# DOT NET Framework

**What is COM?**

Component object model یا com یک تکنولوژی ماکروسافت است با استفاده از آن امکان توسعه اپلیکیشن ویندوزی نیز وجود دارد در com از زبان برنامه نویسی vb استفاده می شود.

**What are the disadvantages of COM?**

عدم پشتیبانی از تمام ویژگی های برنامه نویسی شی گرا

قابلیت اجرا فقط در سیستم عامل Windows

**برای حل این مسائل .net framework وارد شد.**

**What .NET Represents?**

**.net** خلاصه شده ی network enable technology است و . به معنای object oriented است کلمه ی net به اینترنت اشاره دارد که معنای کامل می شود می توانیم اپلیکیشن های بر پایه اینترنت را با استفاده از oop ایجاد کنیم.

**What is a Framework?**

مجموعه ای از تکنولوژی های کوچک است که در کنار هم دیگیر تجمیع شده اند برای توسعه اپلیکیشن ها .

**What is .NET Framework?**

یک framework برای توسعه ایجاد و اجرای اپلیکیشن ها در ویندوز اندورید مک لینوکس اندروید و .. است.

**Different Types of .NET Framework.**

از زبان ها و کتابخانه های متفاوتی ساخته شده است که برای ساخت و توسعه اپلیکیشن ها دستکاپی و وب موبایل و ..

1. **.NET Framework** is the original implementation of .NET. It supports running websites, services, desktop apps, and more on Windows.
2. **.NET** is a cross-platform implementation for running websites, services, and console apps on Windows, Linux, and macOS. .NET is open source on GitHub. .NET was previously called .NET Core.
3. **Xamarin/Mono** is a .NET implementation for running apps on all the major mobile operating systems, including iOS and Android.

#### **What does the .NET Framework Provide?**

2 کامپوننت اصلی آن عبارتند از class library – common language runtime

Cl , clr

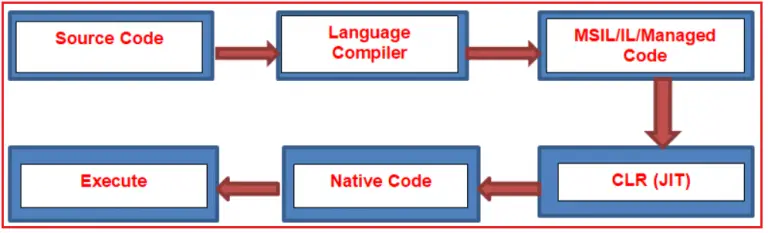
**.NET Framework Class Libraries:**

برای این که بتوانیم کدنویسی .net انجام دهیم باید class library وجود داشته باشد در واقع class library بلوک های ایجاد اپلیکیشن .net است و زمانی که .net را نصب می کنیم در ماشین ما نصب می شود که شامل کلاس ها و رابط های است که از قبل نصب شده اند که هدف آن توسعه اپلیکیشن است. ارائه دهنده data type ها و .. است.

**Common Language Runtime (CLR)**

هسته اصلی برای تبدیل msil یا Microsoft intermediate language به کد native است و ارائه دهنده محیط برای اجرای کد است.

Clr در واقع موتور اجرا و مدیریت اپلیکیشن های .net است و سرویس های مثل thread management – garbage collector – type safety و ... را ارائه می دهد



**کد .net در واقع 2 بار کامپایل می شود**

1. **In the 1st compilation, the source code is compiled by the respective language compiler and generates the intermediate code which is known as MSIL (Microsoft Intermediate Language) or IL (Intermediate language code) Or Managed Code.**
2. **In the 2nd compilation, MSIL code is converted into Native code (native code means code specific to the Operating system so that the code is executed by the Operating System ) and this is done by CLR.**

**Always 1st compilation is slow and 2nd compilation is fast.**

**What is JIT?**

به معنای just in time است که در واقع یک کاپوننت از clr است که برای تبدیل msil به native code است که مستقیم توسط سیستم عامل فهمیده می شود.

Microsoft-designed programming languages are as follows

1. VB.NET
2. C#.NET
3. VC++.NET
4. J#.NET
5. F#.NET
6. Jscript.NET
7. WindowsPowerShell
8. Iron phyton
9. Iron Ruby

Technologies supported by the .NET framework are as follows

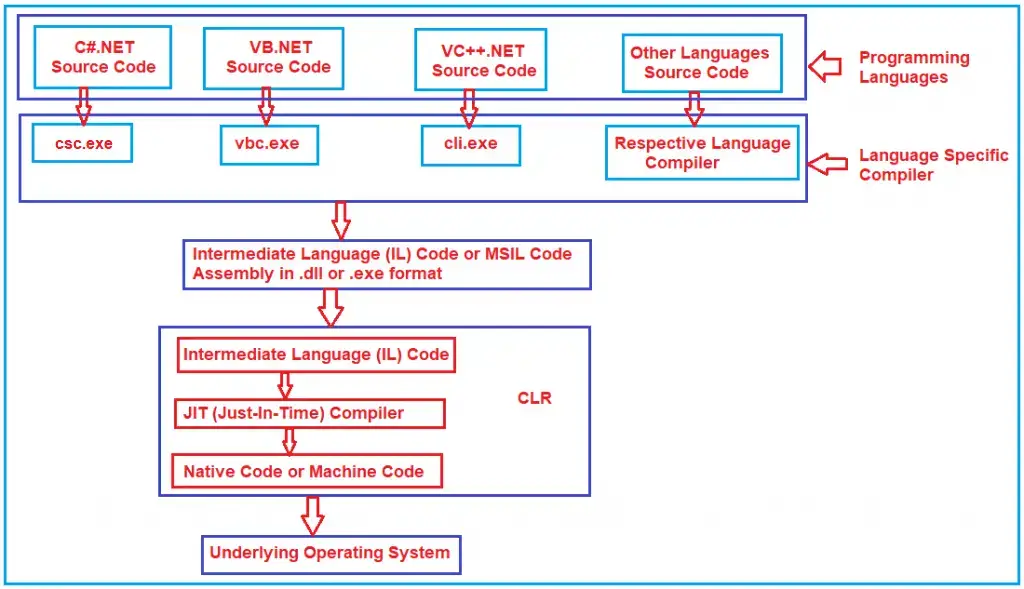
1. ASP.NET (Active Server Pages.NET)
2. ADO.NET (Active Data Object.NET)
3. WCF (Windows Communication Foundation)
4. WPF (Windows Presentation Foundation)
5. WWF (Windows Workflow Foundation)
6. AJAX (Asynchronous JavaScript and XML)
7. LINQ (Language Integrated Query)

**What is a Language and its need?**

1. Language acts as the mediator between the programmer and the system.
2. It offers some rules and regulations for writing the program.
3. The language also offers some libraries which are required for writing the program.

# Common Language Runtime in .NET Framework

**How is a .NET Application Compiled and Run?**



ابتدا کد نوشته شده با یک از زبان های تحت پوشش .net کامپایل می شود توسط کامپایلر داخلی و تبدیل به کدی به نام msil می شود intermidate language به طور مثال در زبان برنامه نویسی سی شارپ با استفاده از کامپایلر csc کامپایل می شود .

کد il یک کد نیمه کامپایل شده است که نمی تواند مستقیما توسط os فهمیده شود برای اجرای کد il توسط سیستم عامل لازم است که کد il به کد نزدیک به ماشین با استفاده از ابزار crl که jit نام دارد کد میانی به کد ماشین تبدیل شود clr یک محیط runtime ایجاد می کند برای اجرای اپلیکیشن.

##### **What is Intermediate Language (IL) Code in .NET Framework?**

کد نیمه کامپایل شده است که مستقیم توسط سیستم عامل نمی تواند اجرا شود.

##### **Why Partial Compiled Code or Why Not Fully Compiled Code?**

دلیل این که کامپایل یک مرحله نداریم این است که نمی دانیم محیط اجرای نرم افزار ما چیست ! کدام سیستم عامل است؟

نمی دانیم cpu مقصد چیست نمی دانیم تنظیمات امنیتی مقصد چیست و ... به همین دلیل زمان اجرا کد میانی را به کد ماشین مقصد تبدیل می کنیم.

##### **Common Language Runtime (CLR) in .NET Framework:**

clr قلب نرم افزار های .net است.

کامپوننت های مهم clr

1. Security Manager
2. JIT Compiler
3. Memory Manager
4. [**Garbage Collector**](https://dotnettutorials.net/lesson/garbage-collector/)
5. [**Exception Manager**](https://dotnettutorials.net/lesson/exception-handling-csharp/)
6. [**Common Language Specification (CLS)**](https://dotnettutorials.net/lesson/common-language-specification/)
7. [**Common Type System (CTS)**](https://dotnettutorials.net/lesson/common-type-system/)

##### **Security Manager:**

1. **CAS (Code Access Security)**
2. **CV (Code Verification)**

بررسی می کنند که آیا کاربر جاری به اسمبلی ها دسترسی دارد یا نه. همچنین بررسی می کند که کد دسترسی های لازم را دارد که به صورت امن توسط سیستم عامل اجرا شود یا نه؟

##### **JIT Compiler**

تبدیل کننده کد میانی به کد قابل فهم ماشین. کد را قبل اجرا کامپایل می کند و در memory ذخیره می کند.

##### **Memory Manager:**

کار تخصیص حافظه برای متغییر ها و object ها را انجام می دهد.

##### **Garbage Collector**

زمانی که اپلیکیشن اجرا می شود object های فراوانی ایجاد می شوند و ممکن است برخی توسط اپلیکیشن استفاده نشوند بنابراین gb یک backGroundProccess thread است که تشخیص می دهد کدام شی ها نیاز نیستند که حافظه را بگیرد.

##### **Exception Manager:**

جهت هدایت و کنترل کد catch و finally در زمان اجرا در صورتی که exception رخ دهد اگر مدیریت نشده باشد سبب توقف اجرا terminate شدن می شود.

##### **Common Type System (CTS) in .NET Framework:**

برای اطمینان از این که کد های نوشته شده در هر زبان که دارای data type خود هستند در مواقعی که نیاز به تامل با یکدیگر دارند به درستی اجرا شوند. تمام Data type ها به clr data type تبدیل می شوند.

##### **CLS (Common Language Specification) in .NET Framework:**

هر زبان برنامه نویسی در چهارچوب .net دارای syntax با قوانین خود است برای ارتباط بین زبان ها از cls استفاده می شود در Clr که به زبان خود ترجمه می کند تمامی زبان ها را در زمان اجرا.

# **.NET Program Execution Process**

##### **Non-DOT NET Program Execution** **Process**

زمانی که سایز زبان ها کامپایل می شوند به کد ماشین یا binary تبدیل می شوند مثل c++ که توسط سیستم عامل قابل فهم هستند.

اگر در این نوع زبان ها کد native تولید شده را به ماشین دیگری ببریم که از پلتفرم ما پشتیبانی نکند اجرا با خطا مواجه شده و امکان پذیر نیست به دلیل وابستگی به پلتفرم.

##### **.NET Program Execution Process:**

با استفاده از .net می توانیم اپلیکیشن وب ویندوزی موبایل و .. ساخت کد مربوط به آن ها به il کامپایل می شود. تمامی اپلیکیشن های .net و غیر .net به اسمبلی تبدیل می شوند که دارای پسوند dll ,exe هستند. مثال اگر اپ کنسولی باشد exe و اگر کلاس لایبرری باشد dll داریم.

تفاوت .net و مابقی در این است که assembly در .net از نوع il و در سایرین از نوع native code است.

غیر .net مستقیم توسط os اجرا می شود اما .net در محیطی مجازی به نام clr و با کامپوننت jit که تبدیل کننده il به native code است اجرا می شود.

کد il در تمامی پلتفرم های پشتیبانی شده توسط clr قابل اجرا است.

Managed code Or IL به این دلیل است که توسط clr مدیریت و اجرا می شود کد درون آن.

کد native در جای ذخیره نمی شود بلکه بعد از بسته شدن آپ از بین می رود و با اجرای مجدد دوباره generate می شود.

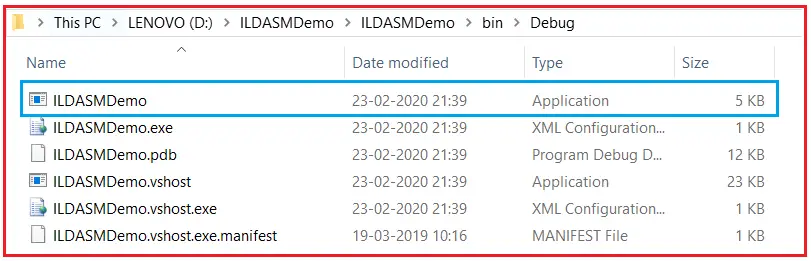
کد java هم مانند .net درای jvm است برای اجرا.

# **Intermediate Language (ILDASM & ILASM) Code in C#**

**ILDASM**  Intermediate Language Disassembler

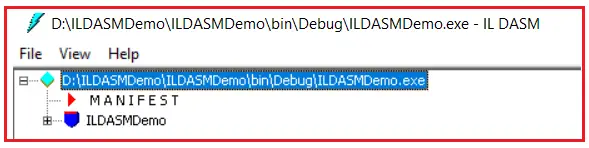
**ILASM**  Intermediate Language Assembler.

یک assembly شامل Manifest , intermediate language است.



برای اجرایفایل ILDASM می توان از cmd استفاده کرد

**D:\ILDASMDemo\ILDASMDemo\bin\Debug\ILDASMDemo.exe**



با کلیک بر روی Demo می توانید جزئیات برنامه را ببینید.

Manifest شامل متا دیتای مربوط به assembly مثل نام و ورژن است.

تغییر اطلاعات Assembly از طریق properties امکان پذیر است.



امکانexport گرفتن از فایل ILDASM نیز به txt وجود دارد.

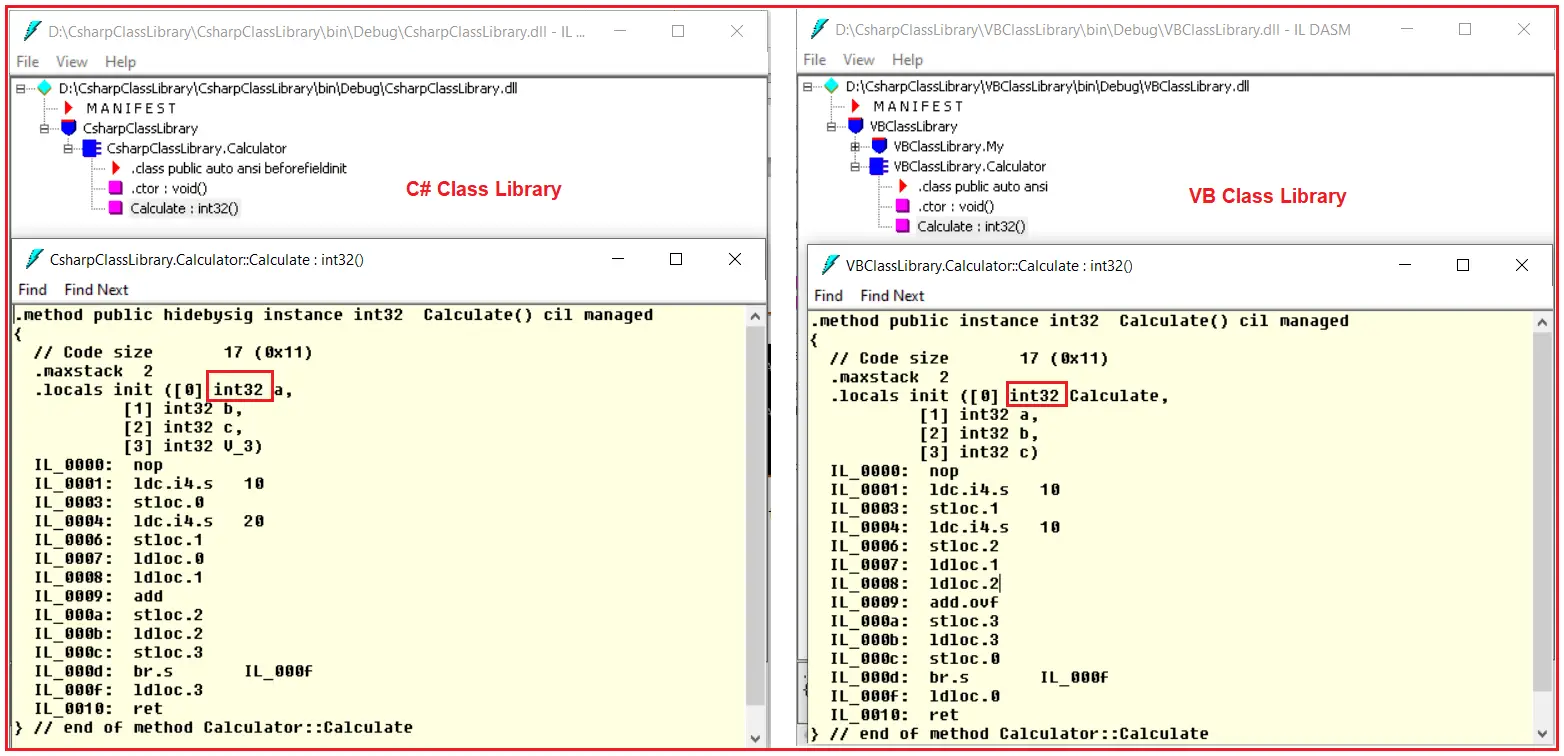
امکان rebuild کردن فایل assembly از طریق فایل text نیز وجود دارد.

# **Common Type System in .NET Framework**

Cts مشخص کننده نحوه استفاده تعریف و مدیریت data type ها ست در clr.

مواقعی که نیاز است نرم افزار ما با زبان برنامه نویسی دیگری از .net تعامل داشته باشد cts به میدان می آید.

Clr تمامی Data type های زبان های مختلف .net را پشتیبانی می کند به این دلیل که دارای Data type خودش است که همه ی زبان ها به Data type clr تبدیل می شوند در زمان اجرا.



# **Common Language Specification in .NET Framework**

به دلیل این که زبان های برنامه نویسی مختلف که در محیط .net وجود دارد هر کدام دارای syntax rule منحصر به فرد خود هستن و بخواهند با یکدیگر تعامل داشته باشند لازم است که همه ی آن ها به زبان مشترکی clr با کامپایل کردن به il ایجاد می کند تبدیل شوند. که این عمل توسط cls انجام می شود.

به عنوان مثال C# از نوع Case sensitive است و vb این گونه نیست. Cls قابلیت تعامل بین زبانی در چهارچوب .net را فراهم می کند.

# **Managed and Unmanaged Code in .NET Framework**

کد مدیریت نشده کدی است که توسط .net نوشته و مدیریت نشده است. زمانی که برنامه exe یا یک dll مینویسیم با استفاده از زبان های تحت پوشش .net چه در قالب web , windows و console این اپلیکیشن ها در زمان اجرا به صورت کامل تحت کنترل Clr اجرا می شوند. به این معنی که تمامی کامپوننت های مربوط به .clr به این معنی که تمامی عملیات های garbage collector , common type system , common language specifiacation , code access security , code verification توسط clr انجام می شود. بنابراین زمانی در برنامه خود از third party app استفاده می کنیم طبیعی است که هر کدام در environment خود اجرا شوند و تحت نظر clr نباشند.

مزیت آن استفاده از ویژگی کامپوننت های Clr است و معایب آن عدم دسترسی به لایه های پایین سخت افزار است به دلیل جلوگیری clr.

## **Assembly DLL EXE in .NET Framework**

اسمبلی بلاک های ساختاری مربوط به اپلیکیشن های .net است یک واحد مبنایی اپلیکیشن. کد پیش کامپایل شده که آماده اجرا توسط clr است

##### **Types of Assemblies of in .NET Framework:**

Executable => exe

Dll => dynamic link library

هر دو اسمبلی نامیده می شوند.

##### **What is the difference between the DLL and the EXE in .NET Framework?**

فایل های Exe در فضای آدرس خود در حافظه اجرا می شوند واگر یک برنامه را 2 بار اجرا کنید هر دو اجرا می شوند به این دلیل که هر کدام در فضای حافظه خود هستند. Exe می تواند خود به عنوان آپ اجرا شود.

DLL

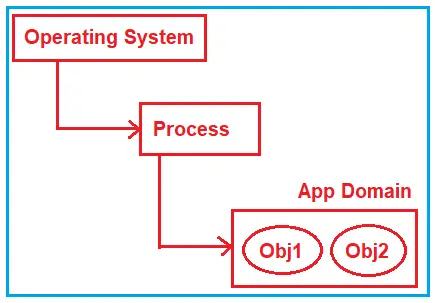
نمی تواند به صورت تنها اجرا شود بلکه نیاز دارد به یک مصرف کننده که آن را اجرا کند. Dll در فضای حافظه ی دیگری اجرا می شود.

# App Domain in .NET Framework

یک نگه دارنده ایزوله کد است در زمانی که کد .net اجرا می شود.

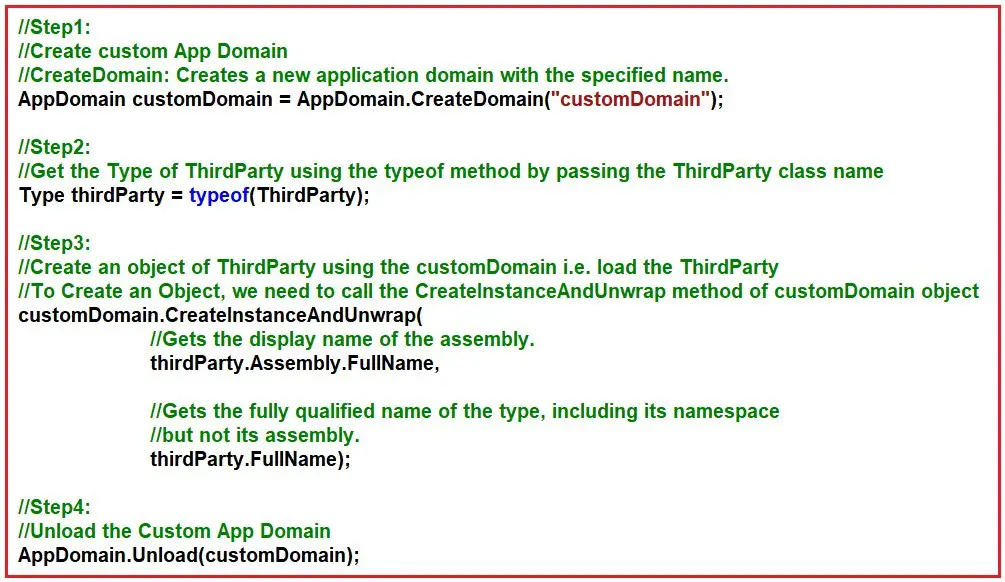
**Understanding App Domain in .NET Framework**

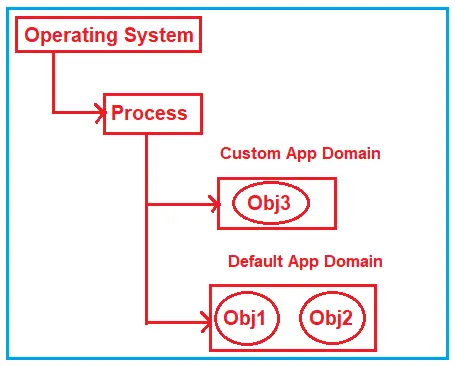
کدی را فرض کنید از نوع console که دارای 2 کلاس است و از هر کدام در متد main یک object جدید ایجاد شده است یک نرم افزار را اجرا کنید در یک process اجرا شده و شامل یک domain می شود که در این domain از 2 کلاس یک نمونه وجود دارد



**Need for App Domain in .NET Application**

فرض کنید که به یک dll خارج از برنامه خود نیاز دارید و آن را برای گزارش گیری می خواهید استفاده کنید از اینترنت یا third party های دیگر آن را دانلود کرده و اجرا می کنید اما به جای آن که گزارش گیری کند به ساخت فایل در درایو شما می پردازد برای جلوگیری از این کار و ایجاد دامنه عمل برای dll می توان از appDomain استفاده کرد. برای محدود کردن





نحوه ایجاد محدودیت برای appDomain

**static** **void** Main**(string[]** args**)**

**{**

//Step1: Create Permission object

var permission = new PermissionSet**(**PermissionState.None**)**;

//Permission for the code to run.

//Execution: Without this permission, Managed Code will not be Executed.

permission.AddPermission**(**

new SecurityPermission**(**SecurityPermissionFlag.Execution**)**

**)**;

//Set No Access to C Drive,

//NoAccess: No access to a file or directory.

permission.AddPermission**(**

new FileIOPermission**(**FileIOPermissionAccess.NoAccess, @"D:\")

);

//Step2: Create setup for App Domain

var setUp = new AppDomainSetup

{

//CurrentDomain: Gets the current application domain

//SetupInformation: Gets the application domain configuration information

//ApplicationBase: Gets or sets the name of the directory containing the application.

ApplicationBase = AppDomain.CurrentDomain.SetupInformation.ApplicationBase

};

//Step3: Create custom App Domain

//Create custom App Domain using the setup and permission

//CreateDomain: Creates a new application domain with the specified name.

AppDomain customDomain = AppDomain.CreateDomain("customDomain", null, setUp, permission);

//Step4:

//Get the Type of ThirdParty using the typeof method by passing the ThirdParty class name

Type thirdParty = typeof(ThirdParty);

//Step5:

//Create an object of ThirdParty using the customDomain i.e. load the ThirdParty

//To Create an Object, we need to call the CreateInstanceAndUnwrap method of customDomain object

customDomain.CreateInstanceAndUnwrap(

//Gets the display name of the assembly.

thirdParty.Assembly.FullName,

//Gets the fully qualified name of the type, including its namespace

//but not its assembly.

thirdParty.FullName);

//Step6:

//Unload the Custom App Domain

AppDomain.Unload(customDomain);

//Own DLL

MyClass1 obj1 = new MyClass1();

MyClass2 obj2 = new MyClass2();

Console.Read();

}

}

مزایا:

ایجاد یک isolated container در process برنامه خود. امکان load , unload – امکان تعریف دسترسی محدود

# **Strong and Weak Assemblies in .NET Framework**

Assembly ها به 2 نوع strong , weak تقسیم بندی می شوند.

زمانی که .net را نصب می کنید تمامی bcl ها یا base class library به همراه Clr در قالب assembly در محلی به نام gac یا global access cache ذخیره می شوند.

تمامی assembly های موجود در gac از نوع Strong هستند.

اسمبلی ها دارای اطلاعات هستند نظیر

1. Simple Textual Name (i.e. Assembly Name).
2. The Version Number.
3. Cultural information (If provided, otherwise the assembly is language-neutral)
4. Public Key Token

**Assembly Name (Simple Textual Name):**

This is nothing but the project name. We have created one console application with the name **AssemblyDemo**. Now build the project and go to the **Bin => Debug** folder of your project and you should find an assembly with the name **AssemblyDemo.**

**Version Number:**

The default format of the Version number is **1.0.0.0**. That means the version number again consists of four parts as follows:

1. **Major Version**
2. **Minor Version**
3. **Build Number**
4. **Revision Number**

Typically, any software we develop will go under changes over a period of time. When we fix bugs or add new features, depending on the significance of the change, we either change the major number or the minor number. If the changes we are making to the application are huge, then probably we change the major number else we will change the minor number. Most of the time the build number and revision number have defaulted.

همچنین اطلاعات ILDASM را نیز می توانیم داشته باشیم

**How to Change the Version Number of an Assembly in .NET Framework?**

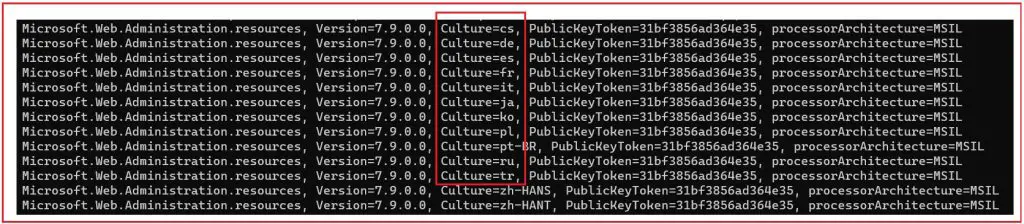
If you want to change the Version Number of your assembly, then you need to use the **AssemblyVersion** attribute within the **AssemblyInfo** class which is present inside the **Properties** folder of your project. You can specify all the values or you can default the Revision and Build Numbers by using the ‘\*’. Suppose, you want to change the Major Number to 3 and the Minor Number to 2, then you need to update the following **AssemblyVersion** attribute of the **AssemblyInfo** class as follows.

**[assembly: AssemblyVersion(“3.2.\*”)]**

With the above changes in place, now if you build the solution and check the version number using the ILDASM tool, then you should see the updated version number. Please read our [**ILDASM and ILASM articles**](https://dotnettutorials.net/lesson/intermediate-language/) to learn more about ILDASM and ILASM.

**Assembly Culture:**

The AssemblyCulture attribute is used for specifying the culture of the assembly. By default in the .NET Framework assemblies are Language-Neutral which means the AssemblyCulture attribute contains Culture=neutral. If you go to the GAC, then you will find most of the assemblies are Culture-neutral. But there could be some assemblies that are culture-specific. For a better understanding, please have a look at the following image which you can also find in your GAC. The following assemblies are specific to the language specified in the Culture attribute.



Satellite assembly

زمانی که برای یک Assembly یک culture در نظر می گیریم غیر از حالت پیش فرض در واقع نوع آن را به Satellite تغیر داده ایم که شامل منابع کد برای یک زبان خاص است. امکان در نظر گرفتن resource برای زبان های متفاوت وجود دارد و آن را در Assembly های مختلف نگه داری کرد و زمانی که لازم باشد کاربر از زبان های مختلف استفاده کند.

**Public Key Token:**

In order to get the Public Key Token, you need to sign your assembly with a Private and Public Key Pair. Now the question is how do I get the Private-Public Key?

**How do I get the Private-Public Key?**

In the .NET Framework, we have a tool called Strong Naming Tool (sn.exe) and we can use this (sn.exe) tool to generate the Private Public Key pair. Again in order to use this tool, we need to use the Developer Command Prompt for Visual Studio. So, open Developer Command Prompt for Visual Studio in Administrator mode and then type **sn.exe -k D:\MyKeyFile.snk**and press enter button as shown in the below image.

**[assembly: AssemblyKeyFile(“D:\\MyKeyFile.snk”)]**

**یک assembly با ویژگی strong name باید دارای ویژگی های زیر باشد.**

1. **The Assembly Name.**
2. **Version Number.**
3. **The Assembly should have been signed with the Private/Public Key Pair.**

نکته : باید در نظر گرفت که زمان که یک Assemby از نوع Strong نیست ممکن است سبب Dll Hell شود همچنین اگر دقت کرده باشید تمامی فایل های gac از نوع strong name است و اگر این گونه نباشد تضمینی برای unique بودن نیست و سبب dll hell problem می شود.

# **How to Install an Assembly into GAC in .NET Framework**

**How to Install an Assembly into GAC in .NET Framework?**

برای نصب یک اسمبلی در gac باید حتما به صورت Strong name باشد در غیر این صورت خطا می دهد.

برای انجام فرآیند strong name کردن مراحل زیر باید طی کنید.

<https://dotnettutorials.net/lesson/how-to-install-an-assembly-into-gac-in-net-framework/>

**DLL Hell Problem and Solution in .NET Framework**

یک مشکل در .net است مربوط به زمانی که 2 اپلیکیشن متفاوت یک assembly را بین خود به اشتراک می گذارند و اگر یکی از آن ها بر روی اسمبلی تغییر ایجاد کند و آن تغییر backward compatible نباشد سبب بروز crash در روند اجرای آپ می شود.

برای شبیه سازی می توان 2 تا کنسول آپ در یک پروژه ایجاد کرد به همراه یک کلاس لایبرری مشترک بین این 2.

**How to Overcome DLL Hell Problem in .NET Framework?**

راه حل این است که فایل اسمبلی را با تکنیک strong name تغییر بدهیم و آن را در global assembly cache قرار بدهیم به این روش ورژن های متفاوتی از یک اسمبلی را در gac می توانیم داشته باشیم که تغییر در آن ها سبب کرش برنامه ما نشود.

Gac برای اشتراک گذاری اسمبلی ها بین برنامه ها ایجاد شده است.

برای تبدیل به strong assembly می توانیم از ابزار داخلی sn.exe استفاده کنیم

**Use the above Assembly in both Applications**

باید از طریق reference manager اسمبلی ایجاد شده را که در Gac قرار دارد reference بدهیم.

حالا اگر نیازمندی جدیدی دارید اسمبلی را تغییر دهید ورژن اسمبلی را عوض کرده در gac نصب کنید و رفرنس بدهید حالا 2 اسمبلی با ورژن های مختلف در gac هستند که بدون خطا کار می کنند.