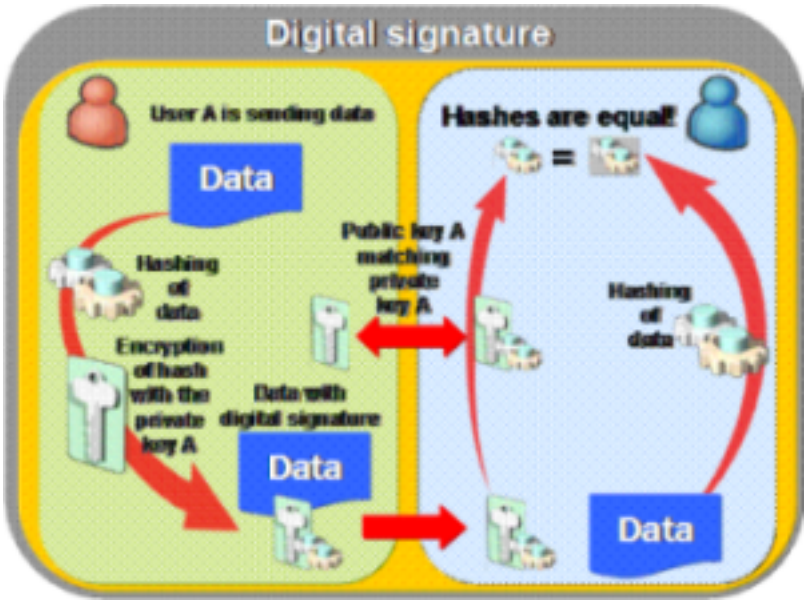
عنوان: **Strong Name**
نویسنده: یوسف نژاد
تاریخ: ۱۳۹۱/۰۸/۲۸ ۱۸:۲۰
آدرس: www.dotnettips.info
برچسب‌ها: .NET, Strong Name

نام قوی (Strong Name یا به‌صورت مخفف SN) تکنولوژی‌ای است که با ورود دانت نت معرفی شده و امکانات متنوعی را در زمینه حفاظت از هویت اسمبلی فراهم کرده است. اما بسیاری از برنامه‌نویسان به اشتباه آن را به‌عنوان ابزاری برای فعال‌سازی امنیت می‌پندارند، درصورتی‌که «نام قوی» درواقع یک تکنولوژی تعیین «هویتِ منحصربه‌فرد» اسمبلی‌ها است. یک نام قوی حاوی مجموعه‌ای از مشخصات یک اسمبلی (شامل نام ساده، نسخه و داده‌های کالچر (culture) آن در صورت وجود) به‌همراه یک کلید عمومی و یک امضای دیجیتال است. در زیر یک نمونه از یک اسمبلی دارای نام قوی را مشاهده می‌کنید:

```
System.Web.Mvc, Version=3.0.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35
```

این نام با استفاده از داده‌های موجود در فایل اصلی یک اسمبلی و نیز یک کلید خصوصی تولید می‌شود. (فایل اصلی اسمبلی فایلی است که حاوی مانیفست اسمبلی است که این مانیفست خود شامل عنوان و هش‌کدهای تمام فایل‌هایی است که اسمبلی را می‌سازند. دات نت از MultiFile Assembly پشتیبانی می‌کند. برای مدیریت این نوع از اسمبلی‌ها می‌توان از Assembly Linker (al.exe) استفاده کرد. البته درحال حاضر امکان توسعه این نوع از اسمبلی‌ها در ویژوال استودیو موجود نیست.) در sdkهای مایکروسافت ابزارهایی برای تولید نام‌های قوی برای اسمبلی‌ها وجود دارد که در ادامه در مورد نحوه استفاده از یک مورد از آن‌ها توضیح داده خواهد شد.
اسمبلی‌هایی که نام‌های قوی یکسانی دارند همانند و یکسان هستند. با اختصاص دادن یک نام قوی به یک اسمبلی می‌توان اطمینان حاصل کرد که نام آن منحصربه‌فرد خواهد شد. به‌طور کلی نام‌های قوی نیازمندی‌های زیر را برطرف می‌کنند:
- نام‌های قوی منحصربه‌فرد بودن نام یک اسمبلی را براساس جفت‌کلیدهای یکتا فراهم می‌کنند. هیچ‌کس دیگری امکان تولید همان اسمبلی‌ای را که شما تولید کرده‌اید ندارد، زیرا اسمبلی‌ای که با یک کلید خصوصی تولید شده است نسبت به اسمبلی دیگری که با یک کلید خصوصی دیگر تولید شده است نام متفاوتی خواهد داشت چون کلید عمومی متناظر با این کلید خصوصی بخشی از نام قوی نهایی تولید شده خواهد بود.
- نام‌های قوی از خط تولید نسخه‌های یک اسمبلی محافظت می‌کنند. یک نام قوی اطمینان می‌دهد تا شخص دیگری نتواند نسخه دیگری از اسمبلی شما را تولید کند. مصرف‌کنندگان می‌توانند مطمئن باشند که نسخه‌ای از اسمبلی را که بارگذاری می‌کنند از همان توزیع‌کننده اسمبلی می‌آید که این نسخه از اسمبلی را تولید کرده است.
- نام‌های قوی بررسی هویت مستحکمی را فراهم می‌کنند. عبور از دروازه امنیتی دات نت فریمورک نشان‌دهنده این است که محتوای اسمبلی پس از تولید آن تغییر نکرده است.
هنگامی‌که به یک اسمبلیِ دارای نام قوی در اسمبلی دیگری ریفرنس داده می‌شود، تا زمانی که به اسمبلی مقصد نیز یک نام قوی داده نشود نمی‌توان در نهایت از مزایای یک نام قوی بهره برد. درواقع در دنیای دات نت به اسمبلی‌های دارای نام قوی تنها می‌توان اسمبلی‌هایی ریفرنس داد که خود نیز دارای نام قوی هستند.
نام قوی یک تکنولوژی براساس اصول کریپتوگرافی و امضاهای دیجیتال است که ایده پایه‌ای آن را می‌توان در تصویر زیر دید:



برای استفاده از این تکنولوژی ابتدا نیاز است تا یک جفت‌کلید عمومی/خصوصی (توسط ادمین، منبع گواهی‌نامه‌ها، یک بانک یا یک ابزار خاص) فراهم شود تا از آن برای اینکریپشن استفاده شود. سپس داده‌های موردنظر (هر داده کلی که قصد ارسال و توزیع آن را داریم مثل یک اسمبلی) با استفاده از یک الگوریتم هش‌کردن (مثل MD5، SHA یا ترکیبی از آن‌ها، هرچند MD5 توصیه نمی‌شود) پردازش شده و یک هش‌کد مخصوص تولید می‌شود. این هش‌کد با استفاده از کلید خصوصی دردسترس اینکریپت می‌شود و به عنوان یک امضای دیجیتال به همراه داده موردنظر ارسال یا توزیع می‌شود. در سمت مصرف کننده که با استفاده از یک روش خاص و امن به کلید عمومی دسترسی پیدا کرده است عملیات دیکریپت کردن این امضای دیجیتال با استفاده از کلید عمومی انجام شده و هش‌کد مربوطه بدست می‌آید. همچنین عملیات تولید هش‌کد با استفاده از داده‌ها در سمت مصرف کننده انجام شده و هش‌کد داده‌ها نیز دوباره با استفاده از همان الگوریتم استفاده شده در سمت توزیع‌کننده تولید می‌شود. سپس این دو مقدار محاسبه شده در سمت مصرف‌کننده با یکدیگر مقایسه شده و درصورت برابر بودن می‌توان اطمینان حاصل کرد همان داده‌ای که توزیع کننده در اصل ارسال کرده بدون تغییر به دست مصرف کننده رسیده است. درواقع ویژگی اینکریپت/دیکریپت کردن داده‌ها توسط جفت‌کلید این است که به‌صورت یکطرفه بوده و داده‌های اینکریپت شده با استفاده از یک کلید خصوصی را تنها با استفاده از کلید عمومی همان کلید خصوصی می‌توان بدرستی دیکریپت کرد.

1. تولید و مدیریت جفت‌کلیدهای قوی- نام‌گذاری‌شده (Strongly Named Key Pairs)

همان‌طور که در قسمت قبل اشاره شد برای نام‌گذاری قوی یک اسمبلی به یک کلید عمومی (public key) و یک کلید خصوصی (private key) که در مجموع به آن یک جفت کلید (key pair) می‌گویند، نیاز است.برای این‌کار می‌توان با استفاده از برنامه sn.exe (عنوان کامل آن Microsoft .Net Framework Strong Name Utility است) یک جفت کلید تولید کرده و آن را در یک فایل و یا در CSP (یا همان cryptographic service provider) ذخیره کرد. هم‌چنین این‌کار را می‌توان توسط ویژوال استودیو نیز انجام داد. امکان موردنظر در فرم پراپرتی یک پروژه و در تب Signing آن وجود دارد.

نکته : یک CSP عنصری از API کریپتوگرافی ویندوز (Win32 CryptoAPI) است که سرویس‌هایی چون اینکریپشن، دیکریپشن، و تولید امضای دیجیتال را فراهم می‌کند. این پرووایدرها هم‌چنین تسهیلاتی برای مخازن کلیدها فراهم می‌کنند که از اینکریپشن‌های قوی و ساختار امنیتی سیستم عامل (سیستم امنیتی و دسترسی کاربران ویندوز) برای محافظت از تمام کلیدهای کریپتوگرافی ذخیره شده در مخزن استفاده می‌کند. به‌طور خلاصه و مفید می‌شود اشاره کرد که می‌توان کلیدهای کریپتوگرافی را درون یک مخزن کلید CSP ذخیره کرد و تقریبا مطمئن بود که تا زمانی‌که هیچ‌کس کلمه عبور سیستم عامل را نداند، این کلیدها امن خواهند ماند. برای کسب اطلاعات بیشتر به داده‌های CryptoAPI در اسناد SDK سیستم عامل خود مراجعه کنید.

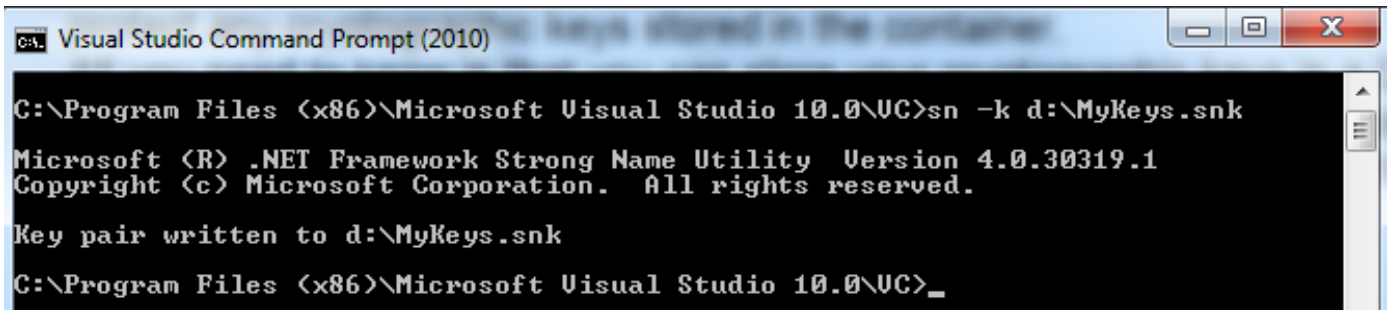
برنامه sn به همراه SDKهای ویندوز نصب می‌شود. البته با نصب ویژوال استودیو تمام SDKهای موردنیاز مطابق با نسخه‌های

موجود، نصب خواهد شد. مسیر نسخه 4 و 32 بیتی این برنامه در سیستم عامل Windows 7 به‌صورت زیر است:

```
C:\Program Files\Microsoft SDKs\Windows\v7.0A\Bin\NETFX 4.0 Tools\sn.exe
```

با استفاده از آرگومان k همانند دستور زیر یک جفت‌کلید جدید تولید شده و در فایل MyKeys.snk در ریشه درایو d: ذخیره می‌شود:

```
sn -k d:\MyKeys.snk
```



نکته : به بزرگی و کوچکی حروف سوییچ‌های دستورات برنامه sn دقت کنید!

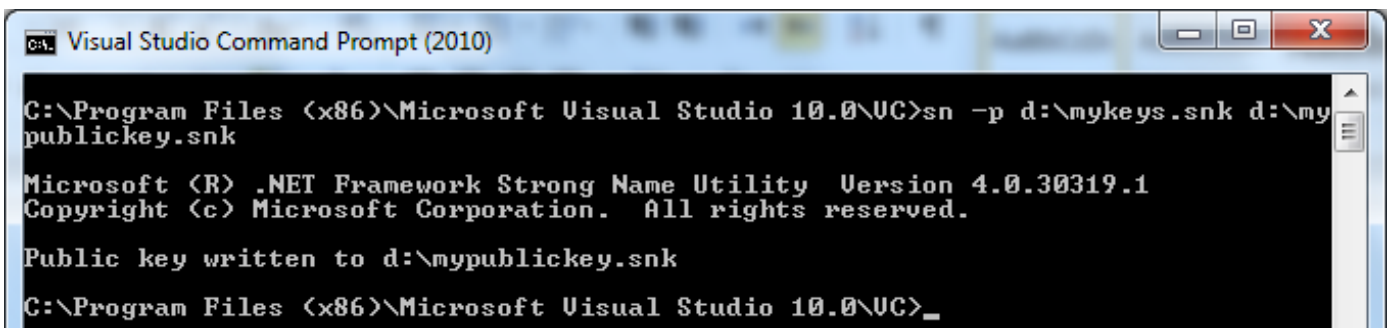
این کار یک جفت کلید کریپتوگرافی 1024 بیتی به‌صورت تصادفی تولید می‌کند. این دستور را باید در خط فرمانی (Command Prompt) اجرا نمود که مسیر فایل sn.exe را بداند. برای راحتی کار می‌توان از خط فرمان ویژوال استودیو (Visual Studio Command Prompt) استفاده کرد.

نکته : اجرای عملیات فوق در یک شرکت یا قسمت توسعه یک شرکت، تنها یک بار نیاز است زیرا تمام اسمبلی‌های تولیدی تا زمانی‌که عناوین ساده متمایزی دارند می‌توانند از یک جفت کلید مشترک استفاده کنند.

نکته : هرچند که می‌توان از پسوندهای دیگری نیز برای نام فایل حاوی جفت کلید استفاده کرد، اما توصیه می‌شود از همین پسوند **snk.** استفاده شود.

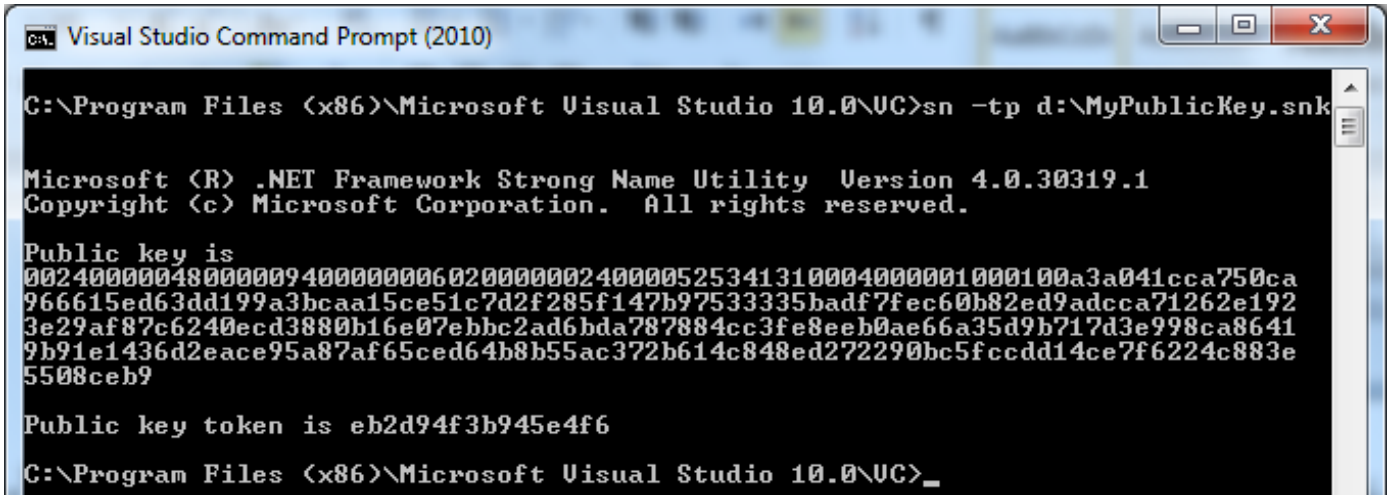
فایل تولید شده حاوی هر دو کلید «عمومی» و «خصوصی» است. می‌توان با استفاده از دستور زیر کلید عمومی موجود در فایل mykeys.snk را استخراج کرده و در فایل mypublickey.snk ذخیره کرد:

```
sn -p d:\mykeys.snk d:\mypublickey.snk
```



با استفاده از فایل حاوی کلید عمومی می‌توان با استفاده از دستور زیر کلید عمومی موجود در آن را بدست آورد:

```
sn -tp MyPublicKey.snk
```



مقدار نمایش داده در انتهای تصویر فوق به‌عنوان «توکِن کلید عمومی» (Public key Token) درواقع 8 بایت پایانی کد هش‌شده کریپتوگرافیِ محاسبه‌شده از کلید عمومی است. چون خود کلید عمومی همان‌طور که مشاهده می‌شود بسیار طولانی است، دات‌نت‌فریمورک معمولا از این توکِن برای نمایش آن و ریفرنس دادن اسمبلی‌ها استفاده می‌کند. نیازی نیست تا راز این کلیدها توسط توسعه‌دهنده حفظ شود! پس از نام‌گذاری قوی اسمبلی (که در ادامه توضیح داده می‌شود) کامپایلر با استفاده از کلید خصوصی فراهم شده یک امضای دیجیتالی (یک کد اینکریپت شده) با استفاده از داده‌های «مانیفست اسمبلی» تولید می‌کند. در ادامه کامپایلر این «امضای دیجیتال» و «کلید عمومی» را درون اسمبلی قرار می‌دهد تا مصرف‌کننده‌های اسمبلی بتوانند این امضای دیجیتال را تایید کنند. حفظ کردن «کلید خصوصی» بسیار مهم است! اگر کسی به کلید خصوصی اسمبلی دست یابد می‌تواند با استفاده از آن نسخه‌ای تغییریافته از اسمبلی را امضا کرده و در اختیار مصرف‌کنندگان قرار دهد. مصرف‌کنندگان نیز بدون اینکه متوجه شوند می‌توانند از این نسخه تغییر یافته با همان توکِن کلید عمومی که در اختیار دارند استفاده کنند. درحال حاضر روشی برای فهمیدن این تغییر وجود ندارد. اگر کلید خصوصی لو رفت، باید یک جفت کلید دیگر تولید و با استفاده از کلید خصوصی جدید اسمبلی را دوباره امضا کرد و در اختیار مصرف‌کنندگان قرار داد. هم‌چنین باید مشتریانِ اسمبلی را از این تغییر آگاه ساخت و کلید عمومی مورد اطمینان را در اختیار آن‌ها قرار داد.

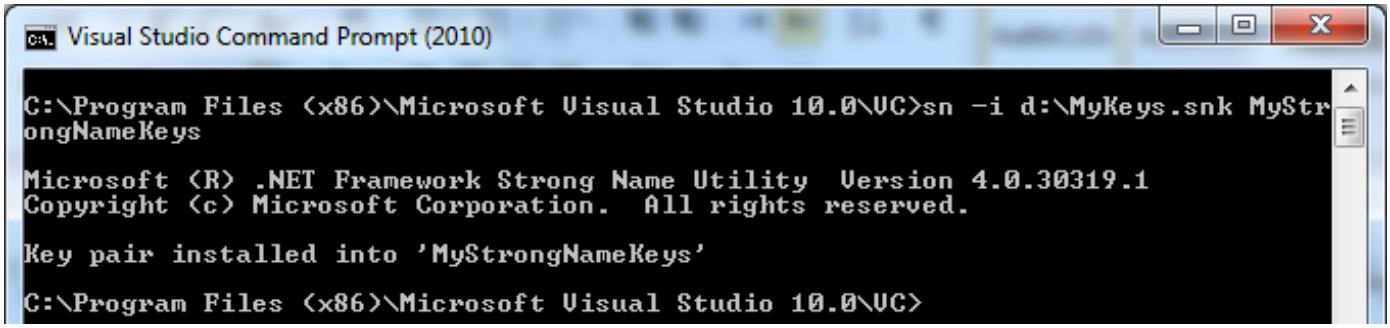
نکته : معمولا گروه کوچکی از افراد مورد اطمینان (که دسترسی امضای اسمبلی را دارند: signing authority) مسئولیت کلیدهای نامگذاری قوی یک شرکت را بر عهده دارند و برای امضای تمام اسمبلی‌ها قبل از ریلیز نهایی آن‌ها مسئول هستند.

قابلیت امضای تاخیری اسمبلی (که در ادامه بحث می‌شود) تسهیلاتی را برای بهره‌برداری راحت‌تر از این روش و جلوگیری از توزیع کلیدهای خصوصی میان تمام توسعه‌دهندگان را فراهم می‌کند.

یکی از روش‌هایی که sn برای افزایش امنیت کلیدها ارائه می‌دهد، استفاده از مخزن کلید CSP است. پس از تولید فایل حاوی جفت کلید، می‌توان با استفاده از دستور زیر این کلیدها را درون CSP با نام MyStrongNameKeys ذخیره کرد:

```
sn -i MyKeys.snk MyStrongNameKeys
```

سپس می‌توان فایل حاوی جفت کلید را حذف کرد.

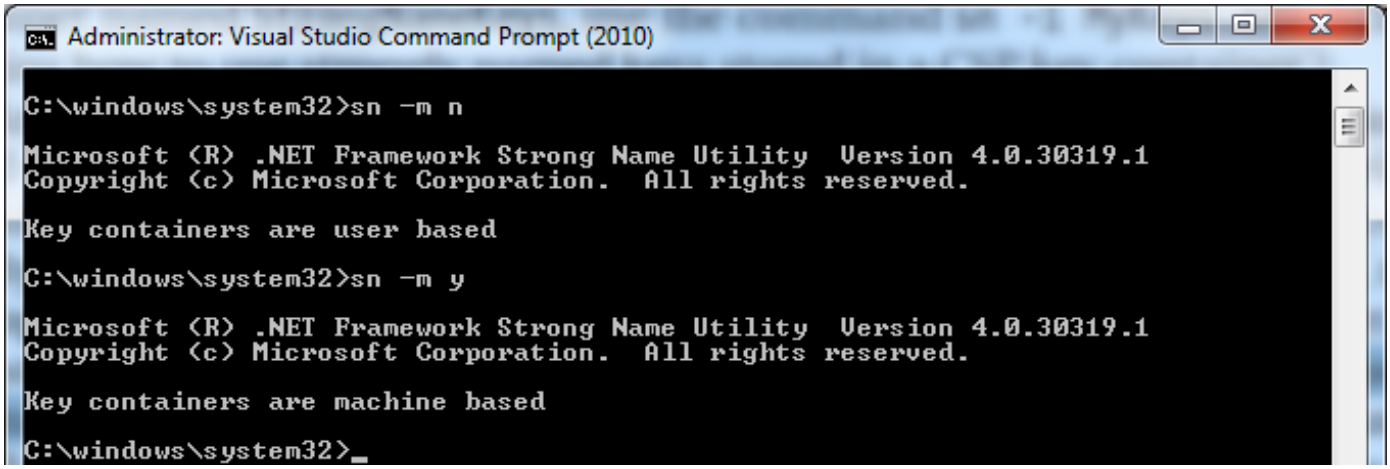


نکته مهمی که درباره مخازن کلید CSP باید بدان اشاره کرد این است که این مخازن شامل مخازن تعریف‌شده توسط «کاربر» و نیز مخازن «سیستمی» است. سیستم امنیتی ویندوز به کابران اجازه دسترسی به مخازنی غیر از مخازن خودشان و مخازن سیستمی را نمی‌دهد. برنامه sn به‌صورت پیش‌فرض کلیدها را درون مخازن سیستمی ذخیره می‌کند. بنابراین هر کسی که بتواند به سیستم لاگین کند و نیز از نام مخزن مربوطه آگاه باشد، به‌راحتی می‌تواند اسمبلی شما را امضا کند! برای اینکه ابزار sn کلیدها را در مخازن کاربری ذخیره کند باید از دستور زیر استفاده کرد:

```
sn -m n
```

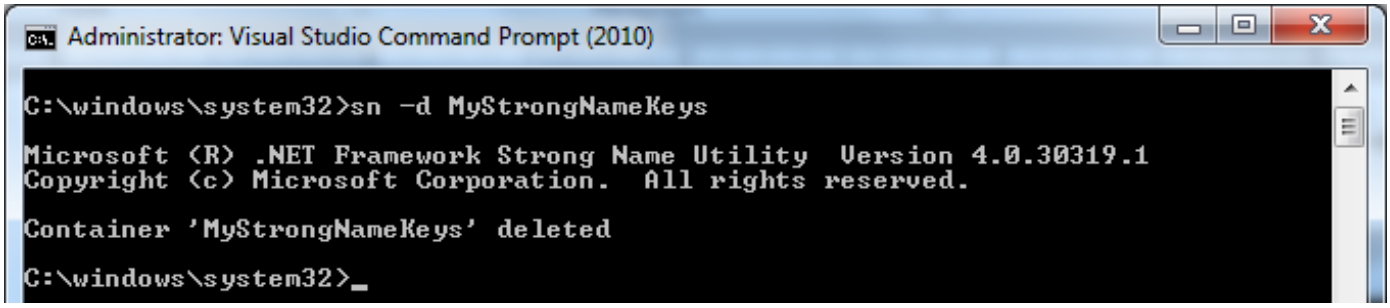
برای برگرداند تنظیم به ذخیره در مخازن سیستمی نیز باید از دستور زیر استفاده کرد:

```
sn -m y
```



برای حذف کلیدها از مخزن می‌توان از دستور زیر استفاده کرد:

```
sn -d MyStrongNameKeys
```

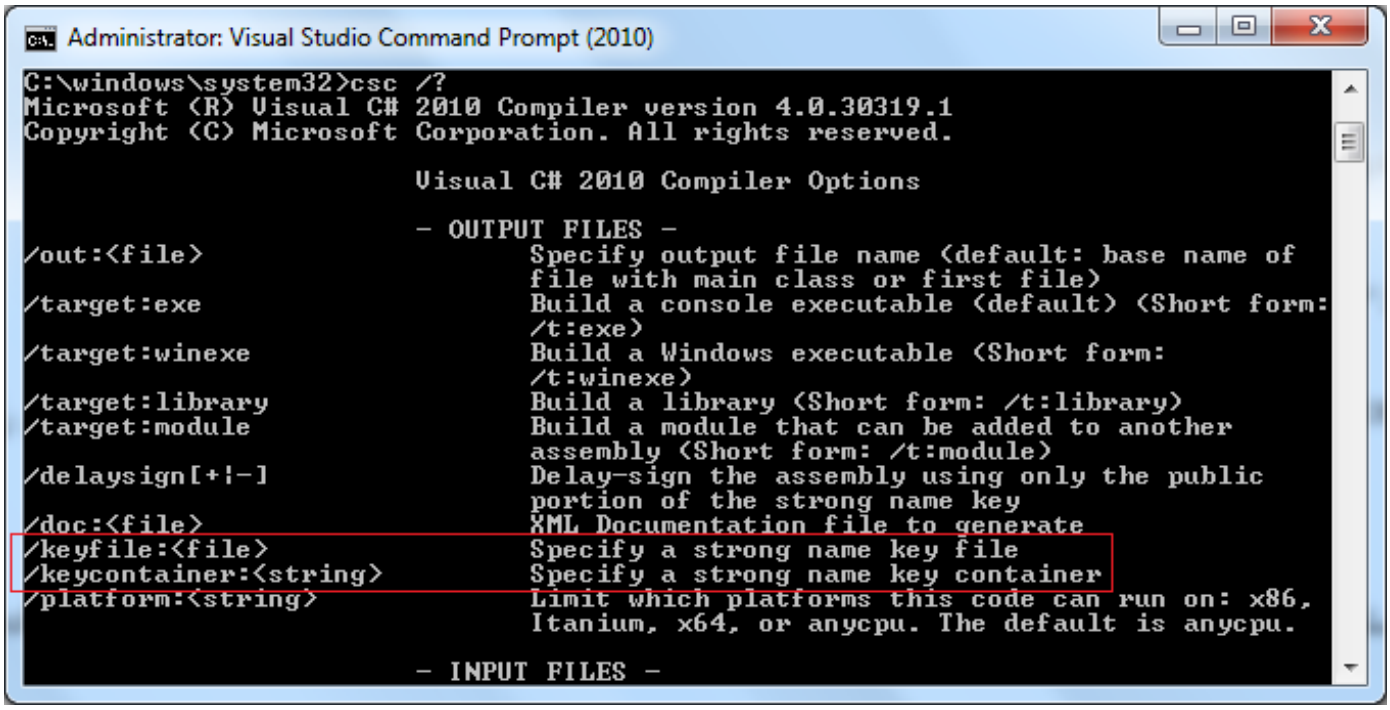


2. نام‌گذاری قوی یک اسمبلی

نام‌گذاری قوی یک اسمبلی به دلایل زیادی انجام می‌شود:

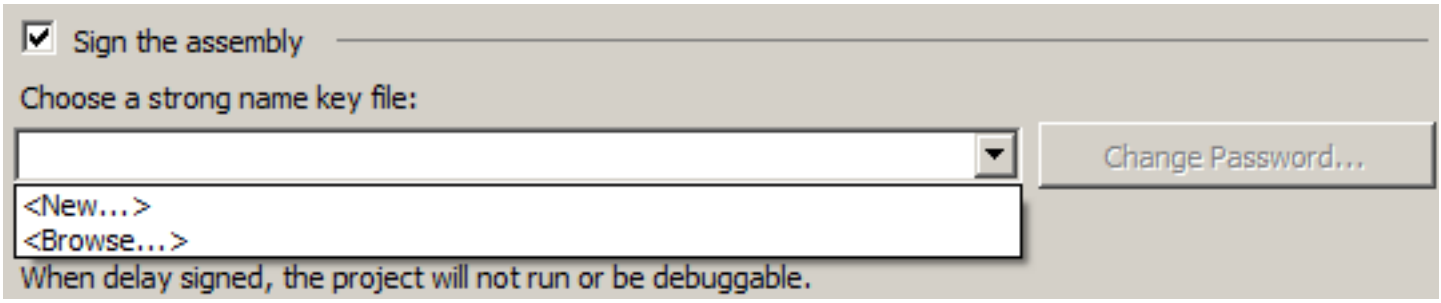
- برای اینکه اسمبلی شناسه‌ای منحصربه‌فرد داشته باشد، تا کاربران بتوانند مجوزهای ویژه‌ای را در حین تنظیم سیاست‌های امنیتی دسترسی به کد اعمال کنند.
- تا اسمبلی را نتوان تغییر داده و سپس به عنوان اسمبلی اصلی توزیع نمود.
- تا اسمبلی بتواند نسخه‌گذاری (Versioning) و سیاست‌های نسخه‌گذاری را پشتیبانی کند.
- تا بتوان اسمبلی را در GAC (همان Global Assembly Cache که در مسیر windir%\assembly% قرار دارد) ذخیره کرده و آن را بین چند اپلیکیشن به اشتراک گذاشت.

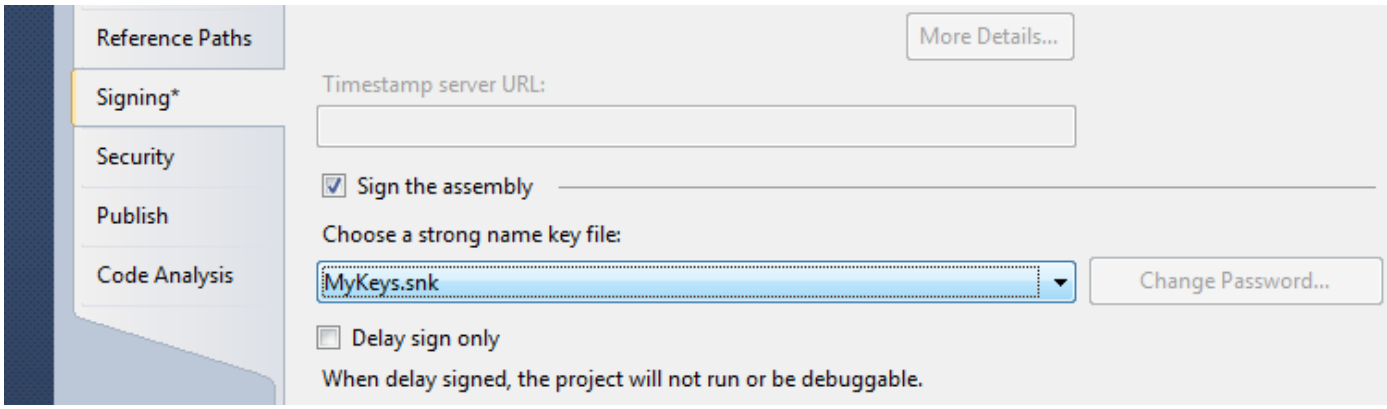
برای نام‌گذاری قوی اسمبلی با استفاده از خط فرمان کامپایلر C# باید از سوییچهای keyfile/ و یا keycontainer/ استفاده کنید.



"csc /keyfile:d:\mykeys.snk /out:"C:\Projects\ClassLibrary1\Class1.exe" "C:\Projects\ClassLibrary1\Class1.cs

نکته : برای استفاده از این ویژگی در ویژوال استودیو، باید در تب Signing در تنظیمات پروژه گزینه Sign the Assembly را انتخاب کرد. سپس می‌توان فایل حاوی جفت کلیدهای تولیدشده را انتخاب یا فایل جدیدی تولید کرد. البته ویژوال استودیو تا نسخه 2010 امکانی جهت استفاده از مخازن CSP را ندارد.





روش ساده دیگر استفاده از attributeهای سطح اسمبلی است:

```
[assembly:AssemblyKeyFileAttribute("MyKeys.snk")]
```

3. بررسی اینکه آیا یک اسمبلی قوی-نام‌گذاری‌شده تغییر یافته یا خیر

زمانی‌که CLR در زمان اجرا یک اسمبلی قوی-نام‌گذاری‌شده را بارگذاری می‌کند:

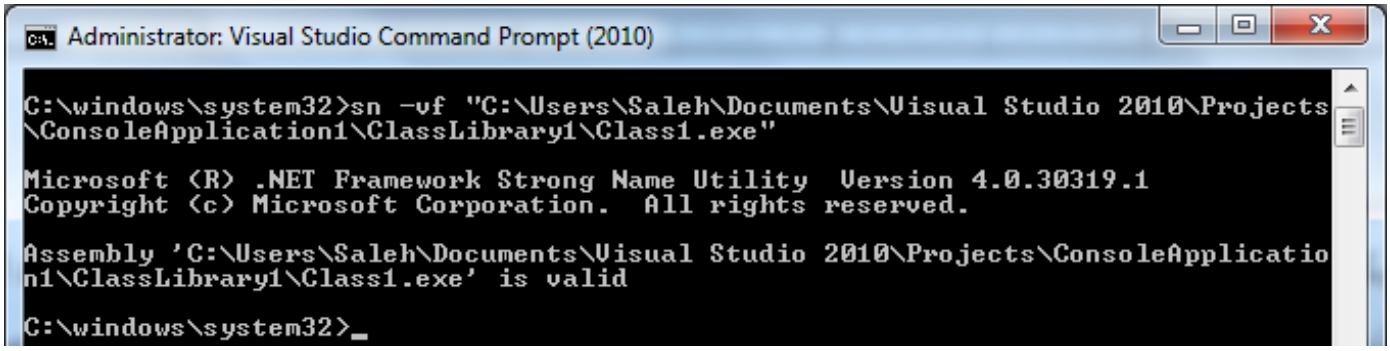
-ابتدا با استفاده از کلید عمومی (که در خود اسمبلی ذخیره شده است) هش‌کد اینکریپت‌شده که در زمان کامپایل محاسبه شده (یا همان امضای دیجیتال که این نیز درون خود اسمبلی ذخیره شده است) را دیکریپت می‌کند. (هش‌کد زمان کامپایل)

-پس از آن هش‌کد اسمبلی را با استفاده از داده‌های مانیفست اسمبلی محاسبه می‌کند. (هش‌کد زمان اجرا)

-سپس این دو مقدار بدست آمده (هش‌کد زمان کامپایل و هش‌کد زمان اجرا) را با یکدیگر مقایسه می‌کند. این عملیات مقایسه و تایید مشخص می‌کند که آیا اسمبلی پس از امضا دچار تغییر شده است یا خیر!

اگر یک اسمبلی نتواند عملیات تایید نام قوی را پشت سر بگذارد، CLR پیغام خطایی به نمایش خواهد گذاشت. این خطا یک اکسپشن از نوع System.IO.FileLoadException با پیغام Strong name validation failed خواهد بود. با استفاده از ابزار sn نیز می‌توان یک اسمبلی قوی-نام‌گذاری شده را تایید کرد. برای مثال برای تایید اسمبلی MyAsm.exe می‌توان از دستور زیر استفاده کرد:

```
sn –vf MyAsm.exe
```



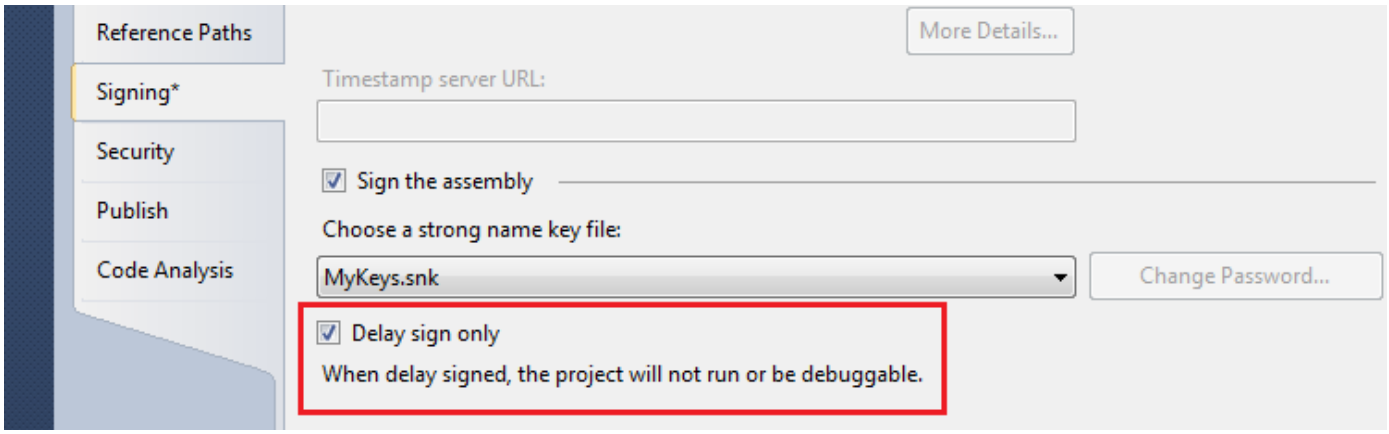
سوییچ v موجب تایید نام قوی اسمبلی شده و سوییچ f برنامه را مجبور به بررسی صحت نام قوی اسمبلی می‌کند، حتی اگر این امکان قبلا برای اسمبلی غیرفعال شده باشد. (با استفاده از سویچ Vr مثل دستور sn -Vr MyAsm.exe می‌توان عملیات تایید نام قوی یک اسمبلی خاص را غیرفعال کرد). اگر اسمبلی تغییر کرده باشد و نتواند آزمون فوق را پشت سر بگذارد خطایی به شکل زیر نمایش داده می‌شود:

```
Microsoft (R) .NET Framework Strong Name Utility Version 2.0.50727.42
Copyright (C) Microsoft Corporation. All rights reserved.
Failed to verify assembly --
Strong name validation failed for assembly MyAsm.exe'.
```

4. امضای تاخیری (Delay Sign) یک اسمبلی

درصورتی‌که بخواهیم یک اسمبلی را امضا کنیم اما نخواهیم تمام اعضای تیم توسعه به کلید خصوصی مربوطه دسترسی داشته باشند باید از تکنیک امضای با تاخیر اسمبلی استفاده کنیم. ابتدا باید کلید عمومی تولیدشده برای اسمبلی را استخراج کرده و آنرا توزیع کنیم. با توجه به توضیحات داده شده در بخش اول، به اسمبلی خود یک نام قوی اختصاص دهید. هم‌چنین اسمبلی خود را با استفاده از سویچ /delaysign باید کامپایل کنید. سپس با استفاده از سوییچ Vr برنامه sn عملیات تایید اسمبلی خود را غیرفعال کنید.

نکته : برای استفاده از این امکان در ویژوال استودیو باید گزینه Delay sign only را در تب Signing از پراپرتی پروژه انتخاب کرد.



اسمبلی‌هایی که ریفرنسی به اسمبلی‌های نام‌گذاری قوی شده دارند، حاوی توکِن کلید عمومی آن اسمبلی‌ها نیز هستند. این بدین معنی است که این گونه اسمبلی‌ها بایستی قبل از ریفرنس داده شدن امضا شده باشند. در یک محیط توسعه که اسمبلی‌ها مرتبا کامپایل می‌شوند نیاز است تا تمام توسعه دهندگان و آزمایش‌کنندگان به جفت‌کلیدهای موجود دسترسی داشته باشند (یک ریسک امنیتی بزرگ). به جای توزیع کلید خصوصی، دات‌نت‌فریمورک مکانیزمی به نام امضای تاخیری (delay-signing) فراهم کرده است، که به شما اجازه می‌دهد تا یک اسمبلی را به‌صورت ناکامل (ناقص) امضا کنید. اسمبلی «ناقص-نام‌گذاریِ قوی شده»! حاوی

کلید عمومی و توکِن کلید عمومی است که برای ریفرنس دادن اسمبلی نیاز است، اما تنها حاوی مکانِ خالیِ امضای دیجیتالی است که توسط کلید خصوصی تولید می‌شود. پس از کامل شدن توسعهٔ برنامه، فرد مسئول امضای اسمبلی‌ها (signing authority - شخصی که مسئول امنیت و حفظ جفت‌کلیدهاست) اسمبلی‌های حاوی امضای تاخیری را دوباره امضا می‌کند، تا نام‌گذاریِ قوی آن اسمبلی کامل شود. برای امضای تاخیری یک اسمبلی تنها نیاز به کلید عمومی آن است، که هیچ ریسک امنیتی‌ای برای آن وجود ندارد. برای استخراج کلید عمومی یک جفت کلید همان‌طور که قبلا اشاره شده است، می‌توان از دستورات زیر استفاده کرد:

```
sn -p d:\MyKeys.snk d:\MyPublicKey.snk
sn -pc MyKeysContainer d:\MyPublicKey.snk
```

با داشتن فایل حاوی کلید عمومی، و با استفاده از از دستور کامپایل زیر می‌توان اسمبلی را امضای تاخیری کرد:

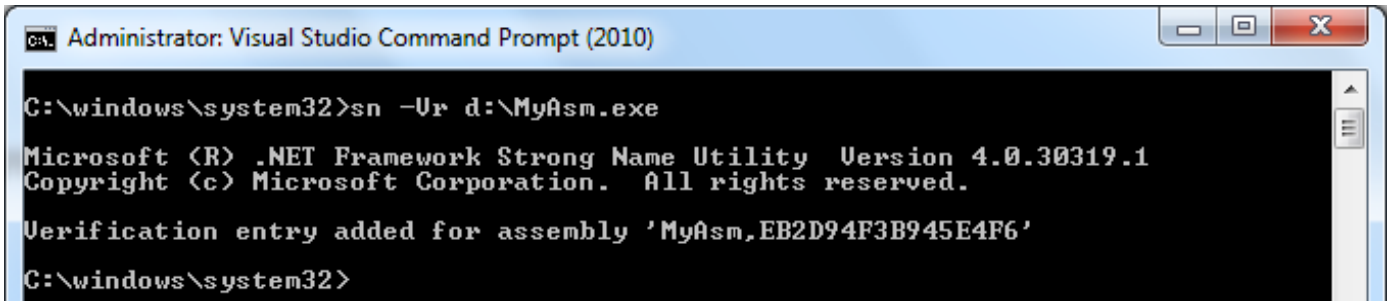
```
csc.exe /delaysign /keyfile:d:\MyPublicKey.snk /out:d:\MyAsm.exe d:\Class1.cs
```

نکته : برای امضای اسمبلی‌های چندفایلی (multifile assembly) باید از Assembly Linker (نام فایل اجرایی آن al.exe است) استفاده کرد. این ابزار نیز مانند ابزار sn.exe در sdkهای ویندوز یافت می‌شود. دستوری که باید برای امضای این نوع اسمبلی‌های به‌کار برد به‌صورت زیر است:

```
al /out:<assembly name> <module name> /keyfile:<file name>
```

از آنجاکه درهنگام بارگذاری اسمبلی، CLR اسمبلی را به عنوان یک اسمبلی قوی نام‌گذاری شده درنظر می‌گیرد، همان‌طور که قبلا اشاره شده، سعی می‌کند تا صحت آن را بررسی و تایید کند. اما چون اسمبلی با امضای تاخیری هنوز امضا نشده است، باید CLR را جوری تنظیم کنید تا تایید اعتبار این اسمبلی را در کامپیوتر جاری انجام ندهد. این کار را همان‌طور که در بالا توضیح داده شد، می‌توان با دستور زیر انجام داد:

```
sn -Vr d:\MyAsm.exe
```

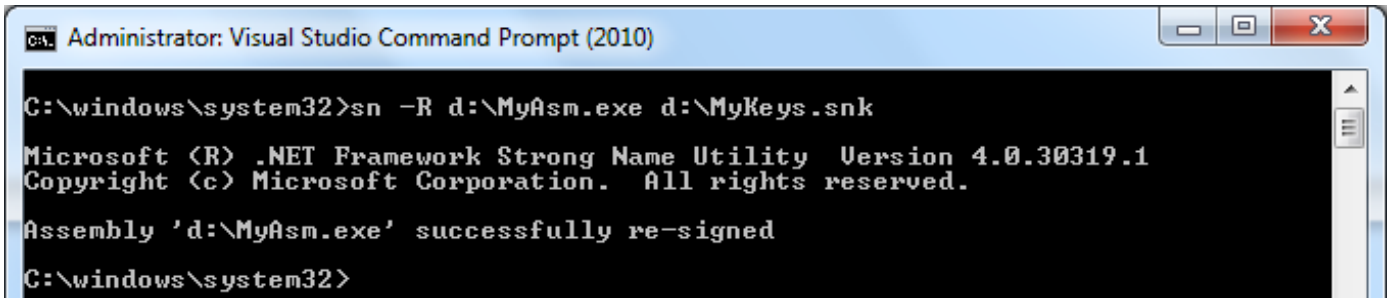


از لحاظ فنی این دستور اسمبلی موردنظر را در لیست «صرف‌نظر از تایید اسمبلی» ثبت (register) می‌کند. دقت کنید که دستور فوق را باید در تمام سیستم‌هایی که قرار است به نحوی با این اسمبلی سروکار داشته باشند اجرا کنید!

نکته : تا زمانی‌که با استفاده از دستور فوق عملیات تایید اعتبار اسمبلی‌های امضای تاخیری شده را غیرفعال نکنید امکان اجرا یا بارگذاری آن اسمبلی‌ها و نیز دیباگ سورس‌کدهای آن را نخواهید داشت!
پس از تکمیل فاز توسعه باید اسمبلی را دوباره امضا کنید تا نام‌گذاری قوی کامل شود. برنامه sn به شما این امکان را می‌دهد تا بدون تغییر سورس‌کد اسمبلی خود یا کامپایل دوباره آن عملیات امضای دوباره آنرا انجام دهید. اما برای این‌کار شما باید به کلید خصوصی آن (در واقع به فایل حاوی جفت‌کلید مربوطه) دسترسی داشته باشید. برای امضای دوباره می‌توان از دستورات زیر استفاده کرد:

```
sn -R d:\MyAsm.exe MyKeys.snk
```

```
sn -R d:\MyAsm.exe MyKeysContainer
```



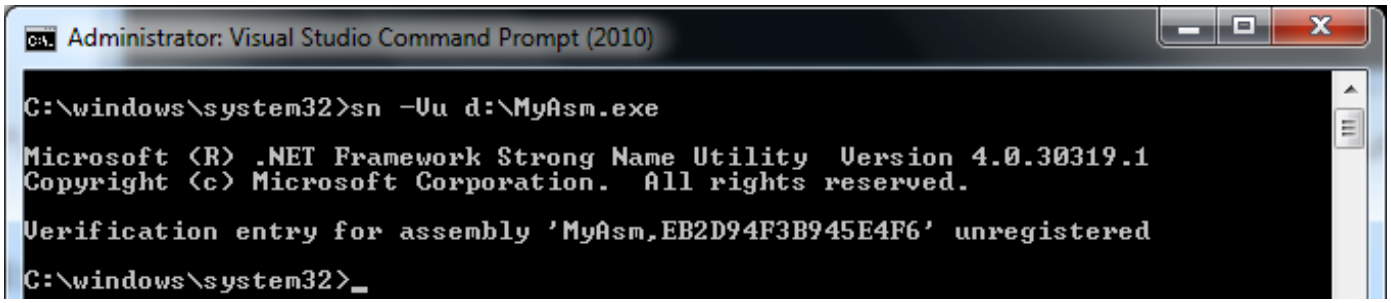
با استفاده از این دستور برنامه sn شروع به محاسبه هش‌کد زمان کامپایل می‌کند و درنهایت مقدار اینکریپت‌شده را درون اسمبلی ذخیره می‌کند.

نکته : هنگام استفاده از اسمبلی‌های با امضای تاخیری، امکان مقایسه بیلدهای مختلف یک اسمبلی خاص برای اطمینان از اینکه تنها در امضای دیجیتال با هم فرق دارند، معمولا مفید است. این مقایسه تنها وقتی امکان‌پذیر است که اسمبلی موردنظر با استفاده از سوییچ R دوباره امضا شود. برای مقایسه دو اسمبلی می‌توان از سوییچ D استفاده کرد:

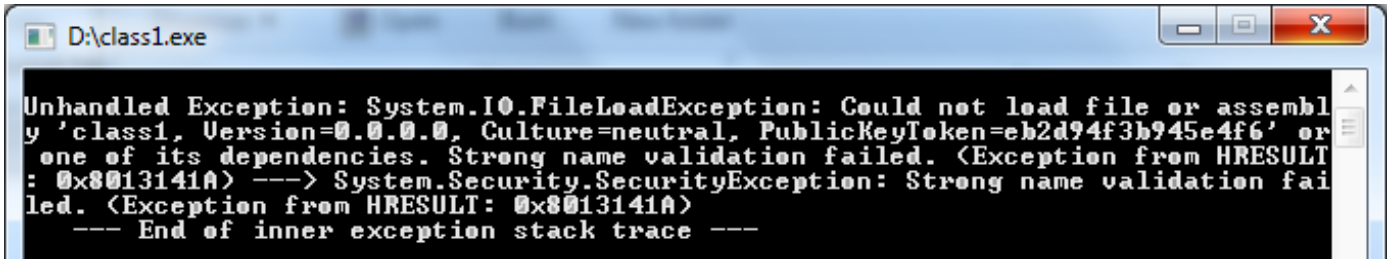
```
sn -D assembly1 assembly2
```

پس از امضای دوباره اسمبلی می‌توان عملیات تایید آنرا که قبلا غیرفعال شده است، با استفاده از دستور زیر دوباره فعال کرد:

```
sn -Vu d:\MyAsm.exe
```



دستور فوق اسمبلی موردنظر را از لیست «صرفنظر از تایید اسمبلی» حذف (Unregister) می‌کند.
نکته : درصورتی‌که بخواهید یک اسمبلی را قبل از امضای دوباره (و یا در حالت کلی، قبل از اینکه اسمبلی دارای یک نام قوی کامل شده باشد) اجرا یا از آن به عنوان یک ریفرنس استفاده کنید، بدون اینکه آن را به لیست «صرفنظر از تایید اسمبلی» اضافی کنید، با خطای زیر مواجه خواهید شد:



برای فعال‌سازی تایید اسمبلی برای تمامی اسمبلی‌هایی که این ویژگی برای آنان غیرفعال شده است، می‌توانید از دستور زیر استفاده کنید:

```
sn -Vx
```

برای لیست کردن اسمبلی‌هایی که تایید آنان غیرفعال شده است، می‌توانید از دستور زیر استفاده کنید:

```
sn -Vl
```

نکته : در دات‌نت 1.0 و 1.1 کامپایلر C# فاقد سوییچ delaysign/ است. برای استفاده از امکان امضای تاخیری اسمبلی می‌توان از attribute سطح اسمبلی System.Reflection.AssemblyDelaySignAttribute استفاده کرد. همچنین می‌شود از ابزار لینکر اسمبلی (al.exe) که از این سوییچ پشتیبانی می‌کند استفاده کرد.
نکته : ابزارهای obfuscating که برای پیچیده‌کردن کد IL اسمبلی تولیدی به‌منظور جلوگیری از عملیات تولید دوباره کد (مثل کاری که برنامه Reflector انجام می‌دهد) به‌کار می‌روند، به دلیل تغییراتی که در محتوای اسمبلی ایجاد می‌کنند، درصورتیکه برای اسمبلی‌های دارای نام قوی استفاده شوند موجب ازکار افتادن آن‌ها می‌شوند. بنابراین یا باید آن‌ها را در سیستم‌هایی استفاده کرد که آن اسمبلی موردنظر در لیست صرفنظر از تایید اسمبلی ثبت شده باشد یا اینکه اسمبلی مربوطه را دوباره با استفاده از روش‌های توضیح داده‌شده (مثلا با استفاده از دستور sn -R myAsm.dll MyKeys.snk) برای تخصیص نام قوی جدید امضا کرد. الگوی معمولی که برای استفاده از obfuscating برای اسمبلی‌های دارای نام قوی استفاده می‌شود به‌صورت زیر است:
- ساخت اسمبلی با امضای تاخیری
- افزودن اسمبلی به لیست صرفنظر از تایید اسمبلی (sn -Vr)
- دیباگ و تست اسمبلی
- obfuscate کردن اسمبلی
- دیباگ و تست اسمبلی obfuscate شده
- امضای دوباره اسمبلی (sn -R)
الگوی ساده‌تر دیگری نیز برای این منظور استفاده می‌شود که به‌صورت زیر است:
- تولید اسمبلی بدون استفاده از تنظیمات امضای تاخیری
- دیباگ و تست اسمبلی
- obfuscate اسمبلی
- امضای دوباره اسمبلی (sn -R)
- دیباگ و تست دوباره نسخه obfuscate شده

5. مدیریت کش عمومی اسمبلی‌ها (Global Assembly Cache)

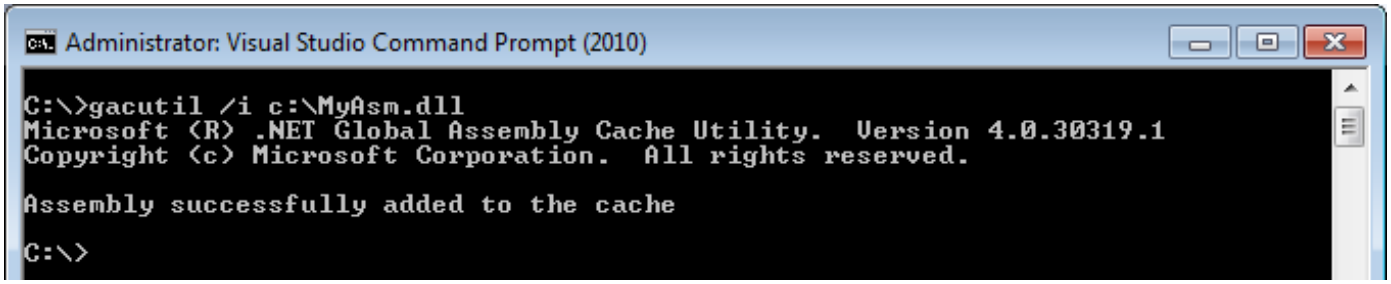
با استفاده از توضیحات این بخش می‌توان اسمبلی‌ها را به GAC اضافه و یا از درون آن حذف کرد. این کار با استفاده از برنامه gacutil.exe انجام می‌شود. مسیر نسخه 4 و 32 بیتی این برنامه به‌صورت زیر است:

```
C:\Program Files\Microsoft SDKs\Windows\v7.0A\Bin\NETFX 4.0 Tools\gacutil.exe
```

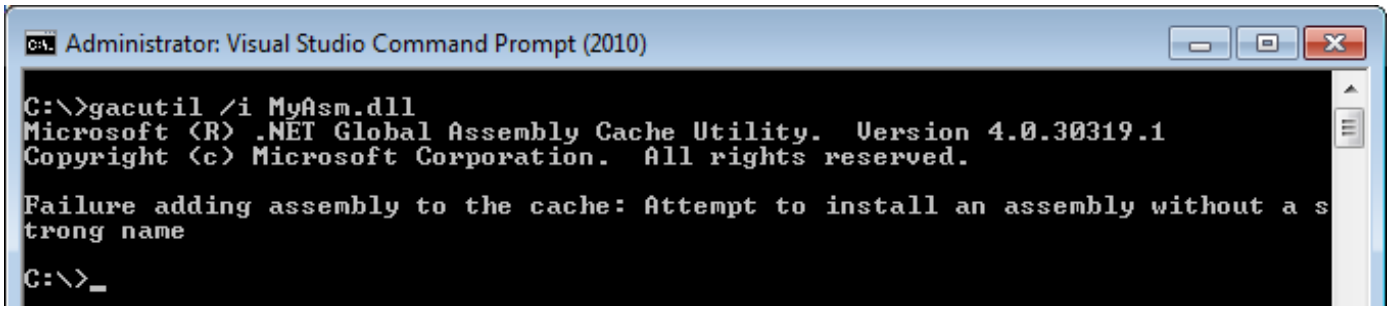
این برنامه به‌همراه SDK ویندوز و یا به‌همراه ویژوال استودیو در مسیری مشابه نشانی بالا نصب می‌شود. همانند توضیحات

داده‌شده در مورد برنامه sn.exe، برای راحتی کار می‌توانید از خط فرمان ویژه‌ای که ویژوال استودیو در اختیار شما قرار می‌دهد استفاده کنید. البته قبل از اجرای هر دستوری مطمئن شوید که خط فرمان شما با استفاده از مجوز مدیریتی (Administrator) اجرا شده است! تنها اسمبلی‌های دارای نام قوی می‌توانند در GAC نصب شوند. بنابراین قبل افزودن یک اسمبلی به GAC باید طبق راهنمایی‌های موجود در قسمت‌های قبلی آن را به‌صورت قوی نام‌گذاری کرد. برای افزودن یک اسمبلی با نام MyAsm.dll می‌توان از دستور زیر استفاده کرد:

```
gacutil /i c:\MyAsm.dll
```



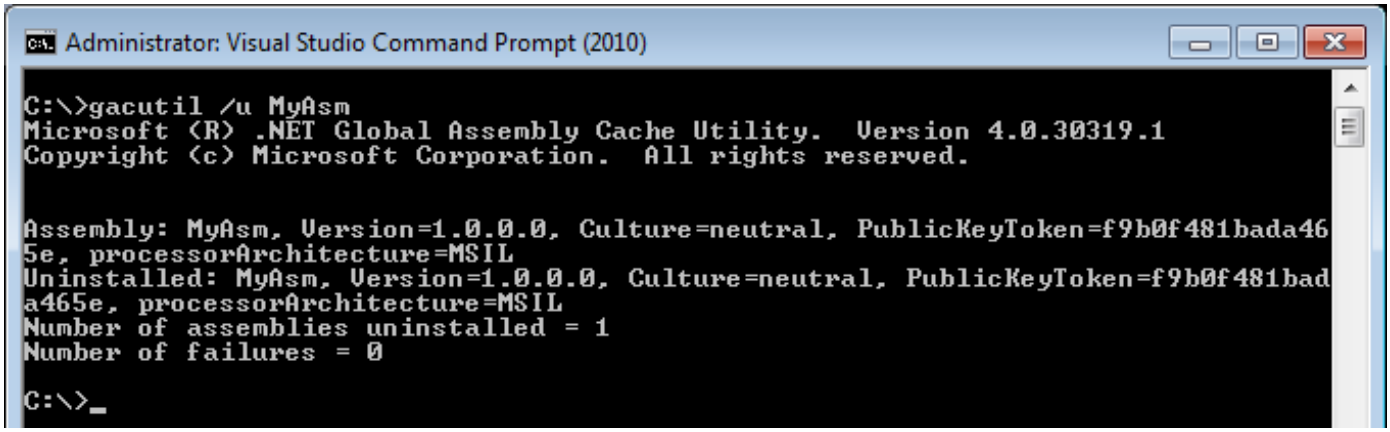
درصورتی‌که اسمبلی موردنظر دارای نام قوی نباشد، خطایی به صورت زیر نمایش داده خواهد شد:



می‌توان نسخه‌های متفاوتی از یک اسمبلی (با نام یکسان) را با استفاده از این ابزار در GAC رجیستر کرد و آن‌ها را در کنار یکدیگر برای استفاده در نرم‌افزارهای گوناگون در اختیار داشت. برای حذف یک اسمبلی از GAC و یا به اصطلاح uninstall کردن آن می‌توان از دستور زیر استفاده کرد:

```
gacutil /u MyAsm
```

نکته : دقت کنید که در این دستور تنها از نام اسمبلی استفاده شده است و نه نام فایل حاوی آن!

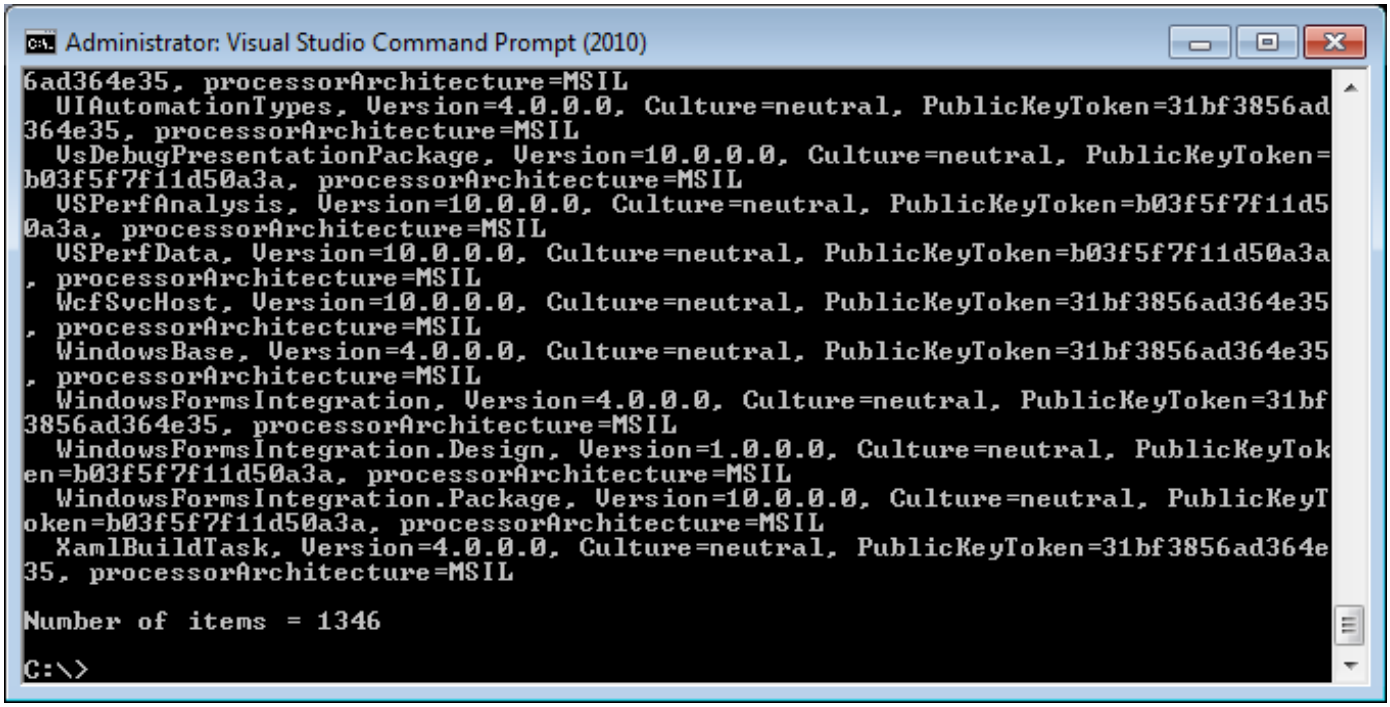


دستور فوق تمام نسخه‌های اسمبلی MyAsm موجود در GAC را حذف خواهد کرد. برای حذف نسخه‌ای خاص باید از دستوری مشابه زیر استفاده کرد:

```
gacutil /u MyAsm,Version=1.3.0.5
```

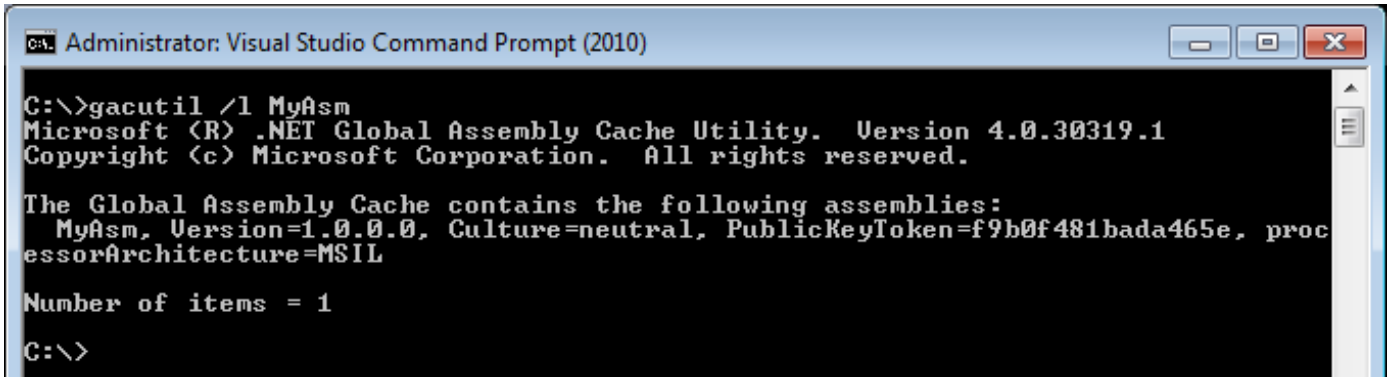
برای مشاهده تمام اسمبلی‌های نصب شده در GAC می‌توان از دستور زیر استفاده کرد:

```
gacutil /l
```



همان‌طور که مشاهده می‌کنید دستور فوق فهرستی بسیار طولانی از تمام اسمبلی‌های نصب‌شده در GAC را به‌همراه لیست اسمبلی‌هایی که در کش ngen به فرم باینری پیش‌کامپایل (Precompiled) شده‌اند، نمایش می‌دهد. برای تعیین اینکه آیا اسمبلی موردنظر در GAC نصب شده است می‌توان از دستور زیر استفاده کرد:

```
gacutil /l MyAsm
```



نکته : دات‌نت از GAC تنها در زمان اجرا استفاده می‌کند. بنابراین کامپایلر #C به‌صورت خودکار درون GAC را برای یافتن ریفرنس‌های یک اسمبلی جستجو نخواهد کرد. در زمان توسعه، کامپایلر #C به یک نسخه لوکال از ریفرنس‌های مذکور نیاز خواهد داشت. برای حل این مشکل می‌توان یک نسخه از این ریفرنس‌ها را به مسیر اسمبلی کپی کرد (در ویژوال استودیو می‌توان از خاصیت Copy Local ریفرنس‌ها استفاده کرد) یا با استفاده از سوییچ /lib کامپایلر، مسیری را که می‌تواند این ریفرنس‌ها را در آن بیابد معرفی کرد (کاری که ویژوال استودیو به‌صورت خودکار انجام می‌دهد).

نکته : نکته‌ای که در پایان باید اشاره کرد این است که تکنولوژی نام قوی برای بحث امنیت کد اسمبلی (مثلا برای جلوگیری از مهندسی معکوس IL و تغییر آن) بوجود **نیامده** است زیرا حذف این نام‌های قوی کار سختی نیست. بلکه هدف اصلی این تکنولوژی جلوگیری از تغییرات مخفی خرابکارانه و محرمانه اسمبلی توزیع شده و توزیع این نسخه‌های دستکاری شده به جای نسخه اصلی است. در زیر ابزارها و روش‌هایی که می‌توانند برای حذف کامل نام قوی یک اسمبلی به‌کار روند آورده شده است. SNRemove v1.00 Removing Strong-Signing from assemblies at file level (byte patching) Remove strong name from assembly while de serialize using regular expressions

البته باید به این نکته اشاره کرد که در صورت حذف نام قوی یک اسمبلی (یا همان حذف امضای دیجیتال درون آن) تمامی اسمبلی‌هایی که قبل از حذف نام قوی به آن ریفرنس داشتند از کار خواهند افتاد. یعنی درواقع تمامی آن اسمبلی‌ها برای ریفرنس دادن به این اسمبلی با نام جدید (نامی که دیگر قوی نیست) باید آپدیت شوند. هم‌چنین درصورتی‌که اسمبلی‌هایی که قبل از حذف نام قوی به اسمبلی موردنظر ما ریفرنس داشتند، خود نام قوی داشته باشند با حذف نام قوی، آنها از کار خواهند افتاد. چون اسمبلی‌های دارای نام قوی تنها می‌توانند از اسمبلی‌های دارای نام قوی ریفرنس داشته باشند. بنابراین برای کارکردن برنامه موردنظر باید نام قوی تمامی اسمبلی‌های درگیر را حذف کرد!

منابع استفاده شده در تهیه این مطلب: Strong Names Explained

Giving a .NET Assembly a Strong Name

How to: Sign an Assembly with a Strong Name

Strong-Named Assemblies

نظرات خوانندگان

نویسنده: مجید
تاریخ: ۱۳۹۱/۰۸/۲۹ ۲۲:۱

فقط می‌تونم بگم عالیه
ممنون

نویسنده: یوسف نژاد
تاریخ: ۱۳۹۲/۰۲/۰۹ ۱۶:۳۹

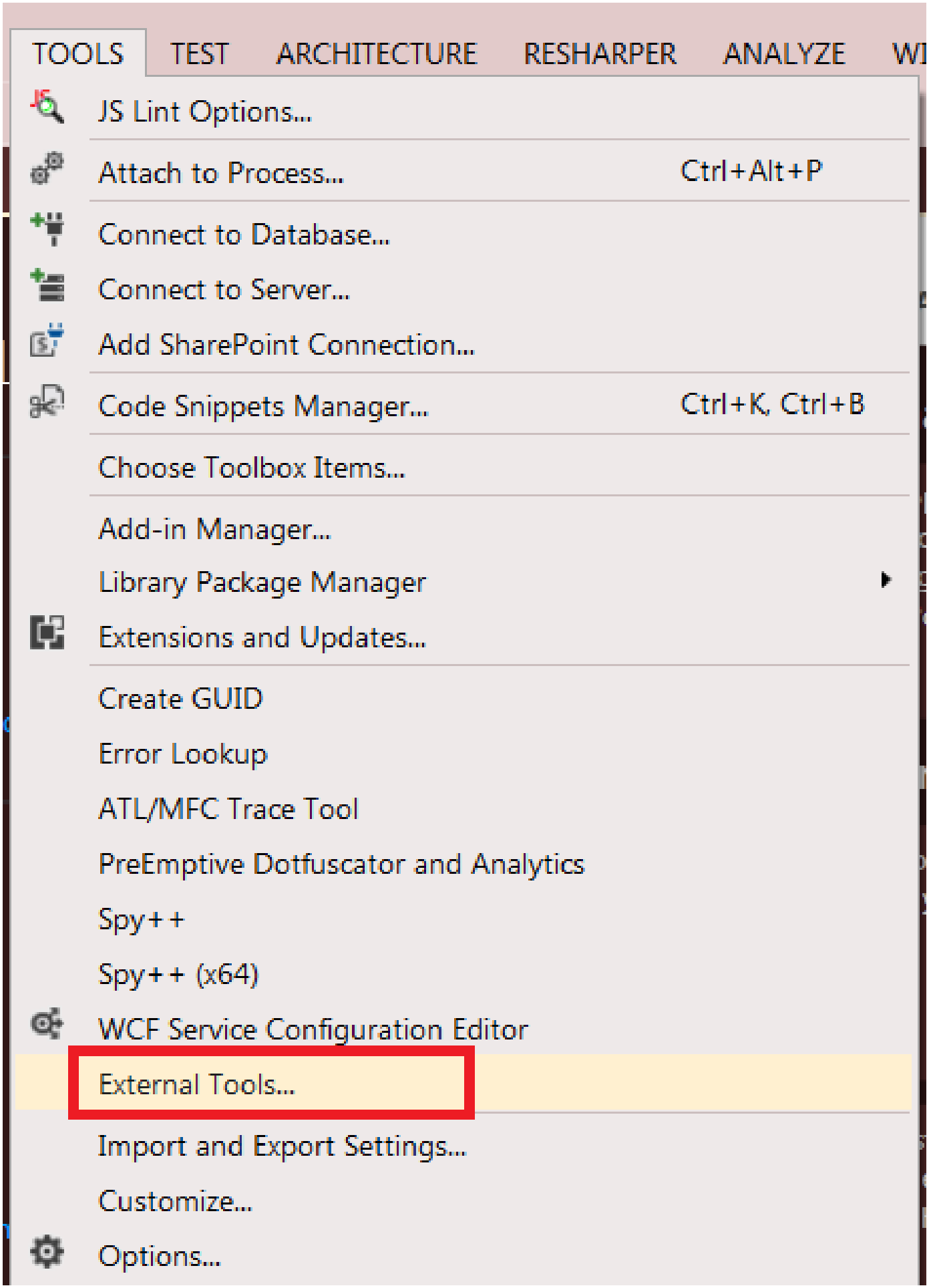
برای مشاهده Public Key Token یک اسمبلی با استفاه از فایل آن نیز میتوان از دستور زیر استفاده کرد:

```
sn  -T  MyAssembly.dll
```

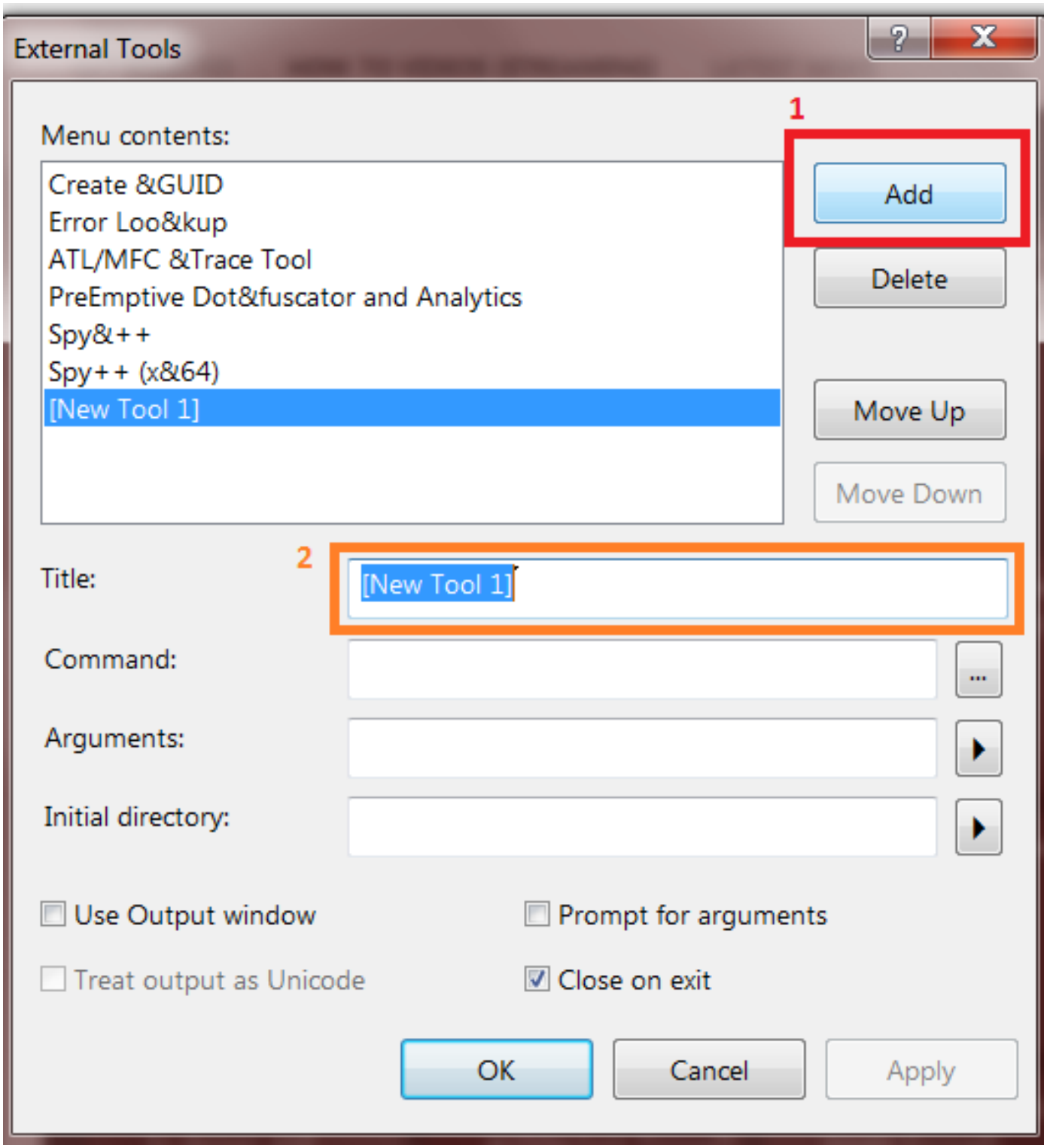
پارامتر T حتما باید با حرف بزرگ وارد شود.

عنوان: **نحوه ایجاد یک External Tools در VS2012 جهت تهیه Public Key Token**
نویسنده: محمد باقر سیف اللهی
تاریخ: ۱۳۹۱/۱۰/۱۴ ۱۱:۳۵
آدرس: www.dotnettips.info
برچسب‌ها: Tools, VisualStudio.NET, .NET, Visual Studio, Strong Name

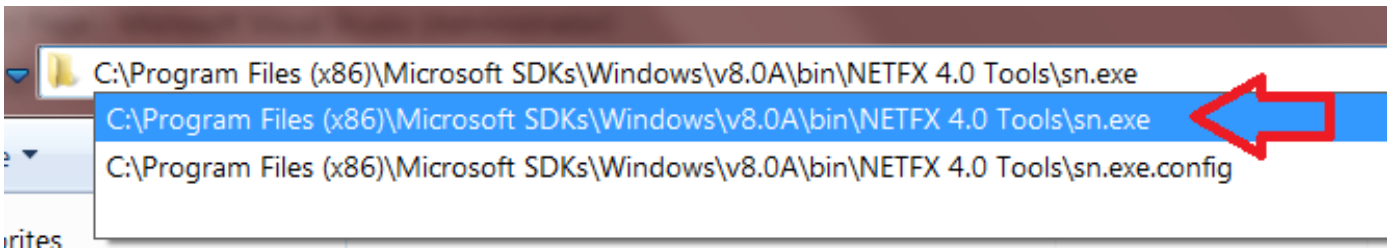
ایجاد Strong Name به اسمبلی برای داشتن یک هویت منحصر به فرد برای آن اسمبلی کمک می‌کند و یکی از پارامترهای آن داشتن Public Key Token برای اسمبلی است (بیشتر). در این پست قصد دارم یه کمک ابزارهای جانبی Visual Studio 2012 که البته در 2010 نیز امکان پذیر است روشی برای تهیه آسان‌تر این Key ارائه کنم .
برای آغاز نرم افزار VS2012 را باز می‌کنیم و به منوی Tools رفته و گزینه External Tools را انتخاب می‌کنیم :


TOOLS
TEST
ARCHITECTURE
RESHARPER
ANALYZE
JS Lint Options...
Attach to Process... Ctrl+Alt+P
Connect to Database...
Connect to Server...
Add SharePoint Connection...
Code Snippets Manager... Ctrl+K, Ctrl+B
Choose Toolbox Items...
Add-in Manager...
Library Package Manager
Extensions and Updates...
Create GUID
Error Lookup
ATL/MFC Trace Tool
PreEmptive Dotfuscator and Analytics
Spy++
Spy++ (x64)
WCF Service Configuration Editor
External Tools...
Import and Export Settings...
Customize...
Options...

در پنجره‌ی پیش رو روی دکمه Add کیلک کنید و نامی برای Tools انتخاب کنید :



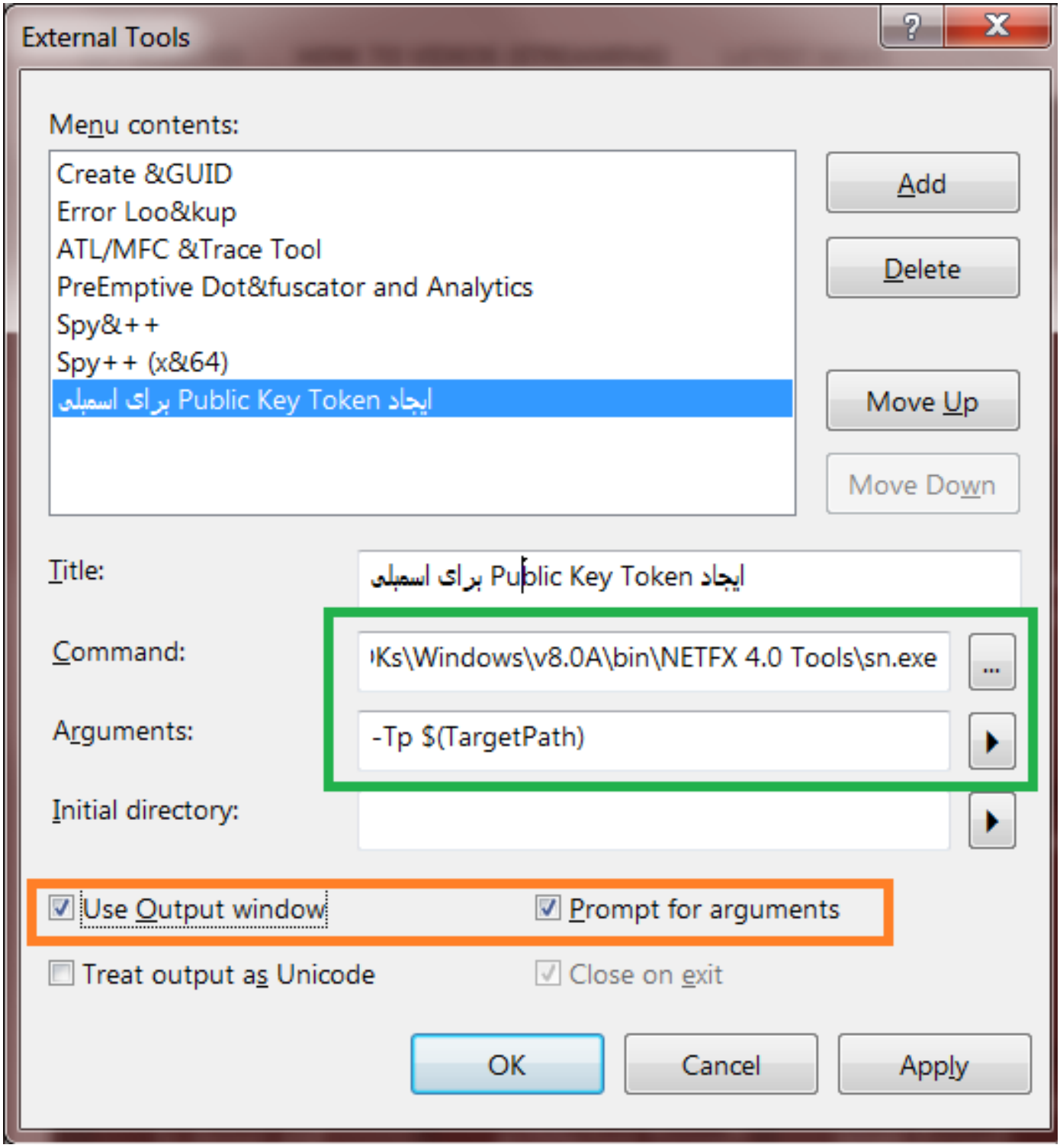
سپس مسیر فایل sn.exe را کپی کرده و در فیلد Command قرار دهید .



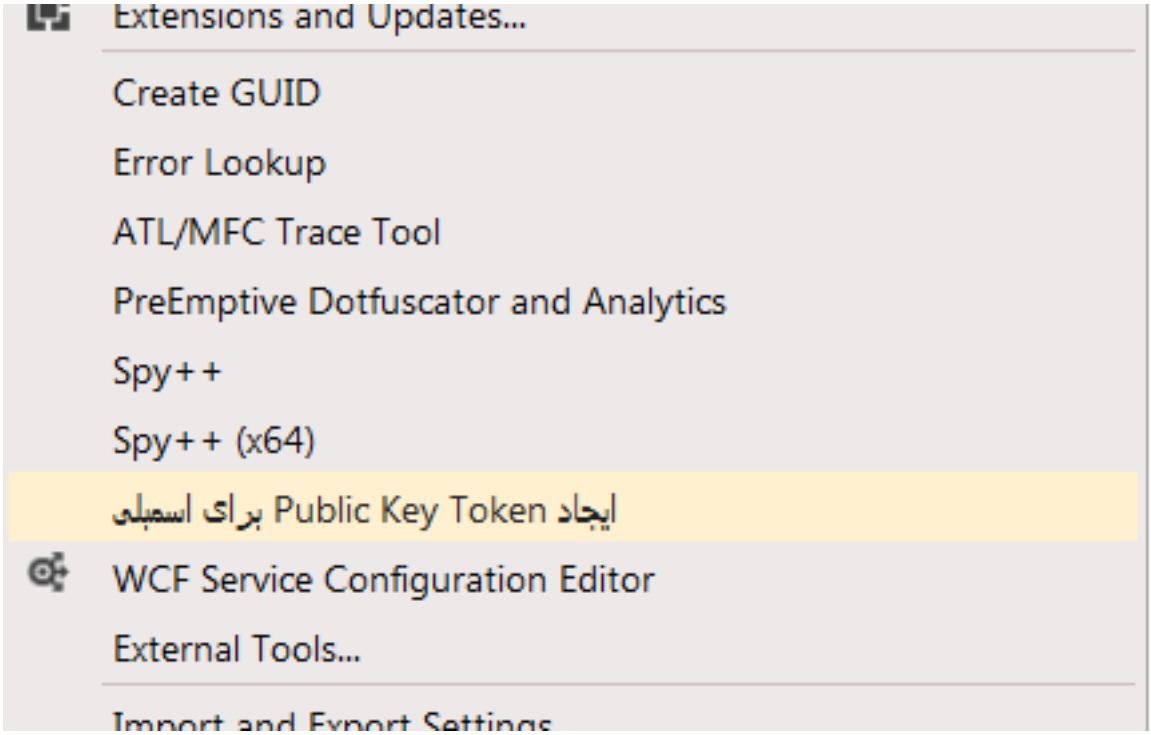
برای پارامتر از عبارت زیراستفاده کرده تا Public Key اسمبلی جاری را به شما بدهد . برای اطلاعات بیشتر در مورد آرگومان‌ها به اینجا مراجعه کنید

```
-Tp $(TargetPath)
```

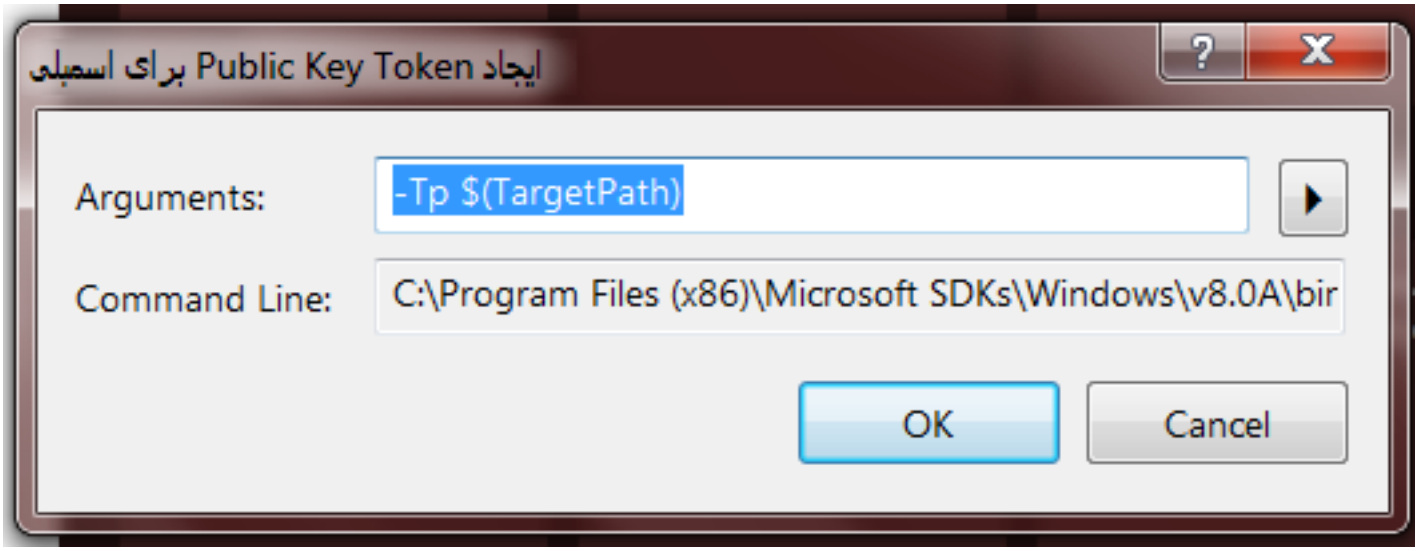
همچنین گزینه‌های Prompt for Arguments را برای دریافت آرگومان دلخواه شما (مثلا مواردی که مایلید برای یک اسمبلی دیگر key استخراج کنید) و Use Output window برای نمایش خروجی را علامت بزنید



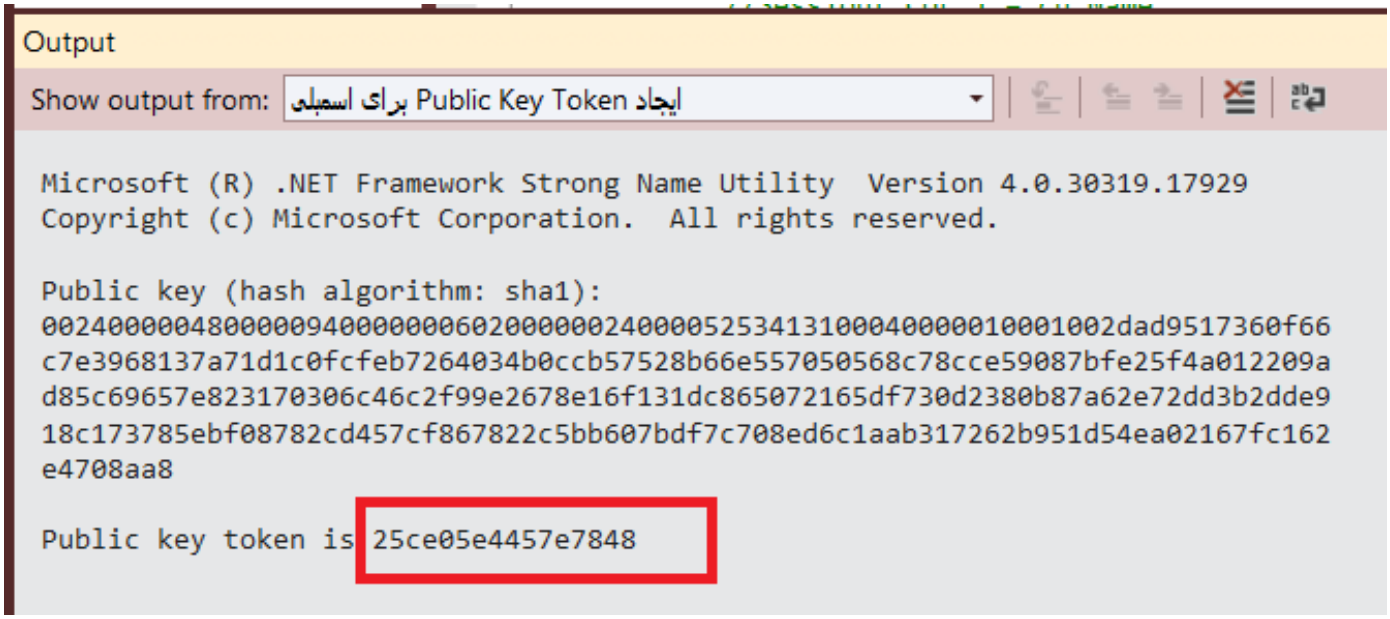
روی OK کلیک کنید و به منوی Tools بازگردید :



حال روی نام پروژه خود در Solution Explorer کلیک کنید و روی Tools ساخته شده کلیک کنید :



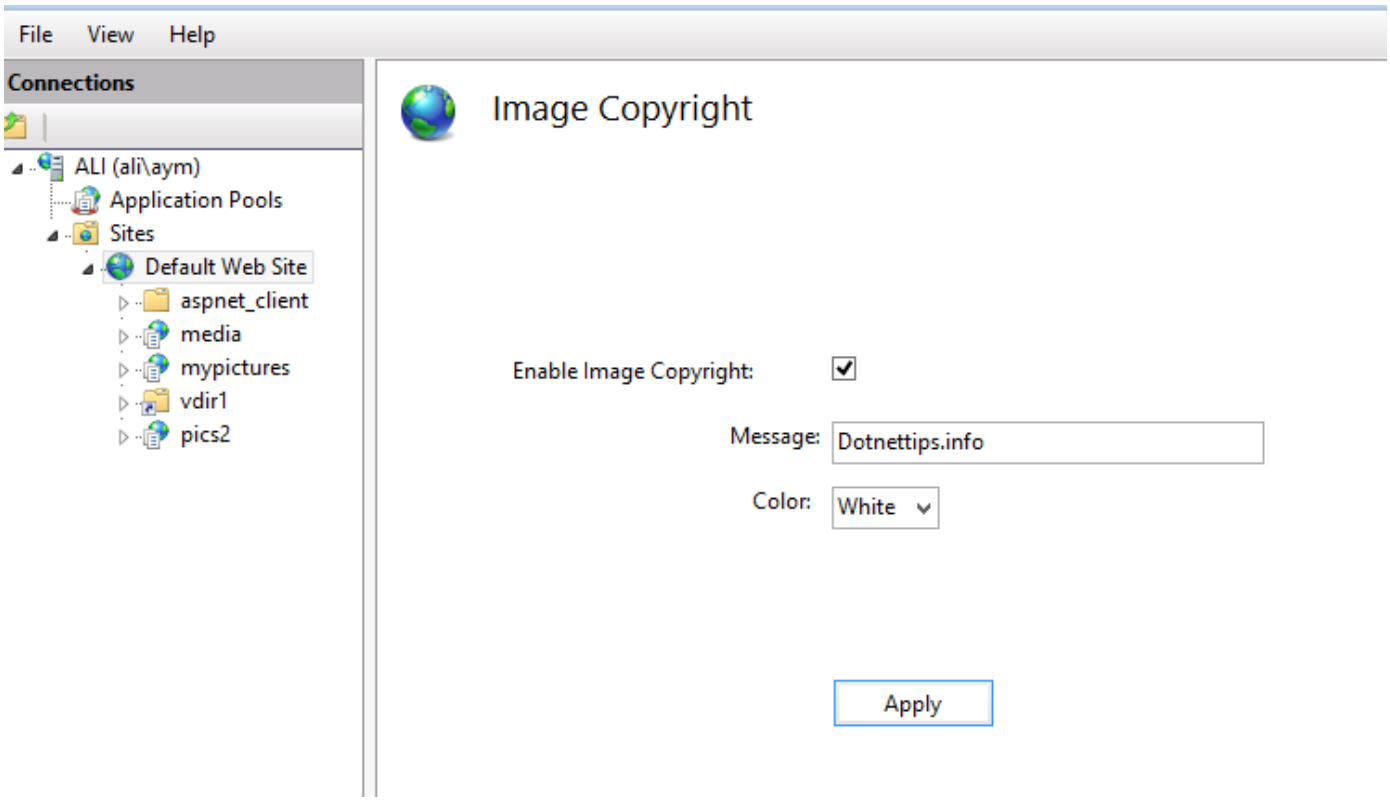
و خروجی:


Output
Show output from: ایجاد Public Key Token برای اسمبلی
Microsoft (R) .NET Framework Strong Name Utility Version 4.0.30319.17929
Copyright (c) Microsoft Corporation. All rights reserved.
Public key (hash algorithm: sha1):
00240000048000009400000006020000002400005253413100040000010001002dad9517360f66
c7e3968137a71d1c0fcfeb7264034b0ccb57528b66e557050568c78cce59087bfe25f4a012209a
d85c69657e823170306c46c2f99e2678e16f131dc865072165df730d2380b87a62e72dd3b2dde9
18c173785ebf08782cd457cf867822c5bb607bdf7c708ed6c1aab317262b951d54ea02167fc162
e4708aa8
Public key token is 25ce05e4457e7848

موفق باشید

عنوان: **آشنایی با ساختار IIS قسمت نهم**
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۳/۱۱/۱۷ ۱:۴۵
آدرس: www.dotnettips.info
گروه‌ها: ASP.Net, IIS, Strong Name, UI, GAC

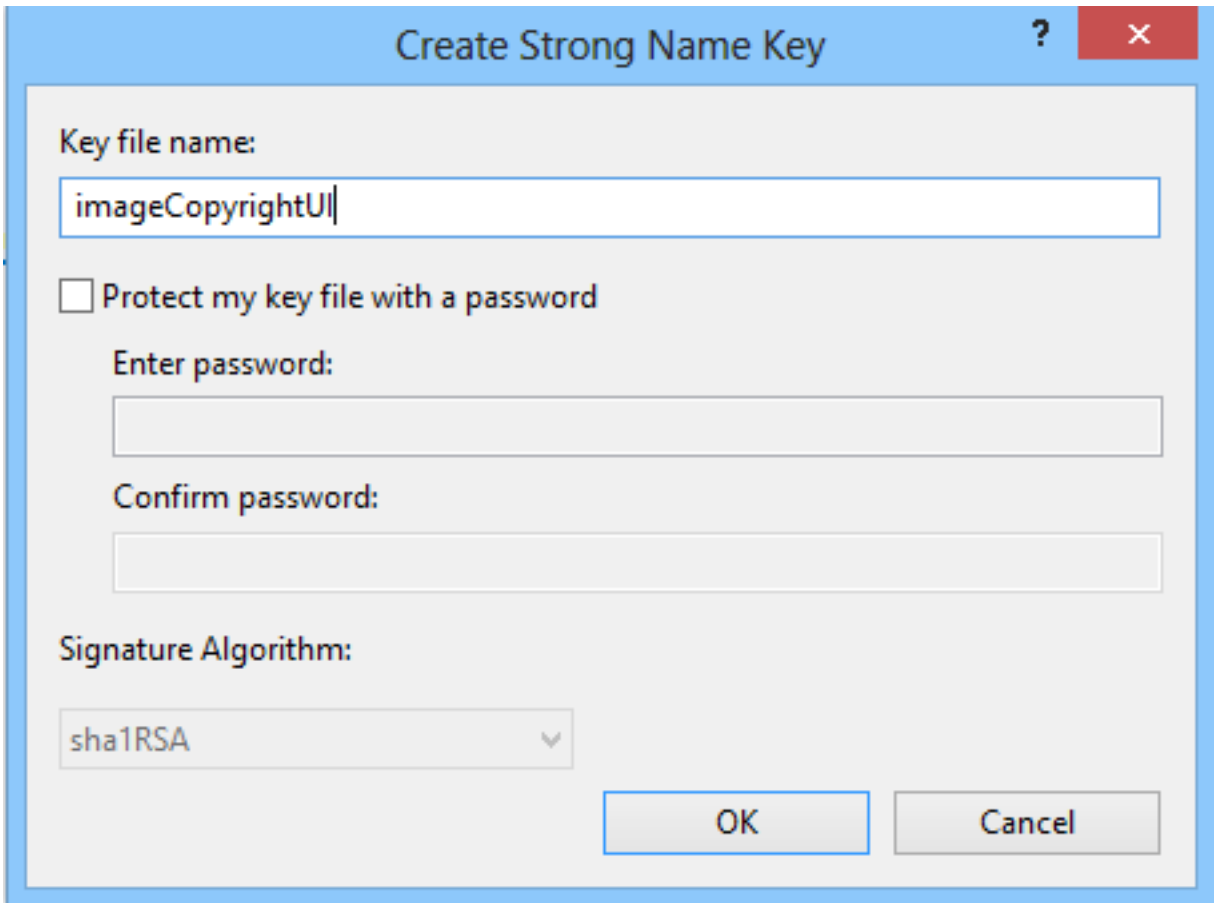
در قسمت قبلی ما یک هندلر ایجاد کردیم و درخواست‌هایی را که برای فایل jpg و به صورت GET ارسال میشد، هندل می‌کردیم و تگی را در گوشه‌ی تصویر درج و آن را در خروجی نمایش میدادیم. در این مقاله قصد داریم که کمی هندلر مورد نظر را توسعه دهیم و برای آن یک UI یا یک رابط کاربری ایجاد نماییم. برای توسعه دادن ماژولها و هندلرها ما یک dll نوشته و باید آن را در GAC که مخفف عبارت Global Assembly Cache ریجستر کنیم.



جهت اینکار یک پروژه از نوع class library ایجاد کنید. فایل class1.cs را که به طور پیش فرض ایجاد می‌شود، حذف کنید و رفرنس‌های Microsoft.Web.Management.dll و Microsoft.Web.Administration.dll را از مسیر زیر اضافه کنید:

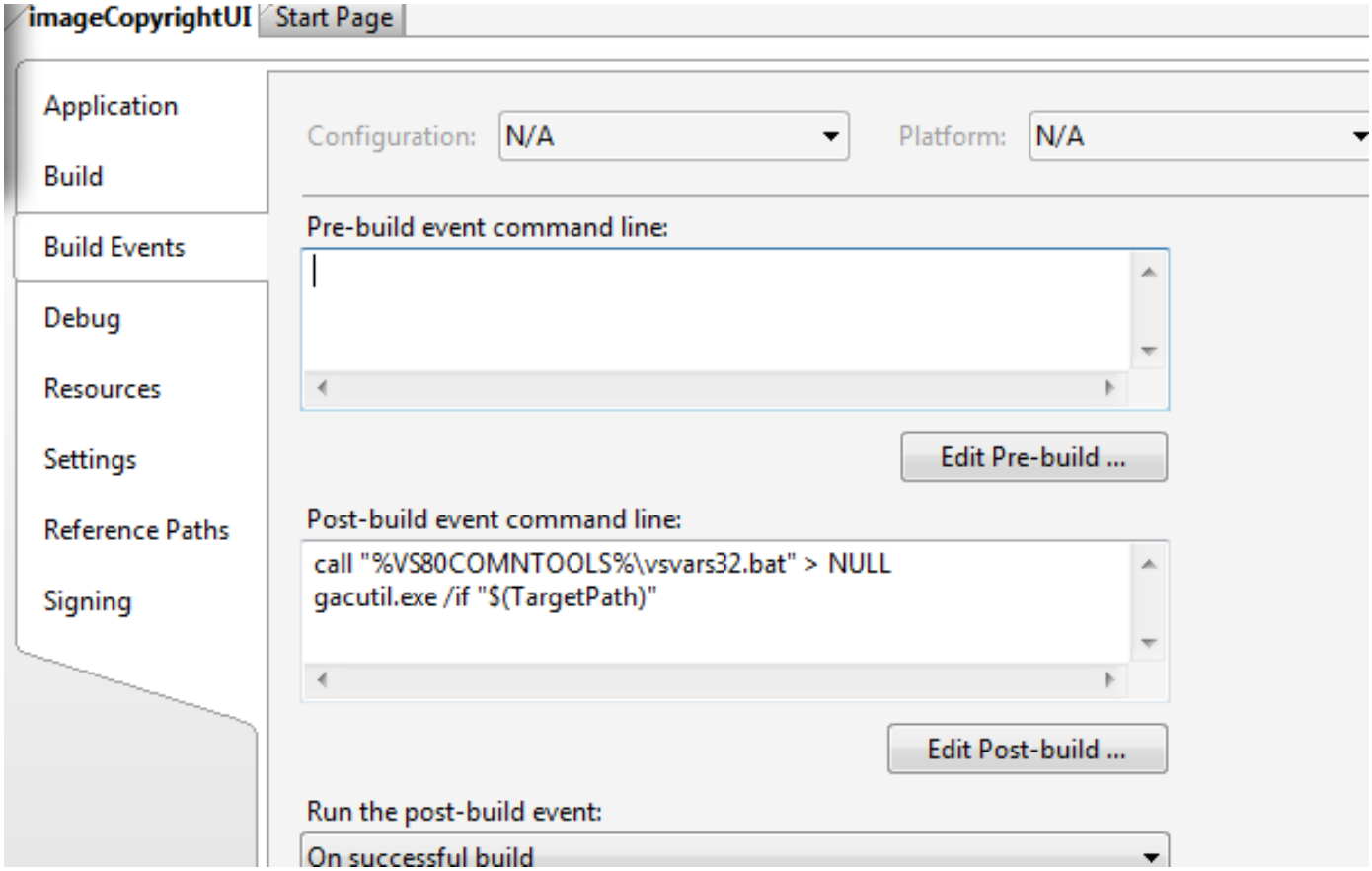
```
\Windows\system32\inetsrv
```

اولین رفرنس شامل کلاس‌هایی است که جهت ساخت ماژول‌ها برای کنسول IIS مورد نیاز است و دومی هم برای خواندن پیکربندی‌های نوشته شده مورد استفاده قرار می‌گیرد.
برای طراحی UI بر پایه winform باید رفرنس‌های System.Windows.Forms.dll و System.Web.dll را از سری اسمبلی‌های دات نت نیز اضافه کنیم و در مرحله‌ی بعدی جهت ایجاد امضاء یا strong name (sn.exe و sn64.exe) به خاطر ثبت در GAC پروژه را انتخاب و وارد Properties پروژه شوید. در تب signing گزینه sign the assembly را تیک زده و در لیست باز شده گزینه new را انتخاب نمایید و نام imageCopyrightUI را به آن نسبت داده و گزینه تعیین کلمه عبور را غیرفعال کنید و تایید و تمام. الان باید یک فایل snk مخفف strong name key ایجاد شده باشد تا بعدا با استفاده از این کلید dll ایجاد شده را در GAC ریجستر کنیم.



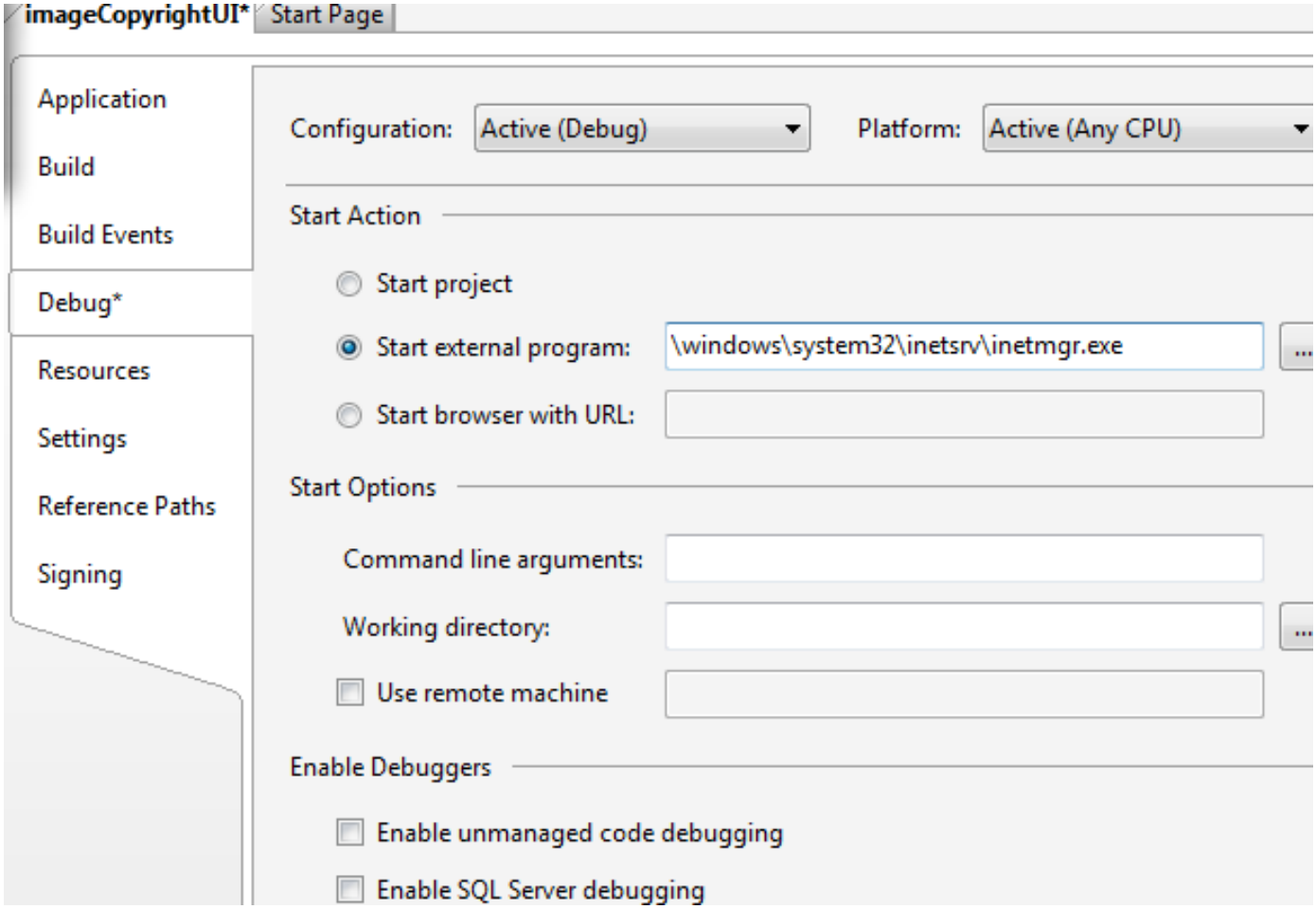
در مرحله بعدی در تب Build Events کد زیر را در بخش Post-build event command line اضافه کنید. این کد باعث می‌شود بعد از هر بار کامپایل پروژه، به طور خودکار در GAC ثبت شود:

```
call "%VS80COMNTOOLS%\vsvars32.bat" > NULL
gacutil.exe /if "$(TargetPath)"
```



نکته:در صورتی که از VS2005 استفاده می‌کنید در تب Debug در قسمت Start External Program مسیر زیر را قرار بدهید.
اینکار برای تست و دیباگینگ پروژه به شما کمک خواهد کرد. این تنظیم شامل نسخه‌های اکسپرس نمی‌شود.

```
\windows\system32\inetsrv\inetmgr.exe
```



بعد از پایان اینکار پروژه را Rebuild کنید. با اینکار dll در GAC ثبت می‌شود. استفاده از سوییچ‌های if به طور همزمان در درستور gacutil به معنی این هست که اگر اولین بار است نصب می‌شود، پس با سوییچ i نصب کن. ولی اگر قبلا نصب شده است نسخه جدید را به هر صورتی هست جایگزین قبلی کن یا همان reinstall کن.

ساخت یک Module Provider

رابط‌های کاربری IIS همانند هسته و کل سیستمش، ماژولار و قابل خصوصی سازی است. رابط کاربری، مجموعه‌ای از ماژول هایی است که میتوان آن‌ها را حذف یا جایگزین کرد. تگ ورودی یا معرفی برای هر UI یک module provider است. خیلی خودمانی، تگ ماژول پروایدر به معرفی یک UI در IIS می‌پردازد. لیستی از module providerها را می‌توان در فایل زیر در تگ بخش <modules> پیدا کرد.

```
%windir%\system32\inetsrv\Administration.config
```

در اولین گام یک کلاس را به اسم imageCopyrightUIModuleProvider.cs ایجاد کرده و سپس آن‌را به کد زیر، تغییر می‌دهیم. کد زیر با استفاده از ModuleDefinition یک نام به تگ Module Provider داده و کلاس imageCopyrightUI را که بعدا تعریف می‌کنیم، به عنوان مدخل entry رابط کاربری معرفی کرده:

```
using System;
using System.Security;
using Microsoft.Web.Management.Server;

namespace IIS7Demos
{
    class imageCopyrightUIProvider : ModuleProvider
    {
        public override Type ServiceType
```

```
        {
            get { return null; }
        }

        public override ModuleDefinition GetModuleDefinition(IManagementContext context)
        {
            return new ModuleDefinition(Name, typeof(imageCopyrightUI).AssemblyQualifiedName);
        }

        public override bool SupportsScope(ManagementScope scope)
        {
            return true;
        }
    }
}
```

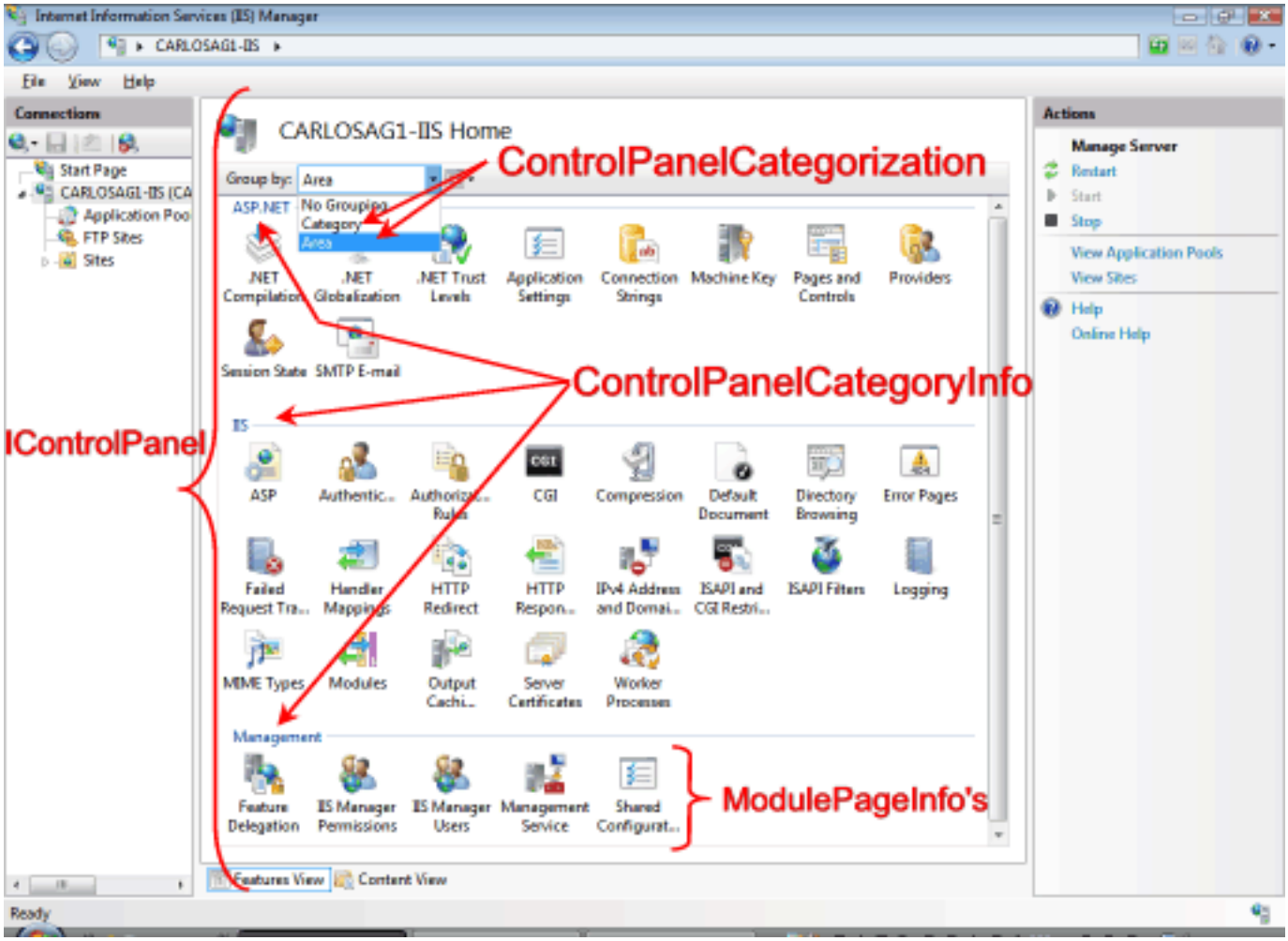
با ارث بری از کلاس module provider، سه متد بازنویسی می‌شوند که یکی از آن ها SupportsScope هست که میدان عمل پروایدر را مشخص می‌کند، مانند اینکه این پروایدر در چه میدانی باید کار کند که می‌تواند سه گزینه‌ی server,site,application باشد. در کد زیر مثلا میدان عمل application انتخاب شده است ولی در کد بالا با برگشت مستقیم true، همه‌ی میدان را جهت پشتیبانی از این پروایدر اعلام کردیم.

```
public override bool SupportsScope(ManagementScope scope)
        {
            return (scope == ManagementScope.Application) ;
        }
```

حالا که پروایدر (معرف رابط کاربری به IIS) تامین شده، نیاز است قلب کار یعنی ماژول معرفی گردد. اصلی‌ترین متدی که باید از اینترفیس ماژول پیاده سازی شود متد initialize است. این متد جایی است که تمام عملیات در آن رخ می‌دهد. در کلاس زیر imageCopyrightUI ما به معرفی مدخل entry رابط کاربری می‌پردازیم. در سازنده‌های این متد، پارامترهای نام، صفحه رابط کاربری وتوضیحی در مورد آن است. تصویر کوچک و بزرگ جهت آیکن سازی (در صورت عدم تعریف آیکن، چرخ دنده نمایش داده می‌شود) و توصیف‌های بلندتر را نیز شامل می‌شود.

```
internal class imageCopyrightUI : Module
    {
        protected override void Initialize(IServiceProvider serviceProvider, ModuleInfo moduleInfo)
        {
            base.Initialize(serviceProvider, moduleInfo);
            IControlPanel controlPanel = (IControlPanel)GetService(typeof(IControlPanel));
            ModulePageInfo modulePageInfo = new ModulePageInfo(this, typeof(imageCopyrightUIPage),
"Image Copyright", "Image Copyright",Resource1.Visual_Studio_2012,Resource1.Visual_Studio_2012);
            controlPanel.RegisterPage(modulePageInfo);
        }
    }
```

شیء ControlPanel مکانی است که قرار است آیکن ماژول نمایش داده شود. شکل زیر به خوبی نام همه قسمت‌ها را بر اساس نام کلاس و اینترفیس آن‌ها دسته بندی کرده است:



پس با تعریف این کلاس جدید ما روی صفحه‌ی کنترل پنل IIS، یک آیکن ساخته و صفحه‌ی رابط کاربری را به نام imageCopyrightUIPage، در آن ریجستر می‌کنیم. این کلاس را پایینتر شرح داده‌ایم. ولی قبل از آن اجازه بدهید تا انواع کلاس هایی را که برای ساخت صفحه کاربرد دارند، بررسی نماییم. در این مثال ما با استفاده از پایه‌ای‌ترین کلاس، ساده‌ترین نوع صفحه ممکن را خواهیم ساخت. 4 کلاس برای ساخت یک صفحه وجود دارند که بسته به سناریوی کاری، شما یکی را انتخاب می‌کنید.

ModulePage	شامل اساسی‌ترین متدها و سورس‌ها شده و هیچگونه رابط کاری ویژه‌ای را در اختیار شما قرار نمی‌دهد. تنها یک صفحه‌ی خام به شما می‌دهد که می‌توانید از آن استفاده کرده یا حتی با ارث بری از آن، کلاس‌های جدیدتری را برای ساخت صفحات مختلف و ویژه‌تر بسازید. در حال حاضر که هیچ کدام از ویژگی‌های IIS فعلی از این کلاس برای ساخت رابط کاربری استفاده نکرده‌اند.
ModuleDialogPage	یک صفحه شبیه به دیالوگ را ایجاد می‌کند و شامل دکمه‌های Apply و Cancel میشود به همراه یک سری متدهای اضافی‌تر که اجازه‌ی override کردن آنها را دارید. همچنین یک سری از کارهایی چون refresh و از این دست عملیات خودکار را نیز انجام میدهد. از نمونه رابطهایی که از این صفحات استفاده می‌کنند میتوان machine key و management service را اسم برد.

ModulePage	شامل اساسی‌ترین متدها و سورس‌ها شده و هیچگونه رابط کاری ویژه‌ای را در اختیار شما قرار نمی‌دهد. تنها یک صفحه‌ی خام به شما می‌دهد که می‌توانید از آن استفاده کرده یا حتی با ارث بری از آن، کلاس‌های جدیدتری را برای ساخت صفحات مختلف و ویژه‌تر بسازید. در حال حاضر که هیچ کدام از ویژگی‌های IIS فعلی از این کلاس برای ساخت رابط کاربری استفاده نکرده‌اند.
ModulePropertiesPage	این صفحه یک رابط کاربری را شبیه پنجره property که در ویژوال استادیو وجود دارد، در دسترس شما قرار میدهد. تمام عناصر آن در یک حالت گرید grid لیست می‌شوند. از نمونه‌های موجود میتوان به CGI,ASP.Net Compilation اشاره کرد.
ModuleListPage	این کلاس برای مواقعی کاربرد دارد که شما قرار است لیستی از آیتم‌ها را نشان دهید. در این صفحه شما یک ListView دارید که میتوانید عملیات جست و جو، گروه بندی و نحوه‌ی نمایش لیست را روی آن اعمال کنید.

در این مثال ما از اولین کلاس نامبرده که پایه‌ی همه کلاس هاست استفاده می‌کنیم. کد زیر را در کلاسی به اسم imageCopyrightUIPage می‌نویسیم:

```
public sealed class imageCopyrightUIPage : ModulePage
    {
        public string message;
        public bool featureenabled;
        public string color;

        ComboBox _colCombo = new ComboBox();
        TextBox _msgTB = new TextBox();
        CheckBox _enabledCB = new CheckBox();

        public imageCopyrightUIPage()
        {
            this.Initialize();
        }


        void Initialize()
        {
            Label crlabel = new Label();
            crlabel.Left = 50;
            crlabel.Top = 100;
            crlabel.AutoSize = true;
            crlabel.Text = "Enable Image Copyright:";
            _enabledCB.Text = "";
            _enabledCB.Left = 200;
            _enabledCB.Top = 100;
            _enabledCB.AutoSize = true;

            Label msglabel = new Label();
            msglabel.Left = 150;
            msglabel.Top = 130;
            msglabel.AutoSize = true;
            msglabel.Text = "Message:";
            _msgTB.Left = 200;
            _msgTB.Top = 130;
            _msgTB.Width = 200;
            _msgTB.Height = 50;

            Label collabel = new Label();
            collabel.Left = 160;
            collabel.Top = 160;
            collabel.AutoSize = true;
            collabel.Text = "Color:";
            _colCombo.Left = 200;
```

```
            _colCombo.Top = 160;
            _colCombo.Width = 50;
            _colCombo.Height = 90;
            _colCombo.Items.Add((object)"Yellow");
            _colCombo.Items.Add((object)"Blue");
            _colCombo.Items.Add((object)"Red");
            _colCombo.Items.Add((object)"White");

            Button apply = new Button();
            apply.Text = "Apply";
            apply.Click += new EventHandler(this.applyClick);
            apply.Left = 200;
            apply.AutoSize = true;
            apply.Top = 250;

            Controls.Add(crlabel);
            Controls.Add(_enabledCB);
            Controls.Add(collabel);
            Controls.Add(_colCombo);
            Controls.Add(msglabel);
            Controls.Add(_msgTB);
            Controls.Add(apply);
        }

        public void ReadConfig()
        {
            try
            {
                ServerManager mgr;
                ConfigurationSection section;
                mgr = new ServerManager();
                Configuration config =
                mgr.GetWebConfiguration(
                       Connection.ConfigurationPath.SiteName,
                       Connection.ConfigurationPath.ApplicationPath +
                       Connection.ConfigurationPath.FolderPath);

                section = config.GetSection("system.webServer/imageCopyright");
                color = (string)section.GetAttribute("color").Value;
                message = (string)section.GetAttribute("message").Value;
                featureenabled = (bool)section.GetAttribute("enabled").Value;

            }

            catch
            { }

        }

        void UpdateUI()
        {
            _enabledCB.Checked = featureenabled;
            int n = _colCombo.FindString(color, 0);
            _colCombo.SelectedIndex = n;
            _msgTB.Text = message;
        }


        protected override void OnActivated(bool initialActivation)
        {
            base.OnActivated(initialActivation);
            if (initialActivation)
            {
                ReadConfig();
                UpdateUI();
            }
        }



        private void applyClick(Object sender, EventArgs e)
        {
            try
            {
                UpdateVariables();
                ServerManager mgr;
                ConfigurationSection section;
                mgr = new ServerManager();
                Configuration config =
                mgr.GetWebConfiguration
                (
```

```
                        Connection.ConfigurationPath.SiteName,
                        Connection.ConfigurationPath.ApplicationPath +
                        Connection.ConfigurationPath.FolderPath
                );

                section = config.GetSection("system.webServer/imageCopyright");
                section.GetAttribute("color").Value = (object)color;
                section.GetAttribute("message").Value = (object)message;
                section.GetAttribute("enabled").Value = (object)featureenabled;

                mgr.CommitChanges();

            }

            catch
            { }

        }

        public void UpdateVariables()
        {
            featureenabled = _enabledCB.Checked;
            color = _colCombo.Text;
            message = _msgTB.Text;
        }
    }
```

اولین چیزی که در کلاس بالا صدا زده می‌شود، سازنده‌ی کلاس هست که ما در آن یک تابع تعریف کردیم به اسم **initialize** که به آماده سازی اینترفیس یا رابط کاربری می‌پردازد و کنترل‌ها را روی صفحه می‌چیند. این سه کنترل، یکی *Combox* برای تعیین رنگ، یک *Checkbox* برای فعال بودن ماژول و دیگری هم یک *textbox* جهت نوشتن متن است. مابقی هم که سه *label* برای نامگذاری اشیاست. بعد از اینکه کنترل‌ها روی صفحه درج شدند، لازم است که تنظیمات پیش فرض یا قبلی روی کنترل‌ها نمایش یابند که اینکار را به وسیله تابع **readConfig** انجام می‌دهیم و تنظیمات خوانده شده را در متغیرهای عمومی قرار داده و با استفاده از تابع **UpdateUI** این اطلاعات را روی کنترل‌ها ست می‌کنیم و به این ترتیب *UI* به روز می‌شود. این دو تابع را به ترتیب پشت سر هم در یک متد به اسم **OnActivated** که *override* کرده‌ایم صدا میزنیم. در واقع این متد یک جورایی همانند رویداد *Load* می‌باشد؛ اگر *true* برگرداند اولین فعال سازی رابط کاربری بعد از باز شدن IIS است و در غیر این صورت *false* بر میگرداند.

در صورتی که کاربر مقادیر را تغییر دهد و روی گزینه *apply* کلیک کند تابع applyClick اجرا شده و ابتدا به تابع UpdateVariables ارجاع داده میشود که در آن مقادیر خوانده شده و در متغیرهای *Global* قرار می‌گیرند و سپس با استفاده از دو شیء از نوع *serverManger* و *ConfigSection* جایگذاری یا ذخیره می‌شوند.
استفاده از دو کلاس Servermanager و Configsection در دو قسمت خواندن و نوشتن مقادیر به کار رفته‌اند. کلاس servermanager به ما اجازه دسترسی به تنظیمات IIS و قابلیت‌های آن را میدهد. در تابع ReadConfig مسیر وب سایتی را که در لیست IIS انتخاب شده است، دریافت کرده و به وب کانفیگ آن وب سایت رجوع نموده و تگ imageCopyright آن را که در تگ system.webserver قرار گرفته است، میخواند (در صورتی که این تگ در آن وب کانفیگ موجود نباشد، خواندن و سپس ذخیره مجدد آن روی تگ داخل فایل applicationHost.config اتفاق میفتد که نتیجتا برای همه‌ی وب سایت هایی که این تگ را ندارند یا مقدارهای پیش فرض آن را تغییر نداده‌اند رخ میدهد) عملیات نوشتن هم مشابه خواندن است. تنها باید خط زیر را در آخر برای اعمال تغییرات نوشت؛ مثل EF با گزینه Context.SaveChanges:

```
mgr.CommitChanges();
```

وقت آن است که رابط کاربری را به IIS اضافه کنیم: پروژه را Rebuild کنید. بعد از آن با خطوطی که قبلا در Post-Build Command نوشتیم باید dll ما در GAC ریجستر شود. برای همین آدرس زیر را در cmd تایپ کنید:

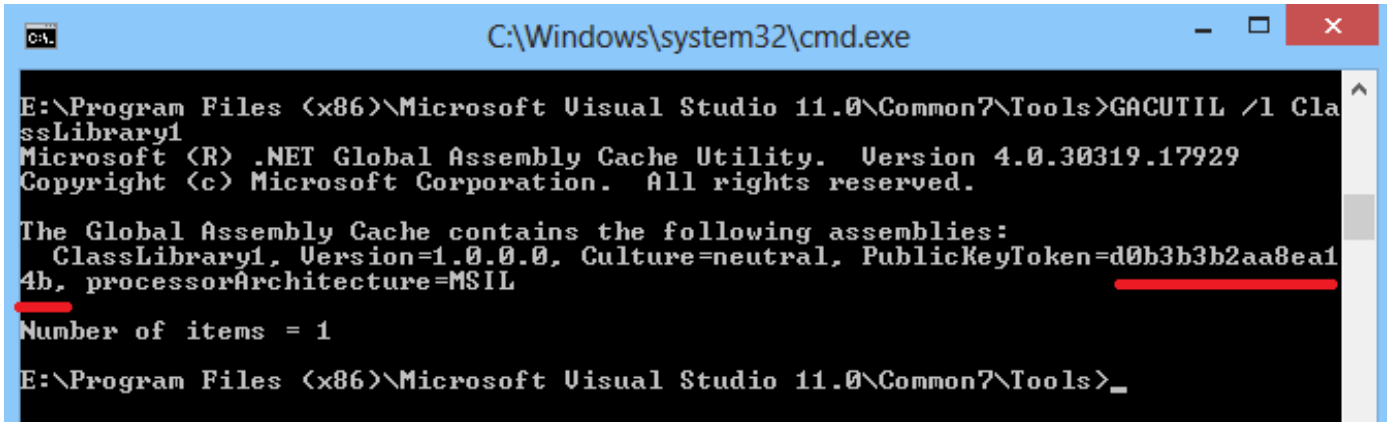
```
%vs110comntools%\vsvars32.bat
```

عبارت اول که مسیر ویژوال استودیوی شماست و عدد 110 یعنی نسخه‌ی 11. هر نسخه‌ای را که استفاده میکنید، یک صفر جلویش بگذارید و جایگزین عدد بالا کنید. مثلا نسخه 8 می‌شود 80 و فایل بچ بالا هم دستورات visual studio را برای شما آزاد می‌کند.

سپس دستور زیر را وارد کنید:

```
GACUTIL /l ClassLibrary1
```

کلمه classLibrary1 نام پروژه‌ی ما بود که در GAC ریجستر شده است. با سوییچ l تمامی اطلاعات اسمبلی‌هایی که در GAC ریجستر شده‌اند، نمایش می‌یابند. ولی اگر اسم آن اسمبلی را جلویش بنویسید، فقط اطلاعات آن اسمبلی نمایش میابد. با اجرای خط فوق میتوانیم کلید عمومی public key اسمبلی خود را بدانیم که در شکل زیر مشخص شده است:



پس اگر کلید را دریافت کرده‌اید، خط زیر را به فایل administration.config در تگ <ModuleProviders> اضافه کنید:

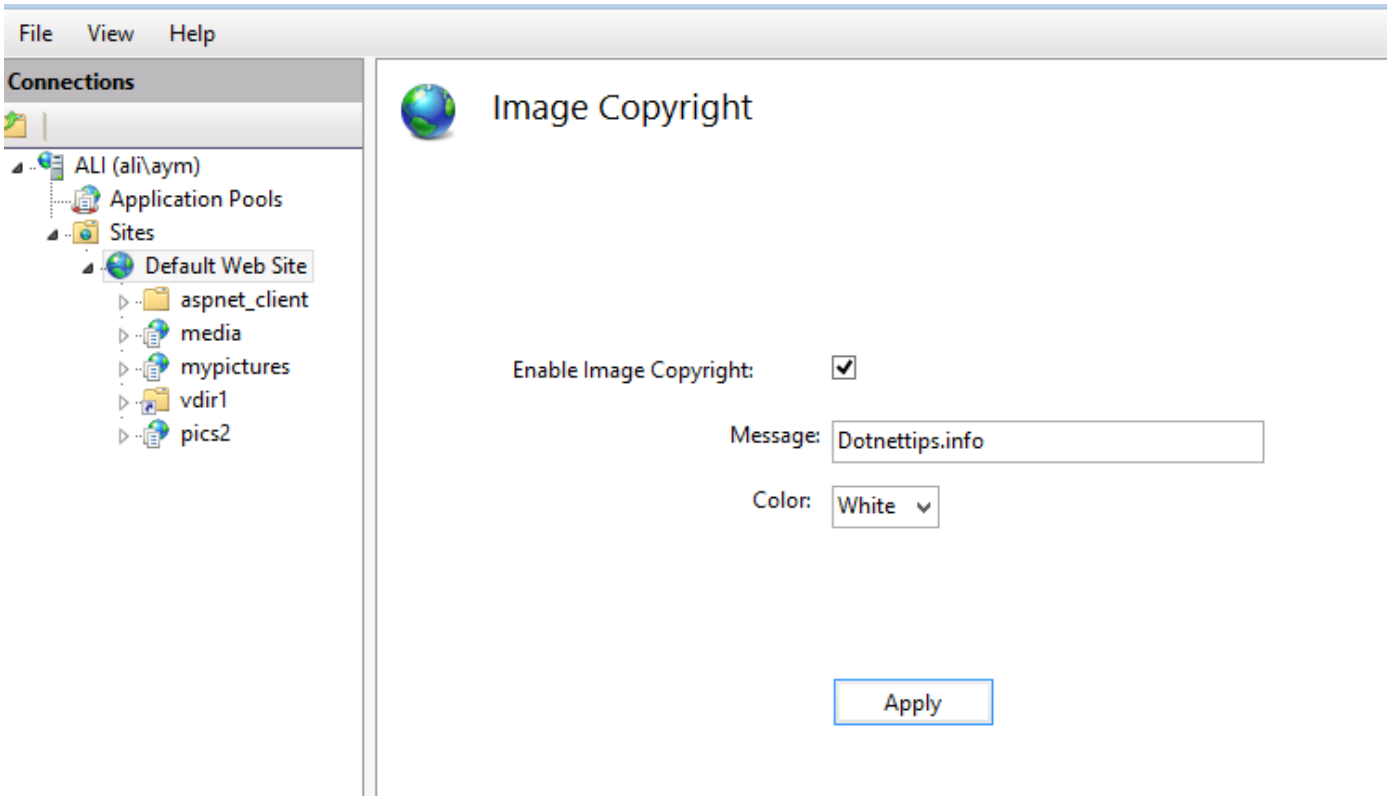
```
<add name="imageCopyrightUI" type="ClassLibrary1.imageCopyrightUIProvider,   ClassLibrary1,
Version=1.0.0.0, Culture=neutral, PublicKeyToken=d0b3b3b2aa8ea14b"/>
```

عبارت ClassLibrary1.imageCopyrightUIProvider به کلاس imageCopyrightUIProvider اشاره می‌کند که در این کلاس UI معرفی می‌شود. مابقی عبارت هم کاملا مشخص است و در لینک‌های بالا در مورد Strong name توضیح داده شده اند.

فایل administration.config در مسیر زیر قرار دارد:

```
%windir%\system32\inetsrv\config\administration.config
```

حالا تنها کاری که نیاز است، باز کردن IIS است. به بخش وب سایت‌ها رفته و اپلیکیشنی که قبلا با نام mypictures را ایجاد کرده بودیم، انتخاب کنید. در سمت راست، آخر لیست، بخش others باید ماژول ما دیده شود. بازش کنید و تنظمیات آن را تغییر دهید و حالا یک تصویر را از اپلیکیشن mypictures، روی مرورگر درخواست کنید تا تغییرات را روی تگ مشاهده کنید:



حالا دیگر باید ماژول نویسی برای IIS را فراگرفته باشیم. این ماژول‌ها می‌توانند از یک مورد ساده تا یک کلاس مهم و امنیتی باشند که روی سرور شما برای همه یا بعضی از وب سایت‌ها در حال اجرا هستند و در صورت لزوم و اجازه شما، برنامه نویس‌ها میتوانند مثل همه‌ی تگ‌های موجود در وب کانفیگ سایتی را که مینویسند، تگ ماژول شما و تنظیمات آن را با استفاده از attribute یا خصوصیت‌های تعریف شده، بر اساس سلایق و نیازهایشان تغییر دهند و روی سرور شما آپلود کنند. الان شما یک سرور خصوصی سازی شده دارید.

از آنجا که این مقاله طولانی شده است، باقی موارد ویرایشی روی این UI را در مقاله بعدی بررسی خواهیم کرد.