :**CLR checks the whether the code has access rights and it is safe and authenticated to be used.
3. **IL to Native Translation :**The main task of CLR is to provide IL code to JIT to ensure that code is fully compiled as per machine specification.

* **CTS :**CTS stands for Common Types System. In .NET we have various languages like C#, VB.NET, etc. There may be many situations where we want code written in one language to be used in another. In order to ensure that we have a smooth communication between different languages, we have CTS. CTS ensures that datatypes defined in two different languages get compiled to a common data type so that code written in one language can be used by another.
* **CLS** : CLS stands for Common Language Specifications. It is a subset of CTS. CLS is a set of rules or guidelines which if followed ensures that code written in one .NET language can be used by another .NET language.
* **For example**, one rule is that you cannot use multiple inheritance within .NET Framework. As you know C++ supports multiple inheritance but; when you will try to use that C++ code within C#, it is not possible because C# doesn’t supports multiple inheritance.
* **For example one rule is that we cannot have member functions with same name with case difference only i.e we should not have add() and Add(). This may work in C# because it is case-sensitive but if try to use that C# code in VB.NET, it is not possible because VB.NET is not case-sensitive.**
* **JIT :**JIT stands for Just In Timer Compiler. It is the internal compiler of .NET which takes IL code from CLR and executes it to machine specific instructions.
* JIT-JIT compiles the IL code to Machine code just before execution and then saves this transaction in memory.

**DataReader**

DataReader is used to read the data from database and it is a read and forward only connection oriented architecture during fetch the data from database. DataReader will fetch the data very fast when compared with dataset. Generally we will use ExecuteReader object to bind data to datareader.

To bind DataReader data to GridView we need to write the code like as shown below:

Protected void BindGridview()

{

using (SqlConnection conn = new SqlConnection("Data Source=abc;Integrated Security=true;Initial Catalog=Test"))

{

con.Open();

SqlCommand cmd = new SqlCommand("Select UserName, First Name,LastName,Location FROM Users", conn);

SqlDataReader sdr = cmd.ExecuteReader();

gvUserInfo.DataSource = sdr;

gvUserInfo.DataBind();

conn.Close();

}

}

* Holds the connection open until you are finished (don't forget to close it!).
* Can typically only be iterated over once
* Is not as useful for updating back to the database

**DataSet**

DataSet is a disconnected orient architecture that means there is no need of active connections during work with datasets and it is a collection of DataTables and relations between tables. It is used to hold multiple tables with data. You can select data form tables, create views based on table and ask child rows over relations. Also DataSet provides you with rich features like saving data as XML and loading XML data.

protected void BindGridview()

{

    SqlConnection conn = new SqlConnection("Data Source=abc;Integrated Security=true;Initial Catalog=Test");

    conn.Open();

    SqlCommand cmd = new SqlCommand("Select UserName, First Name,LastName,Location FROM Users", conn);

    SqlDataAdapter sda = new SqlDataAdapter(cmd);

    DataSet ds = new DataSet();

    da.Fill(ds);

    gvUserInfo.DataSource = ds;

    gvUserInfo.DataBind();

}

**DataAdapter**

DataAdapter will acts as a Bridge between DataSet and database. This dataadapter object is used to read the data from database and bind that data to dataset. Dataadapter is a disconnected oriented architecture. Check below sample code to see how to use DataAdapter in code:

protected void BindGridview()

{

    SqlConnection con = new SqlConnection("Data Source=abc;Integrated Security=true;Initial Catalog=Test");

    conn.Open();

    SqlCommand cmd = new SqlCommand("Select UserName, First Name,LastName,Location FROM Users", conn);

    SqlDataAdapter sda = new SqlDataAdapter(cmd);

    DataSet ds = new DataSet();

    da.Fill(ds);

    gvUserInfo.DataSource = ds;

    gvUserInfo.DataBind();

}

**DataReader :**

It is read only format, we can't update records. It is connection oriented, whenever data bind from database that need connection and then connection is disconnected.

|  |
| --- |
|  |
| **.cs page** |
| protected void Bind()  {  SqlConnection con = new SqlConnection("Data Source=database;Integrated Security=true;Initial Catalog=catalog");  {  con.Open();  SqlCommand cmd = new SqlCommand("Select record FROM table", con);  SqlDataReader dr = cmd.ExecuteReader();  grdvw.DataSource = dr;  grdvw.DataBind();  con.Close();  }  } |
| **Dataset** :  It is connectionless oriented. whenever we bind data from database. It connects indirectly to the database and then disconnected. Its easily read and write data from database. |
| **.cs page** |
| protected void Bind()  {  SqlConnection con = new SqlConnection("Data Source=datasource;Integrated Security=true;Initial Catalog=catalog");  con.Open();  SqlCommand cmd = new SqlCommand("select record from table", con);  SqlDataAdapter da = new SqlDataAdapter(cmd);  DataSet ds = new DataSet();  da.Fill(ds);  grdvw.DataSource = ds;  gvdvw.DataBind();  } |
| **DataTable** :  DataTable represents a single table in the database. It has rows and columns. There is no much difference between dataset and datatable, dataset is simply the collection of datatables. |
| **.cs page**  protected void Bind()  {  SqlConnection con = new SqlConnection("Data Source=datasource;Integrated Security=true;Initial Catalog=catalog");  con.Open();  SqlCommand cmd = new SqlCommand("select record from table", con);  SqlDataAdapter da = new SqlDataAdapter(cmd);  DataTable dt = new DataTable();  da.Fill(dt);  grdvw.DataSource = dt;  gvdvw.DataBind();  } |
| **DataAdapter :**  Dataadapter is a disconnected oriented architecture. DataAdapter is like a mediator between DataSet (or) DataTable and database. This dataadapter is used to read the data from database and bind to dataset. |
| .cs page |
| protected void Bind()  {  SqlConnection con = new SqlConnection("Data Source=datasource;Integrated Security=true;Initial Catalog=catalog");  con.Open();  SqlCommand cmd = new SqlCommand("select record from table", con);  SqlDataAdapter da = new SqlDataAdapter(cmd);  DataSet ds = new DataSet();  da.Fill(ds);  grdvw.DataSource = ds;  gvdvw.DataBind();  } |

- See more at: http://www.codefordotnet.com/Asp.net/differnce\_between\_dataset\_and\_datareader\_aspnet.aspx#sthash.HyIEv4Ng.dpuf

# Assemblies

The new basic entity is called an **Assembly**. This is a **collection of class modules presented as a single DLL or EXE file**. Even though EXE files can also be called assemblies, most of the time we talk about libraries if we use this word. Assemblies are very similar to Java Archives (JAR).

Physically, assemblies are files located somewhere on disk and are per definition so-called **Portable Executable (PE)**. Assemblies are loaded on demand. They are not loaded if not needed.

Assemblies have **Metadata** stored to provide version information along with a complete description of methods and types. Part of this metadata is a **Manifest**. The manifest includes identification information, public types and a list of other used assemblies.

We distinguish between the typical DLLs, we've already used in the times before .NET, **Private Assemblies**, used for single programs, and **Global Assemblies** shared among several applications.

Assemblies are still DLLs even if they differ from the former DLLs. In our context, there is no difference between the direct use of DLLs and Private Assemblies. Thus the direct use of DLLs is still an issue and not outdated.

**Direct Use of DLLs**

Source files may be compiled into DLLs instead of EXEs. Even if we link them later on to assemblies, we first compile them to normal DLLs. During EXE file compilation DLLs may be added directly. There is no obligation to create assemblies first.

* Location is specified at compile time. Usually in the same folder like the application's EXE file or in any of the sub folders.
* PATH is not checked while looking up files, neither set by Control Panel 'System' configuration nor set in a Console Window.
* Identified by name only.
* Get smaller EXE files.
* Dynamic linking, i.e. loading on demand.

**Private Assemblies**

Intended use by single applications. Building modules to group common functionality.

* Location is specified at compile time. Usually in the same folder like the application's EXE file or in any of the sub folders.
* PATH is not checked while looking up files, neither set by Control Panel 'System' configuration nor set in a Console Window.
* Identified by name and version if required. But only one version at a time.
* Digital signature possible to ensure that it can't be tampered.
* Get smaller EXE files.
* Dynamic linking, i.e. loading on demand.

**Global Assemblies**

Publicly sharing functionality among different application.

* Located in Global Assembly Cache (GAC).
* Identified by globally unique name and version.
* Digital signature to ensure that it can't be tampered.
* Get smaller EXE files.
* Dynamic linking, i.e. loading on demand.

# 4.   Sample Application

Our sample application is somehow similar to the typical 'Hello world' programs everywhere around. The program meets somebody and says 'Hello', 'How do you do' and 'Good bye'. All the three greetings printed by another library.

**Steps**

1.  Create the **Private Assembly** *GreetAssembly.dll*  from *Hello.dll* and *GoodBye.dll*2.  Create the **Global Assembly** *HowDoYouDoSharedAssembly.dll* from *HowDoYouDo.dll  
3.* Reference Private and Global Assemblies within *App1.exe  
4.* Call DLLs directly and reference Global Assemblies within *App2.exe*

After we have all the three libraries ready, we may compile two EXE main programs. The first one with a reference to *GreetAssembly.dll* and the second adding the *Hello.dll* and *GoodBye.dll* files directly.

**App.cs**

The main application *App.cs* program looks like this:

**using csharp.test.app.greet;  
namespace csharp.test.app {  
  public class Application {  
    public static void Main() {  
      Application a = new Application();  
  
      Hello h = new Hello();  
      h.SayHello();  
  
      HowDoYouDo d = new HowDoYouDo();  
      d.SayHowDoYouDo();  
  
      a.Leave();  
    }  
    private void Leave() {  
      GoodBye g = new GoodBye();  
      g.SayGoodBye();  
    }  
  }  
}**

The *Main()* method creates an Application object to invoke the *Leave()* method at the end to say 'Good bye'. Inside *Main()*, just after the Application object creation, the *Hello* and *HowDoYouDo* classes are instantiated to say 'Hello' and 'How do you do'.

The *Leave()* method is created to verify that, while the program runs, not all the DLLs are loaded. Instead, the DLL with *GoodBye* stays apart until we enter the *Leave()* method's scope. We are going to investigate this later on in details.

We need a 'using' declaration because *Hello* and *GoodBye* are in another **namespace** as the main program.

**Hello.cs**

There is a simple class providing a method to print out a 'Hello':

**namespace csharp.test.app.greet {  
  public class Hello {  
    public void SayHello() {  
      System.Console.WriteLine("Hello my friend, I am a DLL");  
    }  
  }  
}**

The namespace is declared and equals to *GoodBye.cs*. *Hello.cs* and *GoodBye.cs* will be put into a single**Private Assembly**. They must be in the same namespace.

**GoodBye.cs**

Similar to *Hello.cs* but prints a 'Good bye':

**namespace csharp.test.app.greet {  
  public class GoodBye {  
    public void SayGoodBye() {  
      System.Console.WriteLine("Good bye, I am a DLL too");  
    }  
  }  
}**

Uses the same namespace like *Hello.cs*.

**HowDoYouDo.cs**

We are going to implement this source file as a **Global Assembly**:

**using System.Reflection;  
[assembly:AssemblyKeyFile("app.snk")]  
[assembly:AssemblyVersion("1.0.0.0")]  
namespace csharp.test.app {  
  public class HowDoYouDo {  
    public void SayHowDoYouDo() {  
      System.Console.WriteLine("How do you do, I am a Global Assembly");  
    }  
  }  
}**

With the Attributes at the top we specify the key file used to generate a hash code and to declare the version. For what this hash code is used and how the versioning exactly works will be discussed later on. Like in the other classes there is a method to print out a text.

**Compile Classes to DLLs - The CSharp Compiler (CSC)**

To compile our source files we use the C# Compiler (csc):

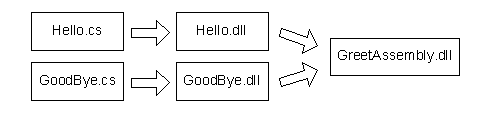
DotNet> **csc /debug /t:module /out:bin\Hello.dll Hello.cs**DotNet> **csc /debug /t:module /out:bin\GoodBye.dll GoodBye.cs**DotNet> **csc /debug /t:module /out:bin\HowDoYouDo.dll HowDoYouDo.cs**

While /debug includes debug information, we are going to use later on, the /t (target) switch lets us create a DLL. We are writing all our DLLs into a bin folder.

**Group DLLs in a Private Assembly - The Assembly Linker (AL)**

We like to combine *Hello.dll* with *GoodBye.dll* and put them into a Private Assembly we call*GreetAssembly.dll*, this is [**Step 1**](http://www.akadia.com/services/dotnet_assemblies.html#Create%20the%20Private%20Assembly) in our Sample Application.

DotNet> **al /t:library /out:bin\GreetAssembly.dll bin\Hello.dll bin\GoodBye.dll**

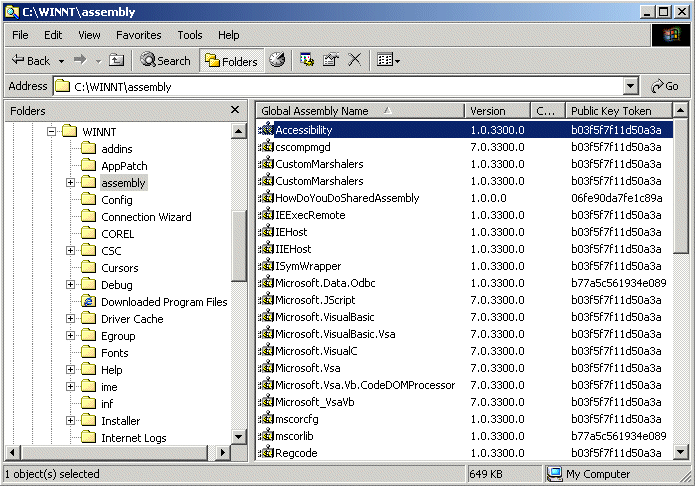


For this purpose we use the Assembly Linker. As /t (target) we generate here a library referencing the two other DLLs. This is also called a Multi-Module Assembly. Again, we store all the binaries in a bin folder.

# 5.   Create Global Assembly

Global Assemblies, also called **Shared Assemblies**, are used to provide globally accessible libraries for different applications. These applications may be used by a single vendor or may be publicly available to other vendors and companies. We do not have control who is using our Global Assemblies. However, this was and is still the case if we store ordinary DLLs somewhere else on disk.

Global Assemblies are presented in the **Global Assembly Cache (GAC)** which can be display by Windows Explorer:



The GAC can be found always in the 'assembly' sub folder inside %SystemRoot%, i.e. WINNT for Windows 2000 and Windows NT.

Assembly is the smallest unit of deployment of a .net application. It can be a **dll** or an **exe**.  
There are mainly two types to it:

1. **Private Assembly:** The dll or exe which is sole property of one application only. It is generally stored in application root folder
2. **Public/Shared assembly:** It is a dll which can be used by multiple applications at a time. A shared assembly is stored in **GAC** i.e [**Global Assembly Cache**](http://msdn.microsoft.com/en-us/library/yf1d93sz).

Sounds difficult? Naa....  
GAC is simply **C:\Windows\Assembly** folder where you can find the public assemblies/dlls of all the softwares installed in your PC.

There is also a third and least known type of an assembly: [**Satellite Assembly**](http://blogs.msdn.com/b/global_developer/archive/2011/07/22/introduction-to-satellite-assemblies.aspx).  
A Satellite Assembly contains only static objects like images and other non-executable files required by the application.

What is satellite assembly?

An assembly which contains culture information is known as satellite assemblies. Satellite assembly is used to get language specific resources for an application.

**Private Assemblies**  
  
Is the assembly which is used by application only, normally it resides in your application folder directory.

There is no version constraint in private assembly.  
  
If an assembly is copied in to the respective application in which we would like to use is known as local assembly. if any changes made to the copy that will not reflect the copies in other applications.  
  
 **Public Assemblies**

* It resides in GAC, so that anyone can use this assembly. Public assemblies are always share the common.
* It has version constraint.
* This public assembly is stored inside the global assembly cache or GAC.GAC contains a collection of shared assemblies.
* If an assembly is copied into the global place and reference is used from all other applications then this is called public or global assembly..if we want to copy assembly in global place we have to create strong name by using sn.exe.

**Satellite Assembly**

Satellite assemblies are used to build **multi-linguistic** applications. Application which has built in supportive of more than one human readable language is known as multi-linguistic applications.

* Satellite Assemblies doesn’t contain any Data
* Satellite assembly is containing cultural information.
* Satellite assembly mainly used for to display information based on the Cultural setting of browser or region.

Suppose you developed your application in an **English (en-US)** locale. Now, your application has multilingual support. When you deploy your code in different location, for example in India, you want to show text, label messages in the national language (local language) which is other than English.  
  
Satellite assemblies give this flexibility. You create any simple text file with translated strings, create resources, and put them into the bin\debug folder. That's it. The next time, your code will read the **CurrentCulture** property of the current thread and accordingly load the appropriate resource.

Please refer below URL for Satellite Assembly

**Globalization, Localization**

To implementing a multilingual user interface, you design the user interface to open in the default UI language and offer the option to change to other languages.

Globalization is the first step in the process. A globalized application supports localized user interfaces and regional data for all users. Truly global applications should be culture-neutral and language-neutral. A globalized application can correctly accept, process, and display a worldwide assortment of scripts, data formats, and languages.

While your globalized application may possess such flexibility, ensure that you have separated the application's resources that require translation from the rest of the application's code. If you correctly test for localizability before proceeding to the localization step, you should not have to modify your application's source code during localization.

**Globalization in .Net - CultureInfo, ResourceManager and Resgen**

In .Net System.Globalization namespace contains classes that define culture-related information, including the language, the country/region, the calendars in use, the format patterns for dates, currency, and numbers, and the sort order for strings. There are around 20 classes in System.Globalization. Here we are only concern to CutureInfo class.

CutureInfo class represents information about a specific culture including the names of the culture, the writing system, and the calendar used, as well as access to culture-specific objects that provide information for common operations, such as formatting dates and sorting strings.

The System.Resources namespace provides classes and interfaces that allow developers to create, store, and manage various culture-specific resources used in an application. One of the most important classes of the System.Resources namespace is the ResourceManager class. The ResourceManager class allows the user to access and control resources stored in the main assembly or in resource satellite assemblies

The Resource File Generator (Resgen.exe) converts .txt files and .resx (XML-based resource format) files to common language runtime binary .resources files that can be embedded in a runtime binary executable or compiled into satellite assemblies.

Globalization is the process of designing and developing applications that function for multiple cultures.   
  
Localization is the process of customizing your application for a given culture and locale.

The language needs to be associated with the particular region where it is spoken, and this is done by using locale (language + location). For example: fr is the code for French language. fr-FR means French language in France. So, fr specifies only the language whereas fr-FR is the locale. Similarly, fr-CA defines another locale implying French language and culture in Canada. If we use only fr, it implies a neutral culture (i.e., location neutral).  
  
**Set culture information**Application level -In web.config file<configuration>  
  <system.web>  
    <globalization culture="fr-FR" uiCulture="fr-FR"/>   
  </system.web>  
</configuration>

The namespaces involved in creation of globalize, localizing applications are

* System.Globalization
* System.Resources
* System.Text
* **Neutral Culture** is only associated with language but not country, while Specific Culture is associated with the language as well as the country and region.
* Let us walkthrough an example to create an application which will change it's Text properties (UI), as and when the language of the system will be changed.
* Create a Windows application (VB / C#), Name it "LocalizedApp", now just Set the forms Localizable property to true, at this time the default language is set to English (United States).
* Place a Label Control over your form (Say Wel\_Label), and Set its text property to "How are you?" from the properties window.
* Let us now program this application for multilingual support; I'll program it for Hindi (India), and French (France) cultures.
* "How are you" is equivalent to "comment soyez vous" in French (France), and is equivalent to http://www.c-sharpcorner.com/UploadFile/brijrajsingh/glbal_110012005021049AM/Images/hindi2.gifin Hindi (India).
* Now change your Forms Language Property to French (France), set the label's (Wel\_Label) Text property to "comment soyez vous", save your program, and compile it.  
    
  Again change the Form's language property to Hindi (India), and set the label's (Wel\_Label) text property to http://www.c-sharpcorner.com/UploadFile/brijrajsingh/glbal_110012005021049AM/Images/hindi2.gif, Also set the Label's Font to Mangal, Save your project and compile it.
* Now, just give a look to your right side solution explorer window, click over the Tab button "Show All Files", it should look like this

**Eval and Bind**

* Eval is one-way, read only databinding.
* Bind is two-way, read/write databinding.
* Bind() is used to two-way databinding. So when used with the proper type of datasource, it will pull the altered values out of the GridView or other control and save them back to the database.
* Eval() is a one-way, "read only" way to bind the values.
* The main difference between eval and bind is eval is read only, we can't change on database thing eval.
* While using bind we can appply some change to the database.

# Boxing and Unboxing

Boxing is the process of converting a [value type](http://msdn.microsoft.com/en-us/library/s1ax56ch.aspx) to the type **object** or to any interface type implemented by this value type. When the CLR boxes a value type, it wraps the value inside a System.Object and stores it on the managed heap. Unboxing extracts the value type from the object. Boxing is implicit; unboxing is explicit. The concept of boxing and unboxing underlies the C# unified view of the type system in which a value of any type can be treated as an object.

In the following example, the integer variable i is *boxed* and assigned to object o.

C#

int i = 123;

// The following line boxes i.

object o = i;

The object o can then be unboxed and assigned to integer variable i:

C#

o = 123;

i = (int)o; // unboxing

[Boxing](javascript:void(0))

Boxing is used to store value types in the garbage-collected heap. Boxing is an implicit conversion of a [value type](http://msdn.microsoft.com/en-us/library/s1ax56ch.aspx) to the type **object** or to any interface type implemented by this value type. Boxing a value type allocates an object instance on the heap and copies the value into the new object.

Consider the following declaration of a value-type variable:

C#

int i = 123;

The following statement implicitly applies the boxing operation on the variable i:

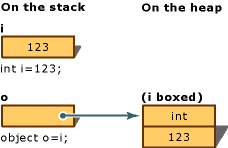
C#

// Boxing copies the value of i into object o.

object o = i;

The result of this statement is creating an object reference o, on the stack, that references a value of the type **int**, on the heap. This value is a copy of the value-type value assigned to the variable i. The difference between the two variables, i and o, is illustrated in the following figure.

Boxing Conversion



It is also possible to perform the boxing explicitly as in the following example, but explicit boxing is never required:

C#

int i = 123;

object o = (object)i; // explicit boxing

[Description](javascript:void(0))

This example converts an integer variable i to an object o by using boxing. Then, the value stored in the variable i is changed from 123 to 456. The example shows that the original value type and the boxed object use separate memory locations, and therefore can store different values.

[Example](javascript:void(0))

C#

class TestBoxing

{

static void Main()

{

int i = 123;

// Boxing copies the value of i into object o.

object o = i;

// Change the value of i.

i = 456;

// The change in i does not effect the value stored in o.

System.Console.WriteLine("The value-type value = {0}", i);

System.Console.WriteLine("The object-type value = {0}", o);

}

}

/\* Output:

The value-type value = 456

The object-type value = 123

\*/

The following example demonstrates a case of invalid unboxing and the resulting **InvalidCastException**. Using **try** and **catch**, an error message is displayed when the error occurs.

C#

class TestUnboxing

{

static void Main()

{

int i = 123;

object o = i; // implicit boxing

try

{

int j = (short)o; // attempt to unbox

System.Console.WriteLine("Unboxing OK.");

}

catch (System.InvalidCastException e)

{

System.Console.WriteLine("{0} Error: Incorrect unboxing.", e.Message);

}

}

}

This program outputs:

Specified cast is not valid. Error: Incorrect unboxing.

If you change the statement:

int j = (short) o;

to:

int j = (int) o;

the conversion will be performed, and you will get the output:

Unboxing OK.

[Unboxing](javascript:void(0))

Unboxing is an explicit conversion from the type **object** to a [value type](http://msdn.microsoft.com/en-us/library/s1ax56ch.aspx) or from an interface type to a value type that implements the interface. An unboxing operation consists of:

* Checking the object instance to make sure that it is a boxed value of the given value type.
* Copying the value from the instance into the value-type variable.

The following statements demonstrate both boxing and unboxing operations:

C#

int i = 123; // a value type

object o = i; // boxing

int j = (int)o; // unboxing

Unsigned and Signed Int

1. UnSigned can hold a larger positive value, and no negative value.
2. Unsigned uses the leading bit, while the signed version uses the left-most-bit to identify if the number is positive or negative.
3. signed integers can hold both positive and negative numbers.
4. A **uint** is an unsigned integer. It is similar to int, but does not reserve space for the sign. Unsigned integers can be used in the place of signed integers in many programs. They help when you need to increase the bounds of a loop.

[**Int**](http://www.dotnetperls.com/int)

uint.MinValue = 0

uint.MaxValue = 4294967295

## Example



The uint type is the same as the System.UInt32 type in the System namespace. This alias provides for clearer code in programs. You can use UInt32 if you include the System namespace or use the fully qualified name "System.UInt32".

**Note:**This example shows some differences between uints and ints, such as the MinValue and MaxValue.

**Example that uses uint type: C#**

using System;

class Program

{

static void Main()

{

// Declare example unsigned and signed integers.

uint value1 = 100;

int value2 = 100;

// Display values.

Console.WriteLine(value1);

Console.WriteLine(value2);

// Assign the maximum values.

uint max1 = uint.MaxValue;

int max2 = int.MaxValue;

// Display maximum values.

Console.WriteLine(max1);

Console.WriteLine(max2);

// Assign the minimum values.

uint min1 = uint.MinValue;

int min2 = int.MinValue;

// Write the minimum values.

Console.WriteLine(min1);

Console.WriteLine(min2);

// Write the types.

Console.WriteLine(typeof(uint));

Console.WriteLine(typeof(int));

}

}

**Output**

100

100

4294967295

2147483647

0

-2147483648

System.UInt32

System.Int32

Convert unsigned to signed

UInt16 x = 65535;

var y = (Int16)x;

unsigned int z;

int y=5;

z= (unsigned int)y

unsigned int foo;

int bar = 10;

foo = (unsigned int)bar;

## Validation Server Controls

A Validation server control is used to validate the data of an input control. If the data does not pass validation, it will display an error message to the user.

The syntax for creating a Validation server control is:

<asp:control\_name id="some\_id" runat="server" />

|  |  |
| --- | --- |
| **Validation Server Control** | **Description** |
| [CompareValidator](http://www.w3schools.com/aspnet/control_comparevalidator.asp) | Compares the value of one input control to the value of another input control or to a fixed value |
| [CustomValidator](http://www.w3schools.com/aspnet/control_customvalidator.asp) | Allows you to write a method to handle the validation of the value entered |
| [RangeValidator](http://www.w3schools.com/aspnet/control_rangevalidator.asp) | Checks that the user enters a value that falls between two values |
| [RegularExpressionValidator](http://www.w3schools.com/aspnet/control_regularexpvalidator.asp) | Ensures that the value of an input control matches a specified pattern |
| [RequiredFieldValidator](http://www.w3schools.com/aspnet/control_reqfieldvalidator.asp) | Makes an input control a required field |
| [ValidationSummary](http://www.w3schools.com/aspnet/control_validationsummary.asp) | Displays a report of all validation errors occurred in a Web page |

Custom validation

The following code example demonstrates how to create a **CustomValidator** control that performs the same validation routine as the previous example, but on the client.

C#

[**VB**](http://msdn.microsoft.com/en-us/library/9eee01cx(v=vs.85).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-4)

<%@ Page Language="C#" AutoEventWireup="True" %>

<html>

<head>

<script runat="server">

void ValidateBtn\_OnClick(object sender, EventArgs e)

{

if (Page.IsValid)

{

lblOutput.Text = "Page is valid.";

}

else

{

lblOutput.Text = "Page is not valid!";

}

}

void ServerValidation (object source, ServerValidateEventArgs arguments)

{

int i = int.Parse(arguments.Value);

arguments.IsValid = ((i%2) == 0);

}

</script>

</head>

<body>

<form runat="server">

<h3>CustomValidator Example</h3>

<asp:Label id=lblOutput runat="server"

Text="Enter an even number:"

Font-Name="Verdana"

Font-Size="10pt" /><br>

<p />

<asp:TextBox id="Text1"

runat="server" />

&nbsp;&nbsp;

<asp:CustomValidator id="CustomValidator1"

ControlToValidate="Text1"

ClientValidationFunction="ClientValidate"

OnServerValidate="ServerValidation"

Display="Static"

ErrorMessage="Not an even number!"

ForeColor="green"

Font-Name="verdana"

Font-Size="10pt"

runat="server"/>

<p />

<asp:Button id="Button1"

Text="Validate"

OnClick="ValidateBtn\_OnClick"

runat="server"/>

</form>

</body>

</html>

<script language="javascript">

<!--

function ClientValidate(source, arguments)

{

if ((arguments.Value % 2) == 0)

arguments.IsValid=true;

else

arguments.IsValid=false;

}

// -->

</script>

The validation control classes are inherited from the BaseValidator class hence they inherit its properties and methods. Therefore, it would help to take a look at the properties and the methods of this base class, which are common for all the validation controls:

|  |  |
| --- | --- |
| **Members** | **Description** |
| ControlToValidate | Indicates the input control to validate. |
| Display | Indicates how the error message is shown. |
| EnableClientScript | Indicates whether client side validation will take. |
| Enabled | Enables or disables the validator. |
| ErrorMessage | Indicates error string. |
| Text | Error text to be shown if validation fails. |
| IsValid | Indicates whether the value of the control is valid. |
| SetFocusOnError | It indicates whether in case of an invalid control, the focus should switch to the related input control. |
| ValidationGroup | The logical group of multiple validators, where this control belongs. |
| Validate() | This method revalidates the control and updates the IsValid property. |

# Overview of the .NET Framework

**.NET Framework 4.5**

[Other Versions](javascript:;)

http://i3.msdn.microsoft.com/Areas/Epx/Content/Images/ImageSprite.png

205 out of 271 rated this helpful - [Rate this topic](http://msdn.microsoft.com/en-us/library/zw4w595w(v=vs.110).aspx#feedback)

The .NET Framework is a technology that supports building and running the next generation of applications and XML Web services. The .NET Framework is designed to fulfill the following objectives:

* To provide a consistent object-oriented programming environment whether object code is stored and executed locally, executed locally but Internet-distributed, or executed remotely.
* To provide a code-execution environment that minimizes software deployment and versioning conflicts.
* To provide a code-execution environment that promotes safe execution of code, including code created by an unknown or semi-trusted third party.
* To provide a code-execution environment that eliminates the performance problems of scripted or interpreted environments.
* To make the developer experience consistent across widely varying types of applications, such as Windows-based applications and Web-based applications.
* To build all communication on industry standards to ensure that code based on the .NET Framework can integrate with any other code.

ASP.NET is a development framework for building web pages and web sites with HTML, CSS, JavaScript and server scripting.

ASP.NET supports three different development models:  
Web Pages, MVC (Model View Controller), and Web Forms:

CLR

its a common laungage runtime environement .The CLR provides an environment in which programs are executed.It is responsible for processs such as memory management and reference counting of objects.  
  
        Another feauture is garbage collector.In c#.net is not responsible for destroying objects.The CLR cleans up after you when your objects are not used no longertime.

CLR (Common Language Runtime) is a heart of Dot Net Framework.

It is a core runtime environment of .NET Framework for executing applications. The main function of Common Language Runtime (CLR) is to convert the Managed Code into native code and then execute the Program. It acts as a layer between Operating Systems and the applications written in .Net languages.

CLR handles the execution of code and provides useful services for the implementation of the program. In addition to executing code, CLR provides services such as memory management, thread management, security management, code verification, compilation, and other system services.

* **What is the difference between Array and Arraylist?**

In an array, we can have items of the same type only. The size of the array is fixed. An arraylist is similar to an array but it doesn’t have a fixed size.

**Arrays**

Arrays are strongly typed collection of same datatype and these arrays are fixed length that cannot be changed during runtime. Generally in arrays we will store values with index basis that will start with zero. If we want to access values from arrays we need to pass index values.

**Declaration of Arrays**

Generally we will declare arrays with fixed length and store values like as shown below

|  |
| --- |
| string[] arr=new string[2];  arr[0] = "welcome";  arr[1] = "Aspdotnet-suresh"; |

In above code I declared array size 2 that means we can store only 2 string values in array.

**Arraylists**

Array lists are not strongly type collection. It will store values of different datatypes or same datatype. Array list size will increase or decrease dynamically it can take any size of values from any data type. These Array lists will be accessible with “**System.Collections**” namespace

**Can a private virtual method be overridden?**

No, because they are not accessible outside the class.

**Is C# code is managed or unmanaged code?**

C# is managed code because Common language runtime can compile C# code to Intermediate language.

What is C#..?

C# is a general object-oriented programming (OOP) language for networking and Web development. C# is specified as a common language infrastructure (CLI) language.

C# is an elegant and type-safe object-oriented language that enables developers to build a variety of secure and robust applications that run on the .NET Framework. You can use C# to create Windows client applications, XML Web services, distributed components, client-server applications, database applications, and much, much more. Visual C# provides an advanced code editor, convenient user interface designers, integrated debugger, and many other tools to make it easier to develop applications based on the C# language and the .NET Framework.

* Difference between Struct and Class.

**Classes Only:**

* Can support inheritance
* Are reference (pointer) types
* The reference can be null
* Have memory overhead per new instance

**Structs Only:**

* Cannot support inheritance
* Are value types
* Are passed by value (like integers)
* Cannot have a null reference (unless Nullable is used)
* Do not have a memory overhead per new instance - unless 'boxed'

1. Structs cannot have explicit parameterless constructor where as a class can  
2. Structs cannot have destructors, where as a class can  
3. Struct can't inherit from another class where as a class can, Both structs and classes can inherit from an interface.

This can be shown with an example:

struct MyStruct

{

string MyProperty { get; set; }

}

void ChangeMyStruct(MyStruct input)

{

input.MyProperty = "new value";

}

...

// Create value type

MyStruct testStruct = new MyStruct { MyProperty = "initial value" };

ChangeMyStruct(testStruct);

// Value of testStruct.MyProperty is still "initial value"

// - the method changed a new copy of the structure.

For a class this would be different

class MyClass

{

string MyProperty { get; set; }

}

void ChangeMyClass(MyClass input)

{

input.MyProperty = "new value";

}

...

// Create reference type

MyClass testClass = new MyClass { MyProperty = "initial value" };

ChangeMyClass(testClass);

// Value of testClass.MyProperty is now "new value"

// - the method changed the instance passed.

Classes can be nothing - the reference can point to a null.

Structs are the actual value - they can be empty but never null. For this reason structs always have a default constructor with no parameters - they need a 'starting value'.

A struct instance cannot be null.

class Nullness

{

static void Eg(Direct s, Indirect c)

{

if (s == null) ... *// compile-time error*

if (c == null) ... *// compiles ok*

}

}

**Use with the as operator?**

A struct type cannot be the right hand side operand of the as operator.

class Fragment

{

static void Eg(Direct s, Indirect c)

{

Direct no = s as Direct; *// compile-time error*

InDirect yes = c as InDirect; *// compiles ok*

*//...*

}

}

**14.   What are value types and reference types?**

Value types are stored in the Stack whereas reference types stored on heap.  
Value types:

[csharp] int, enum , byte, decimal, double, float, long[/csharp]

Reference Types:

[csharp] string , class, interface, object.[/csharp]

**What is serialization?**

When we want to transport an object through network then we have to convert the object into a stream of bytes. The process of converting an object into a stream of bytes is called Serialization. For an object to be serializable, it should inherit ISerialize Interface.  
De-serialization is the reverse process of creating an object from a stream of bytes.

**What is the difference between public, static and void?**

All these are access modifiers in C#. Public declared variables or methods are accessible anywhere in the application. Static declared variables or methods are globally accessible without creating an instance of the class. The compiler stores the address of the method as the entry point and uses this information to begin execution before any objects are created. And Void is a type modifier that states that the method or variable does not return any value.

|  |  |  |
| --- | --- | --- |
| **What isAuthentication and Authorization?** | | |
|  | | | **Authentication** is the process of identifying users. Authentication is identifying/validating the user against the credentials (username and password). **Authorization** performs after authentication. Authorization is the process of granting access to those users based on identity. Authorization allowing access of specific resource to user. |
| **What is difference between constants, readonly and, static ?** |
|  | | * Constants: The value can’t be changed. * Read-only: The value will be initialized only once from the constructor of the class. * Static: Value can be initialized once. | |
|  | |  | |

|  |  |  |
| --- | --- | --- |
| **39.** | **What are the different types of statements supported in C#?** | |
|  | C# supports several different kinds of statements are   * Block statements * Declaration statements * Expression statements * Selection statements * Iteration statements * Jump statements * Try catch statements * Checked and unchecked * Lock statement | |
| **What are the different types of Caching?** | |
|  | | There are three types of Caching :   * Output Caching: stores the responses from an asp.net page. * Fragment Caching: Only caches/stores the portion of page (User Control) * Data Caching: is Programmatic way to Cache objects for performance.  |  | | --- | | **Define namespace ?** | |  | The namespace are known as containers which will be used to organize the hierarchical set of .Net classes. | |

|  |  |
| --- | --- |
| **What is the difference between static and instance methods?** | |
|  | | A method declared with a static modifier is a static method. A static method does not operate on a specific instance and can only access static members.  A method declared without a static modifier is an instance method. An instance method operates on a specific instance and can access both static and instance members. The instance on which an instance method was invoked can be explicitly accessed as this. It is an error to refer to this in a static method. |
| **What are the special operators in C#?** |
|  | C# supports the following special operators.   * is (relational operator) * as (relational operator) * typeof (type operator) * sizeof (size operator) * new (object creator) * .dot (member access operator) * checked (overflow checking) * unchecked (prevention of overflow checking) | |

**Do events have return type?**

No, events do not have return type.

**Does C# provide copy constructor?**

No, C# does not provide copy constructor.

**Can an Interface contain fields?**

No, an Interface cannot contain fields.

**Can you create an instance of an interface?**

No, you cannot create an instance of an interface.