

عنوان: Entity Framework و آینده
نویسنده: علیرضا صالحی
تاریخ: ۱۳۹۱/۰۵/۰۲
آدرس: www.dotnettips.info
برچسب‌ها: Entity framework, DbContext, .NET

همان طور که می‌دانید نسخه 5 (نهایی) از EF به همراه Visual Studio 2012 منتشر خواهد شد ([...](#)) و قابلیت‌های کلیدی افزوده شده به آن عبارتند از:

پشتیبانی از Enum در هر سه حالت (Database First, Code First, Model First)
پشتیبانی از Tabel-valued Function در حالت Database First
پشتیبانی از داده‌های جغرافیایی در هر سه حالت (Database First, Code First, Model First)
افزایش کارایی قابل توجه در LINQ To Entites و Entity SQL ([...](#))

قابلیت داشتن چند دیاگرام برای یک مدل
قابلیت ایمپورت دسته ای Stored Procedure ها
شاید این بهبودها کم به نظر برسند ولی اتفاق مهم دیگری که رخ داده متن باز شدن کامل EF است (قبلا در 4.1 متن باز شده بود) که در این آدرس نه تنها می‌توانید ([...](#)) به سورس کدها دسترسی پیدا کنید بلکه می‌توانید در تکمیل پروژه و رفع نواقص آن نیز شرکت کنید. ([...](#))
بنابراین روند توسعه EF از این پس کاملاً قابل پیگیری (و شاید قابل تغییر) است. ([...](#))

قابلیت‌های جدیدی که برای EF نسخه 6 در نظر گرفته شده اند عبارتند از:

بهره گیری از قابلیت async در دات نت 4.5 و معرفی Async Query & Update

```
public async Task<Store> FindClosestStore(DbGeography location)
{
    using (var context = new StoreContext())
    {
        return await (from s in context.Stores
                      orderby s.Location.Distance(location)
                      select s).FirstAsync();
    }
}
```

پشتیبانی از نگاشت Stored Procedure و Function در حالت Code First
پشتیبانی از Code First conventions سفارشی (یک کاربرد آن برای جلوگیری از حجم زیاد کد نویسی در هنگام تولید مدل OnModelCreating) ([...](#))

نظرات خوانندگان

نویسنده: رضا.ب

تاریخ: ۱۳۹۱/۰۵/۰۲ ۲:۲۶

اگه میشد مطالب مرجع سایت که غالبا مهندس نصیری نوشتند رو همزمان با این تغییرات نهایی تغییر داد خیلی کار جالب توجهی میشد.

مثلا یا به حالت ویکی که بشه نظارت کرد رو ورژن‌های مختلفی که به‌روزرسانی شدند. یا در همچین‌جور پست‌هایی با اشاره به قابلیت جدید و یا منسوخ شده‌ی مطالب مرجع.
یه سوال؛ آیا انتظار درستی که مرز حاضر بین ORM و رابط‌های اشیاء دیتابیس‌های NoSQL رو حذف کرد و به یه اتحاد واحد رسید. که مثلا از EF به عنوان یه روش کلی برای ارتباط با "منبع داده‌ای" یاد شود؟ چراکه همکنون ORM نقشی در NoSQL‌ها ندارند.

نویسنده: سیروان عفیفی

تاریخ: ۱۳۹۱/۰۵/۰۲ ۹:۴۳

دیگه باید شاهد رشد سریع EF باشیم.

نویسنده: علیرضا صالحی

تاریخ: ۱۳۹۱/۰۵/۰۲ ۹:۴۵

در مورد داشتن یک ORM که هم با NoSQL‌ها کار کند و هم با RDBMS‌ها چند نکته وجود دارد، اول این که خود NoSQL‌ها خیلی با هم سازگاری ندارند، روش ذخیره سازی، مدل ذخیره سازی و ...
دوم اینکه به طور کلی طرز تفکر و مورد استفاده و شکل Query‌هایی که همه ما در RDBMS‌ها به آن عادت داریم در NoSQL جاری نیست. بنابراین داشتن ORM ی که هر دو را پوشش دهد شاید منطقی به نظر نرسد.
در اینجا بحث خوبی در این زمینه انجام شده

نویسنده: مهدی پایروند

تاریخ: ۱۳۹۱/۰۵/۰۲ ۱۲:۵۲

هرجا صحبت از تفاوت و مزایای این دو ORM یعنی NH و EF پیش میاد، اولین تفاوت پشتیبانی NHibernate از کش لایه دوم هست.

نویسنده: وحید نصیری

تاریخ: ۱۳۹۱/۰۵/۰۲ ۱۳:۲۰

یک EFCachingProvider رو می‌تونید اینجا ملاحظه کنید: ([^](#))
همچنین من [از این روش](#) راضی هستم.

نویسنده: مرتضی

تاریخ: ۱۳۹۱/۰۵/۲۰ ۳:۳۱

سلام

EF نسخه 6 از Net 4.0 با وجود Async پشتیبانی می‌کنه؟

نویسنده: وحید نصیری

تاریخ: ۱۳۹۱/۰۵/۲۰ ۹:۳۰

5 EF به بعد [بر مبنای](#) دات نت 4 و نیم است.
ویندوز 8 دات نت 4 و نیم سر خود است.
از دیدگاه تیم BCL، دات نت 4 و نیم یک [به روز رسانی درجای](#) دات نت 4 است و صد در صد با آن سازگاری دارد.

دات نت 4 و نیم فقط بر روی ویندوزهای ویستا سرویس پک 2 به بعد [قابل نصب است](#) (روی XP یا ویندوز سرور 2003 نصب نمی‌شود).

نویسنده: ایمان محمدی
تاریخ: ۱۱:۴۹ ۱۳۹۱/۰۵/۲۰

این که روی xp نصب نشه خیلی ناجوره عملا تو بخش application تا مدت‌ها بی استفاده می‌مونه ، بنظرتون این یک اهرم فشار برای حذف xp و سرور 2003 هست یا از لحاظ فنی جوابگو نبودن؟

نویسنده: علیرضا صالحی
تاریخ: ۱۵:۱۸ ۱۳۹۱/۰۵/۲۰

ویندوز ایکس پی در حال حذف شدن، هر چند خیلی‌ها هنوز در حال استفاده از اون هستند، ولی سرعت آپگرید کردن به 7 در حال زیاد شدن.

البته در ایران که هنوز سازمانهایی مانند تامین اجتماعی با فلاپی درایو سر و کار دارند، یک مقداری این مسئله مشکل ساز میشه.

نویسنده: ایمان محمدی
تاریخ: ۱۶:۴۶ ۱۳۹۱/۰۵/۲۰

دو گل سرسبد ایران یکی آموزش و پرورش یکی تامین اجتماعی که علاقتشون به foxpro و فلاپی تموم نمیشه ،ولی در نظر بگیرید به مشتری تون بگید (سازمانی یا عمومی) نرم افزار روی ویندوز xp نصب نمیشه! خودمم باشم قبول نمی‌کنم. با اینکه تمام سیستم‌ها رو معمولا به سون ارتقا میدیم ولی بعضی وقت‌ها سیستم قدیمیه و نمی‌کشه روش سون نصب کرد. نکته ای دیگه ای که وجود داره همه دنیا مثل ما پیشرفته و پول دار نیستند که روی همه کامپیوتر هاشون ویندوز سون ultimate نصب کنند.
پ.ن : [Make .NET 4.5 work on any OS that supports 4.0](#)

نویسنده: وحید نصیری
تاریخ: ۰:۲۴ ۱۳۹۱/۰۵/۲۶

EF 5 امروز [منتشر شد](#) و نکته مهم آن این است که با دات نت 4 هم سازگاری دارد. در دو نسخه دات نت 4 و دات نت 4 و نیم تهیه شده است.

البته اکثر قابلیت‌های جدید آن مخصوص دات نت 4 و نیم است مانند:

enum support
spatial data types
table-valued functions

نویسنده: محمد رضا کارونی
تاریخ: ۹:۲۵ ۱۳۹۱/۰۵/۲۶

سلام جناب مهندس نصیری،

می‌خواستم بدونم EF5 و MVC4 در نسخه‌های Express و ویژوال استودیو قابل نصب و بکارگیری می‌باشند یا خیر؟
در کل مایکروسافت برای ترویج عموم توسعه دهندگان به نوشتن app بر روی ویندوز 8 تا چه میزان بر روی نسخه‌های Express و ویژوال استودیو سرمایه گذاری و آینده نگری می‌کند؟

نویسنده: وحید نصیری
تاریخ: ۹:۳۵ ۱۳۹۱/۰۵/۲۶

EF5 چندتا DLL بیشتر نیست. این‌ها رو دریافت و به پروژه خودتون اضافه کنید.

- نسخه express مخصوص vs2012 هم موجود است ([^](#)).

نویسنده: اژدری
تاریخ: ۱۳۹۱/۰۶/۱۳ ۱۰:۴۳

بسیار هم عالی

نام‌گذاری (**Naming**) اشیا یک برنامه شاید در نگاه اول دارای اهمیت بالایی نباشه، اما تجربه نشون داده که در پروژه‌های بزرگ که با کمک چندین مجموعه به انجام میرسه نام‌گذاری صحیح و اصولی که از یکسری قواعد کلی و مناسب پیروی میکنه میتونه به پیشبرد اهداف و مدیریت راحتتر برنامه کمک بسیاری بکنه. بیشتر موارد اشاره شده در این مطلب از کتاب جامع و مفید [Framework Design Guidelines](#) اقتباس شده که خوندن این کتاب مفید رو به خوانندگان توصیه میکنم.

برای کمک به نوشتن اصولی و راحتتر سورسهای برنامه‌ها در ویژوال استودیو نرم افزارهای متعددی وجود داره که با توجه به تجربه شخصی خودم نرم افزار [Resharper](#) محصول شرکت [Jetbrains](#) یکی از بهترین هاست که در مورد خاص مورد بحث در این مطلب نیز بسیار خوب عمل میکنه.

برخی از موارد موجود در مطلب جاری نیز از قراردادهای پیشفرض موجود در نرم افزار Resharper نسخه 6.0 برگرفته شده است و قسمتی نیز از تجربه شخصی خودم و سایر دوستان و همکاران بوده است.

اصل این مطلب حدود یکسال پیش تهیه شده و اگر نقایصی وجود داره لطفا اشاره کنین.

اصول و قراردادهای نام‌گذاری در دات‌نت

انواع نام‌گذاری

نام‌گذاری اشیا در حالت کلی را می‌توان به سه روش زیر انجام داد:

1. **Pascal Casing** : در این روش حرف اول هر کلمه در نام شی به صورت بزرگ نوشته می‌شود.

```
FirstName
```

2. **camel Casing** : حرف اول در اولین کلمه نام هر شی به صورت کوچک و حرف اول بقیه کلمات به صورت بزرگ نوشته می‌شود.

```
firstName
```

3. **Hungarian** : در این روش برای هر نوع شی موجود یک پیشوند در نظر گرفته می‌شود تا از روی نام شی بتوان به نوع آن پی برد. در ادامه و پس از این پیشوندها سایر کلمات بر اساس روش Pascal Casing نوشته می‌شوند.

```
strFirstName  
lblFirstName
```

نکته: استفاده از این روش به جز در نام‌گذاری کنترل‌های UI منسوخ شده است.

قراردادهای کلی

1. نباید نام اشیا تنها در بزرگ یا کوچک بودن حروف با هم فرق داشته باشند. به عنوان مثال نباید دو کلاس با نام‌های MyClass و myClass داشته باشیم. هرچند برخی از زبان‌ها case-sensitive هستند اما برخی دیگر نیز چنین قابلیتی ندارند (مثل VB.NET). بنابراین اگر بخواهیم کلاس‌های تولیدی ما در تمام محیط‌ها و زبان‌های برنامه نویسی قابل اجرا باشند باید از این قرارداد پیروی کنیم.

2. تا آنجا که امکان دارد باید از به‌کار بردن مخفف کلمات در نام‌گذاری اشیا دوری کنیم. مثلاً به جای استفاده از GetChr باید از

GetCharacter استفاده کرد.

البته در برخی موارد که مخفف واژه موردنظر کاربرد گسترده ای دارد می‌توان از عبارت مخفف نیز استفاده کرد. مثل UI به جای UserInterface و یا IO به جای InputOutput.

آ. اصول نام‌گذاری فضای نام (namespace)

1. اساس نام‌گذاری فضای نام باید از قاعده زیر پیروی کند:

```
<Company>.<Technology|Product|Project>[.<Feature>][.<SubNamespace>]
```

(> : اجباری [] : اختیاری)

2. برای نام‌گذاری فضای نام باید از روش Pascal Casing استفاده شود.

3. در هنگام تعریف فضاهای نام به وابستگی آنها توجه داشته باشید. به عنوان مثال اشیای درون یک فضای نام پدر نباید به اشیای درون فضای نام یکی از فرزندانش وابسته باشد. مثلاً در فضای نام System نباید اشیایی وجود داشته باشند که به اشیای درون فضای نام System.UI وابسته باشند.

4. سعی کنید از نام‌ها به صورت جمع برای عناوین فضای نام استفاده کنید. مثلاً به جای استفاده از Kara.CSS.HQ.Manager.Entity از Kara.CSS.HQ.Manager.Entities استفاده کنید. البته برای مواردی که از عناوین مخفف و یا برندهای خاص استفاده کرده‌اید از این قرارداد پیروی نکنید. مثلاً نباید از عنوانی شبیه به Kara.CSS.Manager.IOs استفاده کنید.

5. از عنوانی پایدار و مستقل از نسخه محصول برای بخش دوم عنوان فضای نام استفاده کنید. بدین معنی که این عناوین با گذر زمان و تغییر و تحولات در محتوای محصول و یا تولیدکننده نباید تغییر کنند. مثال:

```
Microsoft.Reporting.WebForms
Kara.Support.Manager.Enums
Kara.CSS.HQ.WebUI.Configuration
```

6. از عناوین یکسان برای فضای نام و اشیای درون آن استفاده نکنید. مثلاً نباید کلاسی با عنوان Manager در فضای نام Kara.CSS.Manager وجود داشته باشد.

7. از عناوین یکسان برای اشیای درون فضاهای نام یک برنامه استفاده نکنید. مثلاً نباید دو کلاس با نام Package در فضاهای نام Kara.CSS.Manger.Entities و Kara.CSS.Manger داشته باشید.

ب. اصول نام‌گذاری کلاس‌ها و Structها

1. عنوان کلاس باید اسم یا موصوف باشد.

2. در نام‌گذاری کلاس‌ها باید از روش Pascal Casing استفاده شود.

3. نباید از عناوین مخففی که رایج نیستند استفاده کرد.

4. از پیشوندهای زائد مثل C یا C1s نباید استفاده شود.

5. نباید از کاراکترهایی به غیر از حروف (و یا در برخی موارد خیلی خاص، شماره نسخه) در نام‌گذاری کلاس‌ها استفاده شود. مثال:

درست:

```
PackageManager , PackageConfigGenerator
Circle , Utility , Package
```

نادرست:

```
CreateConfig , classdata
CManager , C1sPackage , Config_Creator , Config1389
```

6. در نام‌گذاری اشیای Generic از استفاده از عناوینی چون Element, Node, Log و یا Message پرهیز کنید. این کار موجب به‌وجودآمدن کانفلیکت در عناوین اشیا می‌شود. در این موارد بهتر است از عناوینی چون FormElement, XmlNode, EventLog و

SoapMessage استفاده شود. درواقع بهتر است نام اشیای جنریک، برای موارد موردنیاز، کاملاً اختصاصی و درعین حال مشخص‌کننده محتوا و کاربرد باشند.

7. از عناوینی مشابه عناوین کتابخانه‌های پایه دات‌نت برای اشیای خود استفاده نکنید. مثلاً نباید کلاسی با عنوان Console, Parameter, Action و یا Data را توسعه دهید. همچنین از کاربرد عناوینی مشابه اشیای موجود در فضاهای نام غیرپایه دات‌نت و یا غیردات‌نتی اما مورد استفاده در پروژه خود که محتوا و کاربردی مختص همان فضای نام دارند پرهیز کنید. مثلاً نباید کلاسی با عنوان Page در صورت استفاده از فضای نام System.Web.UI تولید کنید.

8. سعی کنید در مواردی که مناسب به نظر می‌رسد از عنوان کلاس پایه در انتهای نام کلاس‌های مشتق‌شده استفاده کنید. مثلاً FileStream که از کلاس Stream مشتق شده است. البته این قاعده در تمام موارد نتیجه مطلوب ندارد. مثلاً کلاس Button که از کلاس Control مشتق شده است. بنابراین در به‌کاربردن این مورد بهتر است تمام جوانب و قواعد را در نظر بگیرید.

پ. اصول نام‌گذاری مجموعه‌ها (Collections)

یک مجموعه در واقع یک نوع کلاس خاص است که حاوی مجموعه‌ای از داده‌هاست و از همان قوانین کلاس‌ها برای نام‌گذاری آن‌ها استفاده می‌شود.

1. بهتر است در انتهای عنوان مجموعه از کلمه Collection استفاده شود. مثال:

```
CenterCollection , PackageCollection
```

2. البته در صورتی که کلاس مورد نظر ما رابط IDictionary را پیاده‌سازی کرده باشد بهتر است از پسوند Dictionary استفاده شود.

ت. اصول نام‌گذاری Delegate‌ها

1. عنوان یک delegate باید اسم یا موصوف باشد.

2. در نام‌گذاری delegate‌ها باید از روش Pascal Casing استفاده شود.

3. نباید از عناوین مخفی که رایج نیستند استفاده کرد.

4. از پیشوندهای زائد مثل D یا del نباید استفاده شود.

5. نباید از کاراکترهایی به غیر از حروف در نام‌گذاری delegate‌ها استفاده شود.

6. نباید در انتهای نام یک delegate از عبارت Delegate استفاده شود.

7. بهتر است که در صورت امکان در انتهای نام یک delegate که برای Event Handler استفاده نمی‌شود از عبارت Callback استفاده شود. البته تنها در صورتی که معنی و مفهوم مناسب را داشته باشد.

مثال:

نحوه تعریف یک delegate

```
public delegate void Logger (string log);  
public delegate void LoggingCallback (object sender, string reason);
```

ث. اصول نام‌گذاری رویدادها (Events)

1. عنوان یک رویداد باید فعل یا مصدر باشد.

2. در نام‌گذاری کلاس‌ها باید از روش Pascal Casing استفاده شود.

3. نباید از عناوین مخفی که رایج نیستند استفاده کرد.

4. از پیشوندهای زائد نباید استفاده شود.

5. نباید از کاراکترهایی به غیر از حروف در نام‌گذاری رویدادها استفاده شود.

6. بهتر است از پسوند EventHandler در عنوان هندلر رویداد استفاده شود.

7. از به‌کاربردن عباراتی چون AfterXXX و یا BeforeXXX برای نمایش رویدادهای قبل و یا بعد از رخداد خاصی خودداری شود. به جای آن باید از اسم مصدر (شکل یندار فعل) برای نام‌گذاری رویداد قبل از رخداد و همچنین شکل گذشته فعل برای نام‌گذاری رویداد بعد از رخداد خاص استفاده کرد.

مثلاً اگر کلاسی دارای رویداد Open باشد باید از عنوان Opening برای رویداد قبل از Open و از عنوان Opened برای رویداد بعد از

Open استفاده کرد.

8. نباید از پیشوند On برای نام‌گذاری رویداد استفاده شود.

9. همیشه پارامتر e و sender را برای آرگومانهای رویداد پیاده سازی کنید. sender که از نوع object است نمایش دهنده شیئی است که رویداد مربوطه را به وجود آورده است و e درواقع آرگومانهای رویداد مربوطه است.

10. عنوان کلاس آرگومانهای رویداد باید دارای پسوند EventArgs باشد.

مثال:

عنوان کلاس آرگومان:

AddEventArgs , EditEventArgs , DeleteEventArgs

عنوان رویداد:

Adding , Add , Added

تعریف یک EventHandler:

```
public delegate void <EventName>EventHandler
```

```
(object sender, <EventName>EventArgs e);
```

نکته: نیاز به تولید یک EventHandler مختص توسعه یک برنامه به‌ندرت ایجاد می‌شود. در اکثر موارد می‌توان با استفاده از کلاس جنریک EventHandler<TEventArgs> تمام احتیاجات خود را برطرف کرد.
تعریف یک رویداد:

```
public event EventHandler
```

```
<AddEventArgs> Adding;
```

ج. اصول نام‌گذاری Attributeها

1. عنوان یک attribute باید اسم یا موصوف باشد.
 2. در نام‌گذاری attributeها باید از روش Pascal Casing استفاده شود.
 3. نباید از عناوین مخفی که رایج نیستند استفاده کرد.
 4. از پیشوندهای زائد مثل A یا atr نباید استفاده شود.
 5. نباید از کاراکترهایی به غیر از حروف در نام‌گذاری attributeها استفاده شود.
 6. بهتر است در انتهای نام یک attribute از عبارت Attribute استفاده شود.
- مثال:

DisplayNameAttribute , MessageTypeAttribute

ج. اصول نام‌گذاری Interfaceها

1. در ابتدای عنوان interface باید از حرف I استفاده شود.
 2. نام باید اسم، موصوف یا صفتی باشد که interface را توصیف می‌کند.
- به عنوان مثال:

IComponent (اسم)
IConnectionProvider (موصوف)
ICloneable (صفت)

3. از روش Pascal Casing استفاده شود.

4. خودداری از بکاربردن عبارات مخفف غیررایج.

5. باید تنها از کاراکترهای حرفی در نام interface استفاده شود.

ح. اصول نام‌گذاری Enumerationها

1. استفاده از روش Pascal Casing

2. خودداری از کاربرد عبارات مخفف غیررایج

3. تنها از کاراکترهای حرفی در نام Enumretionها استفاده شود.

4. نباید از پسوند یا پیشوند Enum یا Flag استفاده شود.

5. اعضای یک Enum نیز باید با روش Pascal Casing نام‌گذاری شوند.

مثال:

```
public enum FileMode {
    Append,
    Read, ...
}
```

6. در صورتی‌که enum موردنظر از نوع flag نیست باید عنوان آن مفرد باشد.

7. در صورتی‌که enum موردنظر برای کاربرد flag طراحی شده باشد نام آن باید جمع باشد. مثال:

```
[Flag]
public enum KeyModifiers {
    Alt = 1,
    Control = 2,
    Shift = 4
}
```

8. از به‌کاربردن پسوند و یا پیشوندهای اضافه در نام‌گذاری اعضای یک enum نیز پرهیز کنید. مثلاً نام‌گذاری زیر نادرست است:

```
public enum OperationState {
    DoneState,
    FaultState,
    RollbackState
}
```

خ. اصول نام‌گذاری متدها

1. نام متد باید فعل یا ترکیبی از فعل و اسم یا موصوف باشد.

2. باید از روش Pascal Casing استفاده شود.

3. خودداری از بکاربردن عبارات مخفف غیررایج و یا استفاده زیاد از اختصار

4. تنها از کاراکترهای حرفی برای نام متد استفاده شود.

مثال:

```
AddDays , Save , DeleteRow , BindData , Close , Open
```

د. اصول نام‌گذاری Propertyها

1. نام باید اسم، صفت یا موصوف باشد.

2. باید از روش Pascal Casing استفاده شود.

3. خودداری از بکاربردن عبارات مخفف غیررایج.

4. تنها از کاراکترهای حرفی برای نام‌گذاری پراپرتی استفاده شود.

مثال:

```
Radius , ReportType , DataSource , Mode , CurrentCenterId
```

5. از عبارت Get در ابتدای هیچ Property ای استفاده نکنید.

6. نام خاصیت‌هایی که یک مجموعه برمی‌گرداند باید به صورت جمع باشد. عنوان این Property ها نباید به صورت مفرد به همراه پسوند Collection یا List باشد. مثال:

```
public CenterCollection Centers { get; set; }
```

7. خواص Boolean را با عناوینی مثبت پیاده‌سازی کنید. مثلاً به جای استفاده از CantRead از CanRead استفاده کنید. بهتر است این Property ها پیشوندهایی چون Is, Can یا Has داشته باشند، البته تنها در صورتی که استفاده از چنین پیشوندهایی ارزش افزوده داشته و مفهوم آن را بهتر برساند. مثلاً عنوان CanSeek مفهوم روشن‌تری نسبت به Seekable دارد. اما استفاده از Created خیلی بهتر از IsCreated است یا Enabled کاربرد به مراتب راحت‌تری از IsEnabled دارد. برای تشخیص بهتر این موارد بهتر است از روش if سنجی استفاده شود. به عنوان مثال

```
if (list.Contains(item))
if (regularExpression.Matches(text))
if (stream.CanSeek)
if (context.Created)
if (form.Enabled)
```

مفهوم درست‌تری نسبت به موارد زیر دارند:

```
if (list.IsContains(item))
if (regularExpression.Match(text))
if (stream.Seekable)
if (context.IsCreated)
if (form.IsEnabled)
```

8. بهتر است در موارد مناسب عنوان Property با نام نوعش برابر باشد. مثلاً

```
public Color Color { get; set; }
```

د. اصول نام‌گذاری پارامترها

پارامتر در حالت کلی به آرگومان و روی تعریف شده برای یک متد گفته می‌شود.

1. حتماً از یک نام توصیفی استفاده شود. از نام‌گذاری پارامترها براساس نوعشان به شدت پرهیز کنید.
 2. از روش camel Casing استفاده شود.
 3. تنها از کاراکترهای حرفی برای نام‌گذاری پارامترها استفاده شود.
- مثال:

```
firstName , e , id , packageId , centerName , name
```

4. نکاتی برای نام‌گذاری پارامترهای **Operator Overloading** :

- برای operator های دو پارامتری (binary operators) از عناوین left و right برای پارامترهای آن استفاده کنید:

```
public static MyType operator +(MyType left, MyType right)
public static bool operator ==(MyType left, MyType right)
```

- برای operator های تک‌پارامتری (unary operators) اگر برای پارامتر مورد استفاده هیچ عنوان توصیفی مناسبی پیدا نکردید حتماً از عبارت value استفاده کنید:

```
public static MyType operator ++(MyType value)
```

- در صورتی که استفاده از عناوین توصیفی دارای ارزش افزوده بوده و خوانایی کد را بهتر می‌کند حتماً از این نوع عناوین استفاده کنید:

```
public static MyType operator /(MyType dividend, MyType divisor)
```

- نباید از عبارات مخفف یا عناوینی با اندیس‌های عددی استفاده کنید:

```
public static MyType operator -(MyType d1, MyType d2) // incorrect!
```

ر. اصول نام‌گذاری متغیر (Variable)ها

- نام متغیر باید اسم، صفت یا موصوف باشد.

- نام‌گذاری متغیرها باید با توجه به نوع آن انجام شود.

1. متغیرهای عمومی (public) و protected و Constantها

- استفاده از روش Pascal Casing

- تنها از کاراکترهای حرفی برای نام متغیر عمومی استفاده شود.

مثال:

```
Area , DataBinder , PublicCacheName
```

2. متغیرهای private (در سطح کلاس یا همان field)

- نام این نوع متغیر باید با یک "_" شروع شود.

- از روش camel Casing استفاده شود.

مثال:

```
_centersList  
_firstName  
_currentCenter
```

3. متغیرهای محلی در سطح متد

- باید از روش camel Casing استفاده شود.

- تنها از کاراکترهای حرفی استفاده شود.

مثال:

```
parameterType , packageOperationTypeId
```

ز. اصول نام‌گذاری کنترل‌های UI

1. نام باید اسم یا موصوف باشد.

2. استفاده از روش Hungarian !

3. عبارت مخفف معرفی‌کننده کنترل، باید به اندازه کافی برای تشخیص نوع آن مناسب باشد.

مثال:

```
lblName (Label)  
txtHeader (TextBox)  
btnSave (Button)
```

ژ. اصول نام‌گذاری Exceptionها

تمام موارد مربوط به نام‌گذاری کلاس‌ها باید در این مورد رعایت شود.

1. باید از پسوند Exception در انتهای عنوان استفاده شود.

مثال:

```
ArgumentNullException , InvalidOperationExpection
```

س. نام‌گذاری اسمبلی‌ها و DLLها

1. عناوینی که برای نام‌گذاری اسمبلی‌ها استفاده می‌شوند، باید نمایش‌دهنده محتوای کلی آن باشند. مثل:

```
System.Data
```

2. از روش زیر برای نام‌گذاری اسمبلی‌ها استفاده شود:

```
<Company>.<Component>.dll  
<Company>.<Project|Product|Technology>.<Component>.dll
```

مثال:

```
Microsoft.CSharp.dll , Kara.CSS.Manager.dll
```

ش. نام‌گذاری پارامترهای نوع (Generic (type parameter

1. از حرف T برای پارامترهای تک‌حرفی استفاده کنید. مثل:

```
public int IComparer<T> {...}  
public delegate bool Predicate<T> (T item)
```

2. تمامی پارامترهای جنریک را با عناوینی توصیفی و مناسب که مفهوم و کاربرد آنرا برساند نام‌گذاری کنید، مگر آنکه یافتن چنین عباراتی ارزش افزوده‌ای در روشن‌تر کردن کد نداشته باشد. مثال:

```
public int ISessionChannel<TSession> {...}  
public delegate TOutput Converter<TInput, TOutput> (TInput from)  
public class Nullable<T> {...}  
public class List<T> {...}
```

3. در ابتدای عناوین توصیفی حتما از حرف T استفاده کنید.

4. بهتر است تا به صورتی روشن نوع قید قرار داده شده بر روی پارامتری خاص را در نام آن پارامتر نمایش دهید. مثلا اگر قید ISession را برای پارامتری قرار دادید بهتر است نام آن پارامتر را TSession در نظر بگیرید.

ص. نام‌گذاری کلید Resourceها

به دلیل شباهت ساختاری که میان کلیدهای resource و property وجود دارد قواعد نام‌گذاری propertyها در اینجا نیز معتبر هستند.

1. از روش نام‌گذاری Pascal Casing برای کلیدهای resource استفاده کنید.

2. از عناوین توصیفی برای این کلیدها استفاده کنید. سعی کنید تا حد امکان به هیچ وجه از عناوین کوتاه و یا مخففی که مفهوم را به صورت ناکامل می‌رساند استفاده نکنید. درواقع سعی کنید که خوانایی بیشتر کد را فدای فضای بیشتر نکنید.

3. از کلیدواژه‌های CLR و یا زبان مورد استفاده برای برنامه‌نویسی در نام‌گذاری این کلیدها استفاده نکنید.

4. تنها از حروف و اعداد و _ در نام‌گذاری این کلیدها استفاده کنید.

5. سعی کنید از عناوین توصیفی همانند زیر برای پیام‌های مناسب خطاها جهت نمایش به کاربر برای کلیدهای مربوطه استفاده کنید. درواقع نام کلید باید ترکیبی از نام نوع خطا و یک آی‌دی مشخص‌کننده پیغام مربوطه باشد:

```
ArgumentExceptionIllegalCharacters
ArgumentExceptionInvalidName
ArgumentExceptionFileNotFoundException
```

نکاتی درمورد کلمات مرکب

کلمات مرکب به کلماتی گفته می‌شود که در آن از بیش از یک کلمه با مفهوم مستقل استفاده شده باشد. مثل Callback یا FileName.

باید توجه داشت که با تمام کلمات موجود در یک کلمه مرکب نباید همانند یک کلمه مستقل رفتار کرد و حرف اول آن را در روش‌های نام‌گذاری موجود به‌صورت بزرگ نوشت. کلمات مرکبی وجود دارند که به آن‌ها closed-form گفته می‌شود. این کلمات مرکب با اینکه از 2 یا چند کلمه دارای مفهوم مستقل تشکیل شده‌اند اما به‌خودی‌خود دارای مفهوم جداگانه و مستقلی هستند. برای تشخیص این کلمات می‌توان به فرهنگ لغت مراجعه کرد (و یا به‌سادگی از نرم‌افزار Microsoft Word استفاده کرد) و دریافت که آیا کلمه مرکب موردنظر آیا مفهوم مستقلی برای خود دارد، یعنی درواقع آیا عبارتی closed-form است. با کلمات مرکب از نوع closed-form همانند یک کلمه ساده برخورد می‌شود!

در جدول زیر مثال‌هایی از عبارات رایج و نحوه درست و نادرست استفاده از هریک نشان داده شده است.

Wrong	camel Casing	Pascal Casing
CallBack	callback	Callback
Bitflag / bitflag	bitFlag	BitFlag
Cancelled	canceled	Canceled
Donot / Don't	doNot	DoNot
EMail	email	Email
EndPoint / endPoint	endpoint	Endpoint
Filename / filename	fileName	FileName
GridLine / gridLine	gridline	Gridline
HashTable / hashTable	hashtable	Hashtable
ID	id	Id
Indices	indexes	Indexes

Wrong	camel Casing	Pascal Casing
Logoff / LogOut !	logOff	LogOff
Logon / LogIn !	logOn	LogOn
Signout / SignOff	signOut	SignOut
Signin / SignOn	signIn	SignIn
MetaData / metaData	metadata	Metadata
MultiPanel / multiPanel	multipanel	Multipanel
MultiView / multiView	multiview	Multiview
NameSpace / nameSpace	namespace	Namespace
OK	ok	Ok
PI	pi	Pi
PlaceHolder / placeHolder	placeholder	Placeholder
Username / username	username	UserName
Whitespace / whitespace	whiteSpace	WhiteSpace
Writeable / writeable	writable	Writable

همان‌طور که در جدول بالا مشاهده می‌شود در استفاده از قوانین عبارات مخفف دو مورد استثنا وجود دارد که عبارتند از Id و Ok. این کلمات باید همان‌طور که در اینجا نشان داده شده‌اند استفاده شوند.

نکاتی درباره عبارات مخفف

1. عبارات مخفف (Acronym) با خلاصه‌سازی کلمات (Abbreviation) فرق دارند. یک عبارت مخفف شامل حروف اول یک عبارت طولانی یا معروف است، در صورتی‌که خلاصه‌سازی یک عبارت یا کلمه از حذف بخشی از آن به‌دست می‌آید. تا آنجا که امکان دارد از خلاصه‌سازی عبارات نباید استفاده شود. همچنین استفاده از عبارات مخفف غیررایج توصیه نمی‌شود.

مثال: UI و IO

2. براساس تعریف، یک مخفف حداقل باید 2 حرف داشته باشد. نحوه برخورد با مخفف‌های دارای بیشتر از 2 حرف با مخفف‌های دارای 2 حرف باهم متفاوت است. با مخفف‌های دارای 2 حرف همانند کلمه‌ای یک حرفی! برخورد می‌شود، در صورتی‌که با سایر مخفف‌ها همانند یک کلمه کامل چند حرفی برخورد می‌شود. به‌عنوان مثال IOStream برای روش PascalCasing و ioStream برای

روش camelCasing استفاده می‌شود. همچنین از HtmlBody و htmlBody به ترتیب برای روش‌های Pascal و camel استفاده می‌شود.

3. هیچ‌کدام از حروف یک عبارت مخفف در ابتدای یک واژه نام‌گذاری‌شده به روش camelCasing به صورت بزرگ نوشته نمی‌شود!

نکته : موارد زیادی را می‌توان یافت که در ابتدا به نظر می‌رسد برای پیاده‌سازی آنها باید قوانین فوق را نقض کرد. این موارد شامل استفاده از کتابخانه‌های سایر پلتفرم‌ها (مثل HTML, MFC و غیره)، جلوگیری از مشکلات جغرافیایی! (مثلا در مورد نام کشورها یا سایر موقعیت‌های جغرافیایی)، احترام به نام افراد در گذشته، و مواردی از این دست می‌شود. اما در بیشتر قریب به اتفاق این موارد هم می‌توان بدون نقض قوانین فوق اقدام به نام‌گذاری اشیا کرد، بدون اینکه با تغییر عبارت اصلی (که موجب تطابق با این قوانین می‌شود) لطمه‌ای به مفهوم آن بزند. البته تنها موردی که به نظر می‌رسد می‌تواند قوانین فوق را نقض کند نام‌های تجاری هستند. البته استفاده از نام‌های تجاری توصیه نمی‌شود، چون این نام‌ها سریع‌تر از محتوای کتابخانه‌های برنامه‌نویسان تغییر می‌کنند! نکته: برای درک بهتر قانون "عدم استفاده از عبارات مخففی که رایج نیستند" مثالی از زبان توسعه دهندگان داتانت فریمورک ذکر می‌شود. در کلاس Color متد زیر با Overloadهای مختلف در دسترس است:

```
public class Color {
    ...
    public static Color FromArgb(...)
    { ... }
}
```

همان‌طور که مشاهده می‌شود برای رعایت قوانین فوق به جای استفاده از ARGB از عبارت Argb استفاده شده است. اما این نحوه استفاده موجب شده تا این سوال به ظاهر خنده‌دار اما درست پیش‌آید:

"چطور می‌شود در این کلاس رنگی را از ARGB تبدیل کرد؟ هرچه که من میبینم فقط تبدیل از طریق (Argb) آرگومان b است!" حال در این نقطه به نظر می‌رسد که در اینجا باید قوانین فوق را نقض کرد و از عنوان FromARGB که مفهوم درست را می‌رساند استفاده کرد. اما با کمی دقت متوجه می‌شویم که این قوانین در ابتدا نیز با پیاده‌سازی نشان داده شده در قطعه کد بالا نقض شده‌اند! همه می‌دانیم که عبارت RGB مخفف معروفی برای عبارت Red Green Blue است. اما استفاده از ARGB برای افزودن کلمه Alpha به ابتدای عبارت مذکور چندان رایج نیست. پس استفاده از مخفف Argb از همان ابتدا اشتباه به نظر می‌رسد. بنابراین راه‌حل بهتر می‌تواند استفاده از عنوان FromAlphaRgb باشد که هم قوانین فوق را نقض نکرده و هم مفهوم را بهتر می‌رساند.

دیگر نکات

1. قراردادهای اشاره شده در این سند حاصل کار شبانه روزی تعداد بسیاری از برنامه‌نویسان در سرتاسر جهان در پروژه‌های بزرگ بوده است. این اصول کلی تنها برای توسعه آسان‌تر و سریع‌تر پروژه‌های بزرگ تعیین شده‌اند و همان‌طور که روشن است تنها از طریق تجربه دست‌یافتنی هستند. بنابراین چه بهتر است که در این راه از تجارب بزرگان این عرصه بیشترین بهره‌برده شود.
2. همچنین توجه داشته باشید که بیشتر قراردادهای اشاره شده در این سند از راهنمای نام‌گذاری تیم توسعه BCL داتانت فریمورک گرفته شده است. این تیم طبق اعتراف خودشان زمان بسیار زیادی را برای نام‌گذاری اشیا صرف کرده‌اند و توصیه کرده‌اند که دیگران نیز برای نام‌گذاری، زمان مناسب و کافی را در توسعه پروژه‌ها در نظر بگیرند.

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۱:۳۶ ۱۳۹۱/۰۵/۱۸

ضمن تشکر از مطلب مفید شما، علاوه بر ReSharper که می‌تونه در دراز مدت اثر ذهنی قابل ملاحظه‌ای در تطابق با اصول نامگذاری داشته باشه، نرم افزار FxCop هم یک سری از مواردی را که ReSharper تشخیص نمی‌ده می‌تونه به خوبی گزارش بده:

```
CA1717:OnlyFlagsEnumsShouldHavePluralNames
CA1704:IdentifiersShouldBeSpelledCorrectly
CA1709:IdentifiersShouldBeCasedCorrectly
CA1702:CompoundWordsShouldBeCasedCorrectly
...
```

نویسنده: حسین مرادی نیا
تاریخ: ۹:۴۷ ۱۳۹۱/۰۵/۱۹

سلام

مرسی بابت مطلب خوبتون

اما یه سوال

Resharper معمولا توصیه میکنه که برای متغیرهای محلی از this استفاده نشه. حالا میخوامستم نظرتون رو در این زمینه بدونم. استفاده از this خوبه یا بد؟

نویسنده: یوسف نژاد
تاریخ: ۲۱:۲۷ ۱۳۹۱/۰۵/۱۹

به نظر من استفاده بی مورد و اضافی از this اشتباهه. فقط در موارد لازم برای از بین بردن کانفلیکت (مثلا بین نامهای فیلدها و پارامترها) باید استفاده بشه. هرچند اگه اصول و قراردادهای رعایت بشه معمولا این تضادها و کانفلیکتهای پیش نیامد.

نویسنده: حسین
تاریخ: ۱۲:۱ ۱۳۹۱/۰۷/۲۴

خیلی ممنون بابت مطلب مفیدتون. خواهشاً یک مطلب هم درباره نام گذاری پروژه ها قرار بدید (در مواردی که یک پروژه به چند Class Library و ... تقسیم میشود). ممنون.

نویسنده: محمد علی
تاریخ: ۱۱:۳۳ ۱۳۹۱/۱۱/۲۸

عالی بود. ممنون

نویسنده: M.Q
تاریخ: ۲۰:۴۷ ۱۳۹۲/۰۲/۰۵

با سلام و تشکر

اگه یک متد داشته باشیم که این متد پارامتری با نام مثلا (id) داشته باشد و ما بخواهیم به هر دلیلی متغیری محلی برای نگهداری "کد" در بدنه متد نیز داشته باشیم، نام گذاری متغیر محلی چگونه باید باشد؟

```
public void Save(int id, string name)
{
    int id_ = id; // ?
```



```
/*  
ادامه دستورات  
*/  
}
```

با تشکر

نویسنده: محسن خان
تاریخ: ۲۱:۳۹ ۱۳۹۲/۰۲/۰۵

روش خاصی نداره. فقط همان اصول کلی نامگذاری متغیرها در اینجا نیز باید رعایت شود. مثلا localId خوبه.

نام قوی (Strong Name یا به صورت 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 به عنوان ابزاری برای فعال‌سازی امنیت می‌پندارند، در صورتی که «نام قوی» در واقع یک تکنولوژی تعیین «هویت منحصر به فرد» اسمبلی‌ها است. یک نام قوی حاوی مجموعه‌ای از مشخصات یک اسمبلی (شامل نام ساده، نسخه و داده‌های کالچر (culture) آن در صورت وجود) به همراه 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) استفاده کرد. البته در حال حاضر امکان توسعه این نوع از اسمبلی‌ها در ویژوال استودیو موجود نیست.) در sdkهای میکروسافت 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 منحصر به فرد خواهد شد. به طور کلی نام‌های قوی نیازمندی‌های زیر را برطرف می‌کنند:

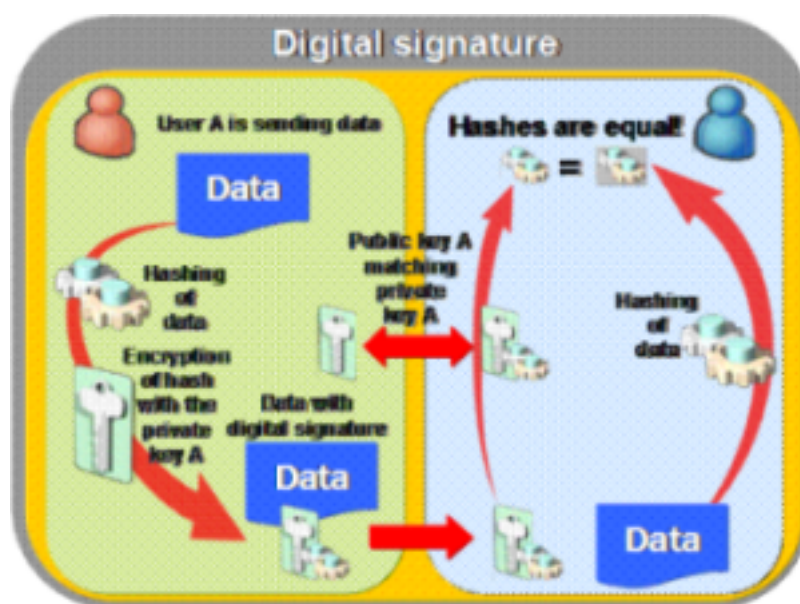
- نام‌های قوی منحصر به فرد بودن نام یک اسمبلی را براساس جفت‌کلیدهای یکتا فراهم می‌کنند. هیچ‌کس دیگری امکان تولید همان اسمبلی‌ای را که شما تولید کرده‌اید ندارد، زیرا اسمبلی‌ای که با یک کلید خصوصی تولید شده است نسبت به اسمبلی دیگری که با یک کلید خصوصی دیگر تولید شده است نام متفاوتی خواهد داشت چون کلید 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.

هنگامی که به یک اسمبلی 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. در واقع 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 جفت کلید 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 هش کردن (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 در دسترس 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 توزیع کننده تولید می‌شود. سپس این دو مقدار محاسبه شده در سمت مصرف کننده با یکدیگر مقایسه شده و در صورت 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 جفت کلید این است که به صورت 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 جفت کلیدهای قوی- نام‌گذاری شده (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 این کار 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 این کار 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 تقریباً مطمئن بود که تا زمانی که 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.

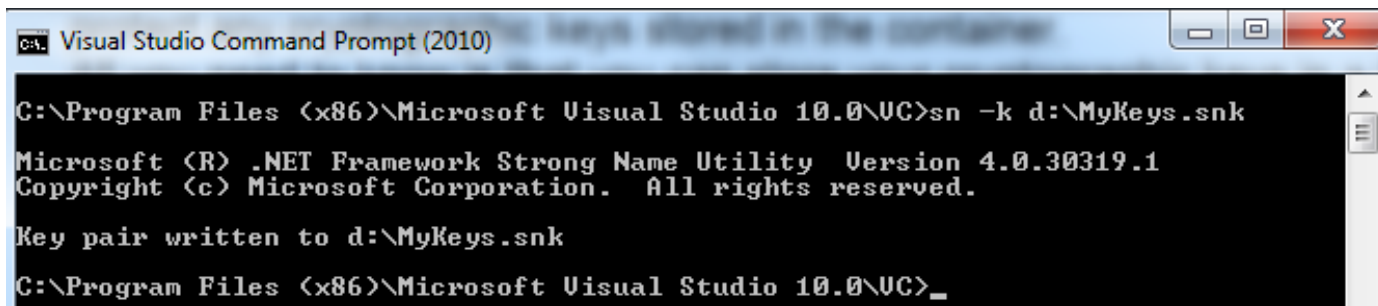
برنامه sn به همراه SDKهای ویندوز نصب می‌شود. البته با نصب ویژوال استودیو تمام SDKهای موردنیاز مطابق با نسخه‌های

موجود، نصب خواهد شد. مسیر نسخه 4 و 32 بیتی این برنامه در سیستم عامل Windows 7 به صورت 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 جفت کلید جدید تولید شده و در فایل MyKeys.snk در ریشه درایو d: ذخیره می شود:

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



```
Visual Studio Command Prompt (2010)
C:\Program Files (x86)\Microsoft Visual Studio 10.0\UC>sn -k d:\MyKeys.snk
Microsoft (R) .NET Framework Strong Name Utility Version 4.0.30319.1
Copyright (c) Microsoft Corporation. All rights reserved.
Key pair written to d:\MyKeys.snk
C:\Program Files (x86)\Microsoft Visual Studio 10.0\UC>_
```

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

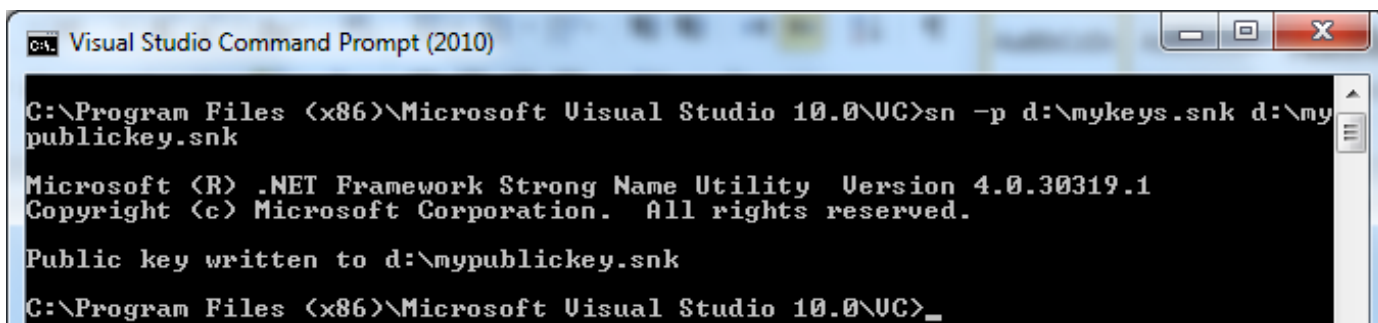
این کار یک جفت کلید کریپتوگرافی 1024 بیتی به صورت تصادفی تولید می کند. این دستور را باید در خط فرمانی (Command Prompt) اجرا نمود که مسیر فایل sn.exe را بداند. برای راحتی کار می توان از خط فرمان ویژوال استودیو (Visual Studio Command Prompt) استفاده کرد.

نکته : اجرای عملیات فوق در یک شرکت یا قسمت توسعه یک شرکت، تنها یک بار نیاز است زیرا تمام اسمبلی های تولیدی تا زمانی که عناوین ساده متمایزی دارند می توانند از یک جفت کلید مشترک استفاده کنند.

نکته : هر چند که می توان از پسوند های دیگری نیز برای نام فایل حاوی جفت کلید استفاده کرد، اما توصیه می شود از همین پسوند snk. استفاده شود.

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

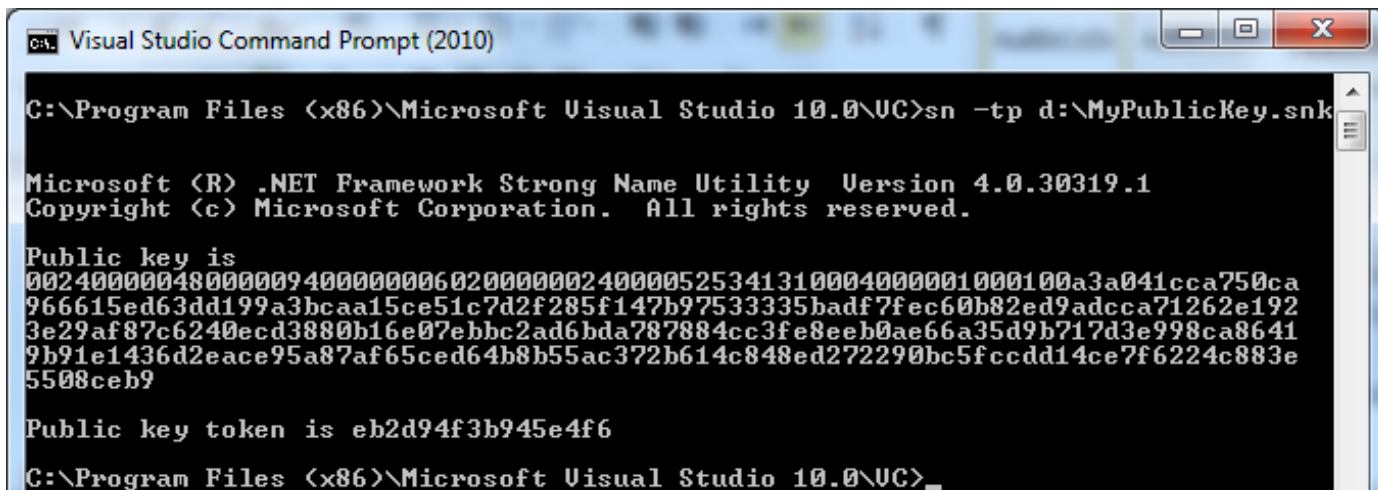
```
sn -p d:\mykeys.snk d:\mypublickey.snk
```



```
Visual Studio Command Prompt (2010)
C:\Program Files (x86)\Microsoft Visual Studio 10.0\UC>sn -p d:\mykeys.snk d:\mypublickey.snk
Microsoft (R) .NET Framework Strong Name Utility Version 4.0.30319.1
Copyright (c) Microsoft Corporation. All rights reserved.
Public key written to d:\mypublickey.snk
C:\Program Files (x86)\Microsoft Visual Studio 10.0\UC>_
```

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
```



```

C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC>sn -tp d:\MyPublicKey.snk

Microsoft (R) .NET Framework Strong Name Utility Version 4.0.30319.1
Copyright (c) Microsoft Corporation. All rights reserved.

Public key is
00240000048000009400000006020000002400000525341310004000001000100a3a041cca750ca
966615ed63dd199a3bcaa15ce51c7d2f285f147b9753335badf7fec60b82ed9adcca71262e192
3e29af87c6240ecd3880b16e07ebbc2ad6bda787884cc3fe8eeb0ae66a35d9b717d3e998ca8641
9b91e1436d2eace95a87af65ced64b8b55ac372b614c848ed272290bc5fccdd14ce7f6224c883e
5508ceb9

Public key token is eb2d94f3b945e4f6
C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC>_

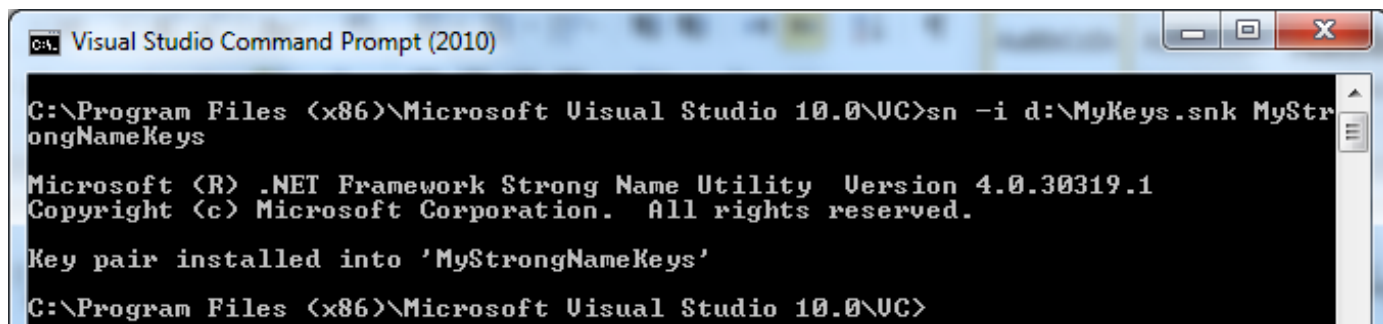
```

مقدار نمایش داده در انتهای تصویر فوق به‌عنوان «توکن کلید عمومی» (Public key Token) در واقع 8 بایت پایانی کد هش شده کریپتوگرافی محاسبه شده از کلید عمومی است. چون خود کلید عمومی همان‌طور که مشاهده می‌شود بسیار طولانی است، دات‌نت فریمورک معمولاً از این توکن 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 تغییر یافته 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 توکن کلید عمومی که در اختیار دارند استفاده کنند. در حال 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 مشتریان اسمبلی را از این تغییر آگاه ساخت و کلید عمومی مورد اطمینان را در اختیار آن‌ها قرار داد.

نکته : معمولاً 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.



```

C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC>sn -i d:\MyKeys.snk MyStrongNameKeys

Microsoft (R) .NET Framework Strong Name Utility Version 4.0.30319.1
Copyright (c) Microsoft Corporation. All rights reserved.

Key pair installed into 'MyStrongNameKeys'

C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC>

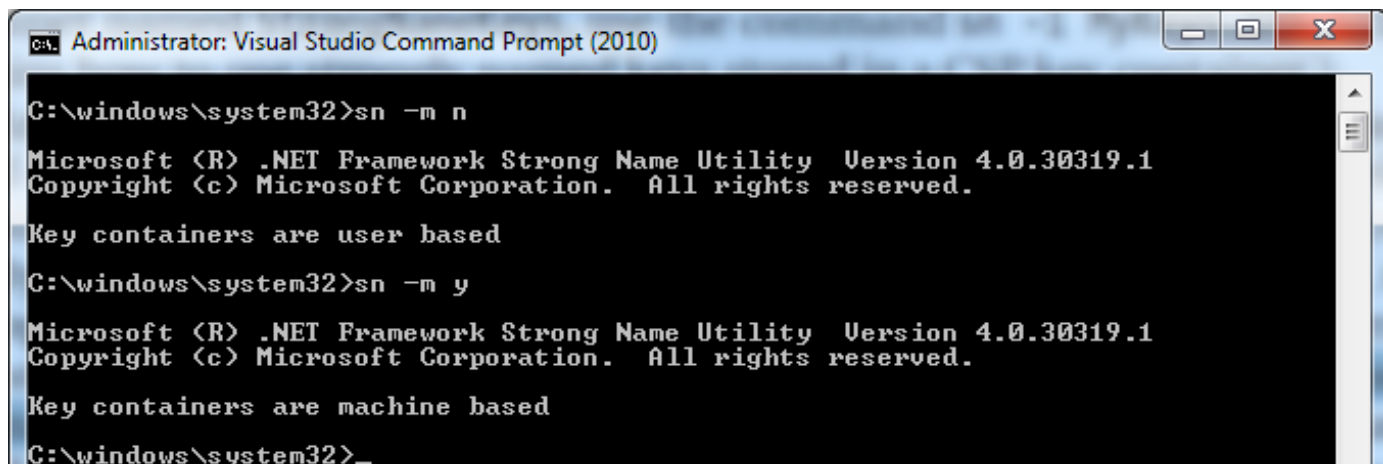
```

نکته مهمی که درباره مخازن کلید CSP باید بدان اشاره کرد این است که این مخازن شامل مخازن تعریف شده توسط «کاربر» و نیز مخازن «سیستمی» است. سیستم امنیتی ویندوز به کاربران اجازه دسترسی به مخازنی غیر از مخازن خودشان و مخازن سیستمی را نمی‌دهد. برنامه sn به صورت پیش فرض کلیدها را درون مخازن سیستمی ذخیره می‌کند. بنابراین هر کسی که بتواند به سیستم لاگین کند و نیز از نام مخزن مربوطه آگاه باشد، به راحتی 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 برگرداندن تنظیم به مخازن سیستمی نیز باید از دستور زیر استفاده کرد:

```
sn -m y
```



```

C:\windows\system32>sn -m n

Microsoft (R) .NET Framework Strong Name Utility Version 4.0.30319.1
Copyright (c) Microsoft Corporation. All rights reserved.

Key containers are user based

C:\windows\system32>sn -m y

Microsoft (R) .NET Framework Strong Name Utility Version 4.0.30319.1
Copyright (c) Microsoft Corporation. All rights reserved.

Key containers are machine based

C:\windows\system32>_

```

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

```
sn -d MyStrongNameKeys
```

```

Administrator: Visual Studio Command Prompt (2010)

C:\windows\system32>sn -d MyStrongNameKeys

Microsoft (R) .NET Framework Strong Name Utility Version 4.0.30319.1
Copyright (c) Microsoft Corporation. All rights reserved.

Container 'MyStrongNameKeys' deleted

C:\windows\system32>_

```

2. نام‌گذاری قوی یک اسمبلی

نام‌گذاری قوی یک اسمبلی به دلایل زیادی انجام می‌شود:

- برای اینکه اسمبلی شناسه‌ای منحصر به فرد 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 /keycontainer استفاده کنید.

```

Administrator: Visual Studio Command Prompt (2010)

C:\windows\system32>csc /?
Microsoft (R) Visual C# 2010 Compiler version 4.0.30319.1
Copyright (C) Microsoft Corporation. All rights reserved.

Visual C# 2010 Compiler Options

- OUTPUT FILES -
/out:<file>          Specify output file name (default: base name of
                    file with main class or first file)
/target:exe         Build a console executable (default) (Short form:
                    /t:exe)
/target:winexe      Build a Windows executable (Short form:
                    /t:winexe)
/target:library     Build a library (Short form: /t:library)
/target:module      Build a module that can be added to another
                    assembly (Short form: /t:module)
/delay:sign[+!-]   Delay-sign the assembly using only the public
                    portion of the strong name key
/doc:<file>         XML Documentation file to generate
/keyfile:<file>     Specify a strong name key file
/keycontainer:<string> Specify a strong name key container
/platform:<string>  Limit which platforms this code can run on: x86,
                    Itanium, x64, or anycpu. The default is anycpu.

- INPUT FILES -

```

"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.

☒ Sign the assembly

Choose a strong name key file:

<New...>

<Browse...>

When delay signed, the project will not run or be debuggable.

Reference Paths

Signing*

Security

Publish

Code Analysis

Timestamp server URL:

☒ Sign the assembly

Choose a strong name key file:

☐ Delay sign only

When delay signed, the project will not run or be debuggable.

روش ساده دیگر استفاده از attribute های سطح اسمبلی است:

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

3. بررسی اینکه آیا یک اسمبلی قوی-نام گذاری شده تغییر یافته یا خیر

زمانی که CLR در زمان اجرا یک اسمبلی قوی-نام گذاری شده را بارگذاری می کند:

-ابتدا با استفاده از کلید عمومی (که در خود اسمبلی ذخیره شده است) هش کد اینکریپت شده که در زمان کامپایل محاسبه شده (یا همان امضای دیجیتال که این نیز درون خود اسمبلی ذخیره شده است) را دیکریپت می کند. (هش کد زمان کامپایل)

-پس از آن هش کد اسمبلی را با استفاده از داده های مانیفست اسمبلی محاسبه می کند. (هش کد زمان اجرا)

-سپس این دو مقدار بدست آمده (هش کد زمان کامپایل و هش کد زمان اجرا) را با یکدیگر مقایسه می کند. این عملیات مقایسه و تایید مشخص می کند 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 می توان یک اسمبلی قوی-نام گذاری شده را تایید کرد. برای مثال برای تایید اسمبلی MyAsm.exe می توان از دستور زیر استفاده کرد:

```
sn -vf MyAsm.exe
```



```

Administrator: Visual Studio Command Prompt (2010)

C:\windows\system32>sn -vf "C:\Users\Saleh\Documents\Visual Studio 2010\Projects\
\ConsoleApplication1\ClassLibrary1\Class1.exe"

Microsoft (R) .NET Framework Strong Name Utility Version 4.0.30319.1
Copyright (c) Microsoft Corporation. All rights reserved.

Assembly 'C:\Users\Saleh\Documents\Visual Studio 2010\Projects\ConsoleApplication1\ClassLibrary1\Class1.exe' is valid

C:\windows\system32>_

```

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 قبلاً برای اسمبلی غیرفعال شده باشد. (با استفاده از سوییچ Vn مثل دستور sn -Vn 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:

```

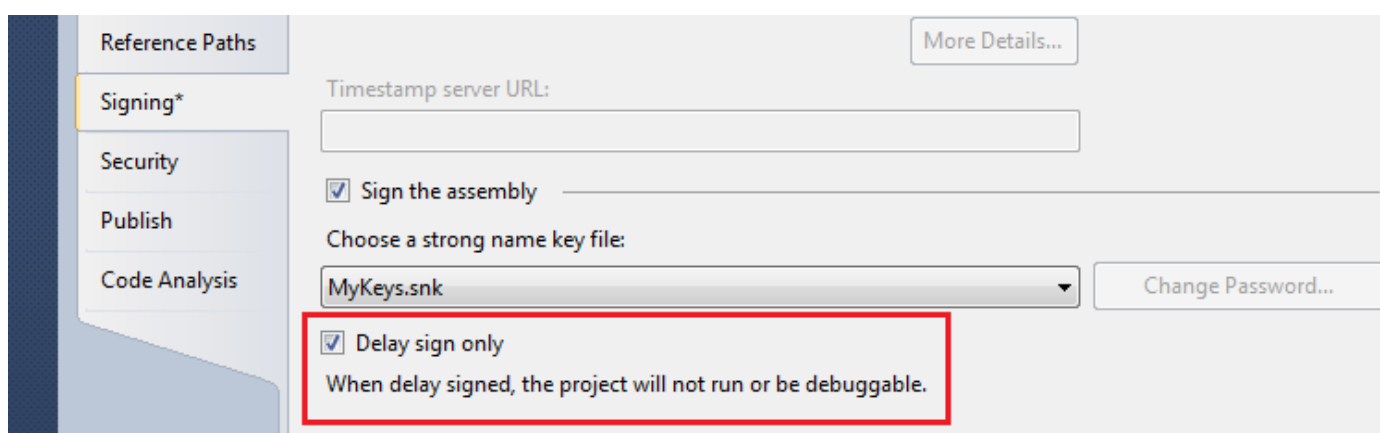
Microsoft (R) .NET Framework Strong Name Utility Version 2.0.50727.42
Copyright (C) Microsoft Corporation. All rights reserved.
Failed to verify assembly --
Strong name validation failed for assembly MyAsm.exe'.

```

4. امضای تأخیری (Delay Sign) یک اسمبلی

در صورتی که بخواهیم یک اسمبلی را امضا کنیم اما نخواهیم تمام اعضای تیم توسعه به کلید خصوصی مربوطه دسترسی داشته باشند باید از تکنیک امضای با تأخیر اسمبلی استفاده کنیم. ابتدا باید کلید عمومی تولید شده 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 سوییچ /delaysign باید کامپایل کنید. سپس با استفاده از سوییچ Vn 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 توکن 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 مرتباً 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، دات‌نت فریمورک مکانیزمی به نام امضای تأخیری (delay-signing) فراهم کرده است، که به شما اجازه می‌دهد تا یک اسمبلی را به‌صورت ناکامل (ناقص) امضا کنید. اسمبلی «ناقص-نام‌گذاری قوی شده»! حاوی

کلید عمومی و توکن کلید عمومی است که برای ریفرنس دادن اسمبلی نیاز است، اما تنها حاوی مکان خالی امضای دیجیتالی است که توسط کلید خصوصی تولید می‌شود. پس از کامل شدن توسعه برنامه، فرد مسئول امضای اسمبلی‌ها (signing authority - شخصی که مسئول امنیت و حفظ جفت‌کلیدهاست) اسمبلی‌های حاوی امضای تأخیری را دوباره امضا می‌کند، تا نام‌گذاری قوی آن اسمبلی کامل شود. برای امضای تأخیری 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 قبلاً 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 دستور کامپایل زیر می‌توان اسمبلی را امضای تأخیری کرد:

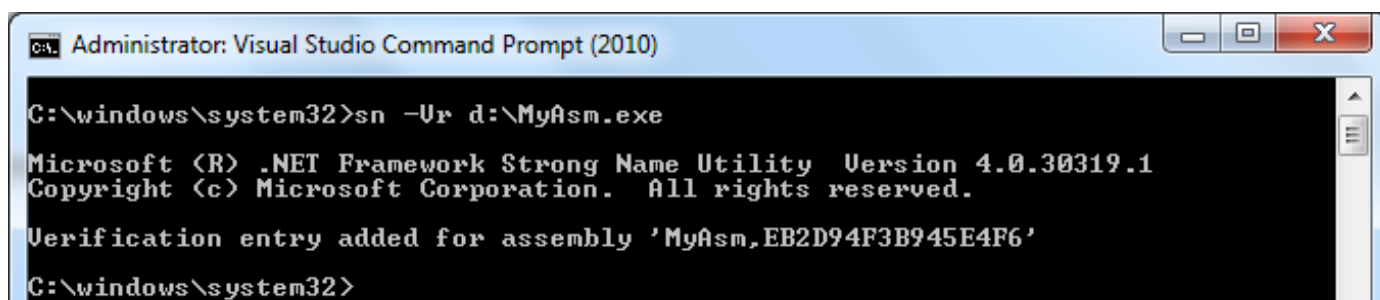
```
csc.exe /delaysign /keyfile:d:\MyPublicKey.snk /out:d:\MyAsm.exe d:\Class1.cs
```

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 اسمبلی‌ها به‌کار برد به‌صورت زیر است:

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

از آنجاکه در هنگام بارگذاری اسمبلی، CLR اسمبلی را به عنوان یک اسمبلی قوی نام‌گذاری شده در نظر می‌گیرد، همان‌طور که قبلاً اشاره شده، سعی می‌کند تا صحت آن را بررسی و تأیید کند. اما چون اسمبلی با امضای تأخیری هنوز امضا نشده است، باید CLR را جوری تنظیم کنید تا تأیید 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
```

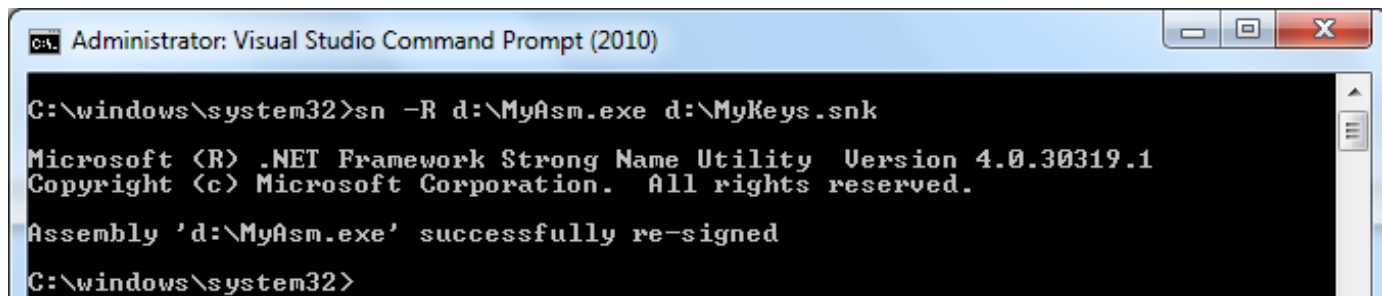


از لحاظ فنی این دستور اسمبلی موردنظر را در لیست «صرف‌نظر از تأیید اسمبلی» ثبت (register) می‌کند. دقت کنید که دستور فوق را باید در تمام سیستم‌هایی که قرار است به نحوی با این اسمبلی سروکار داشته باشند اجرا کنید!

نکته : تا زمانی که با استفاده از دستور فوق عملیات تأیید اعتبار اسمبلی‌های امضای تأخیری 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 این کار 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
```



```
Administrator: Visual Studio Command Prompt (2010)

C:\windows\system32>sn -R d:\MyAsm.exe d:\MyKeys.snk

Microsoft (R) .NET Framework Strong Name Utility Version 4.0.30319.1
Copyright (c) Microsoft Corporation. All rights reserved.

Assembly 'd:\MyAsm.exe' successfully re-signed

C:\windows\system32>
```

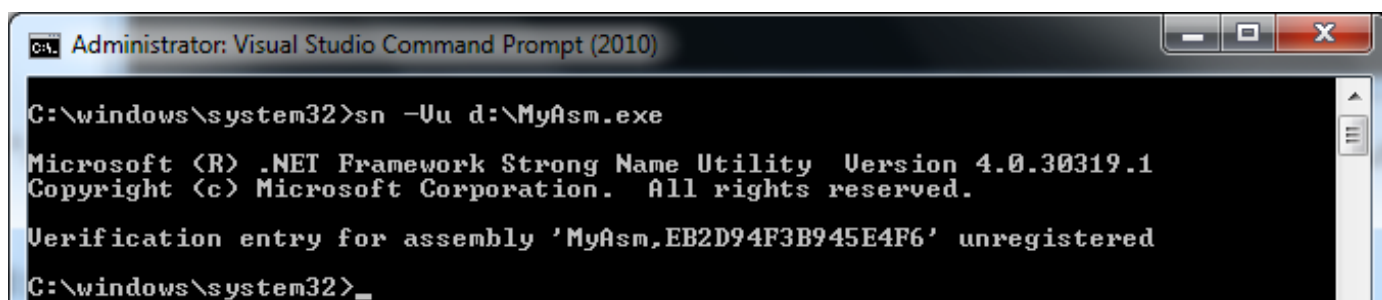
با استفاده از این دستور برنامه sn شروع به محاسبه هش کد زمان کامپایل می‌کند و در نهایت مقدار اینکریپت‌شده را درون اسمبلی ذخیره می‌کند.

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

```
sn -D assembly1 assembly2
```

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

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



```
Administrator: Visual Studio Command Prompt (2010)

C:\windows\system32>sn -Vu d:\MyAsm.exe

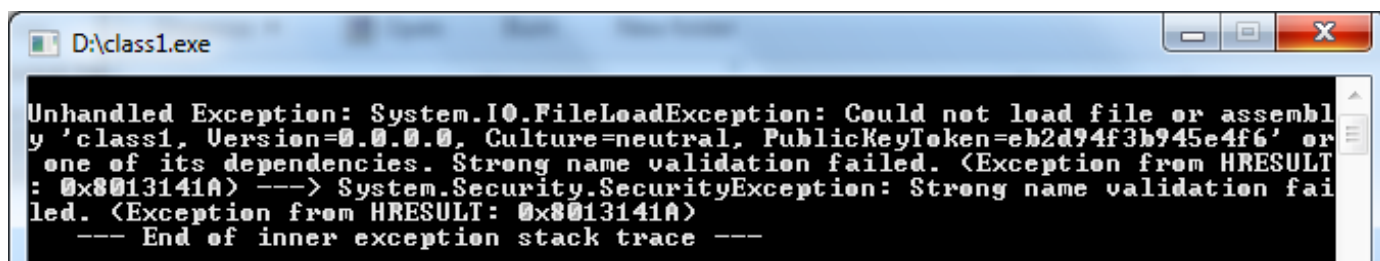
Microsoft (R) .NET Framework Strong Name Utility Version 4.0.30319.1
Copyright (c) Microsoft Corporation. All rights reserved.

Verification entry for assembly 'MyAsm.EB2D94F3B945E4F6' unregistered

C:\windows\system32>_
```

دستور فوق اسمبلی موردنظر را از لیست «صرفنظر از تایید اسمبلی» حذف (Unregister) می‌کند.

نکته : در صورتی که 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 سویچ `delaysign` است. برای استفاده از امکان امضای تأخیری اسمبلی می‌توان از attribute سطح اسمبلی `System.Reflection.AssemblyDelaySignAttribute` استفاده کرد. همچنین می‌شود از ابزار لینکر اسمبلی (`al.exe`) که از این سویچ 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، در صورتیکه برای اسمبلی‌های دارای نام قوی استفاده شوند موجب از کار 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 (مثلاً 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 تأخیری
- 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 تأخیری
- 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)

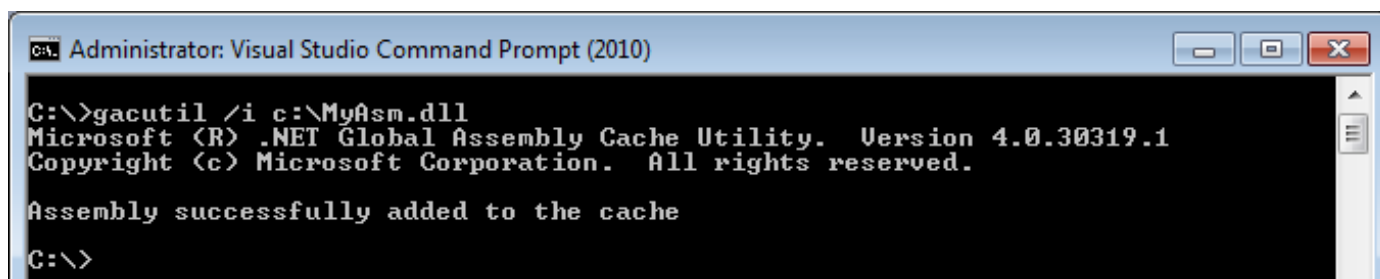
با استفاده از توضیحات این بخش می‌توان اسمبلی‌ها را به GAC اضافه و یا از درون آن حذف کرد. این کار با استفاده از برنامه `gacutil.exe` انجام می‌شود. مسیر نسخه 4 و 32 بیتی این برنامه به‌صورت زیر است:

```
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

داده شده 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 به صورت قوی نام‌گذاری کرد. برای افزودن یک اسمبلی با نام MyAsm.dll می‌توان از دستور زیر استفاده کرد:

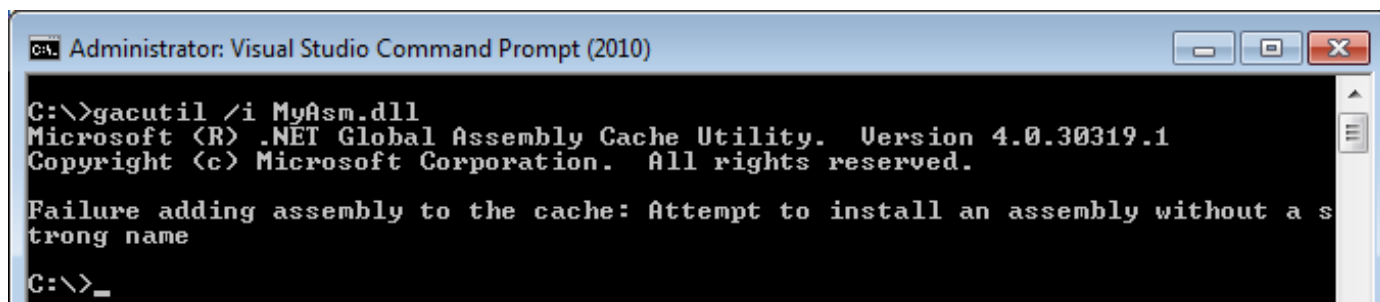
```
gacutil /i c:\MyAsm.dll
```



```
Administrator: Visual Studio Command Prompt (2010)
C:\>gacutil /i c:\MyAsm.dll
Microsoft (R) .NET Global Assembly Cache Utility. Version 4.0.30319.1
Copyright (c) Microsoft Corporation. All rights reserved.

Assembly successfully added to the cache
C:\>
```

در صورتی که اسمبلی مورد نظر دارای نام قوی نباشد، خطایی به صورت زیر نمایش داده خواهد شد:



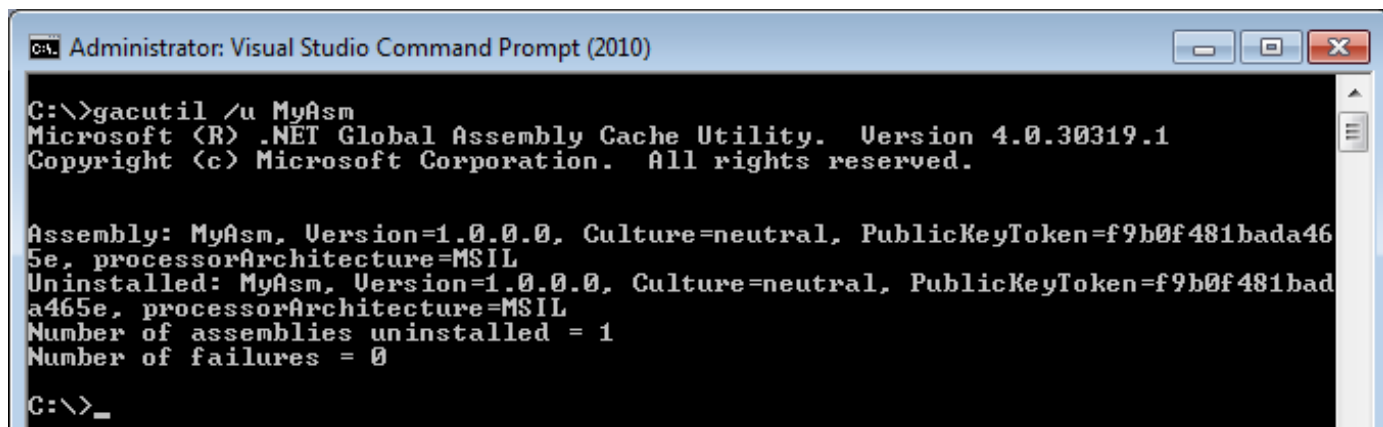
```
Administrator: Visual Studio Command Prompt (2010)
C:\>gacutil /i MyAsm.dll
Microsoft (R) .NET Global Assembly Cache Utility. Version 4.0.30319.1
Copyright (c) Microsoft Corporation. All rights reserved.

Failure adding assembly to the cache: Attempt to install an assembly without a strong name
C:\>_
```

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!



```

Administrator: Visual Studio Command Prompt (2010)

C:\>gacutil /u MyAsm
Microsoft (R) .NET Global Assembly Cache Utility. Version 4.0.30319.1
Copyright (c) Microsoft Corporation. All rights reserved.

Assembly: MyAsm, Version=1.0.0.0, Culture=neutral, PublicKeyToken=f9b0f481bada465e, processorArchitecture=MSIL
Uninstalled: MyAsm, Version=1.0.0.0, Culture=neutral, PublicKeyToken=f9b0f481bada465e, processorArchitecture=MSIL
Number of assemblies uninstalled = 1
Number of failures = 0

C:\>_

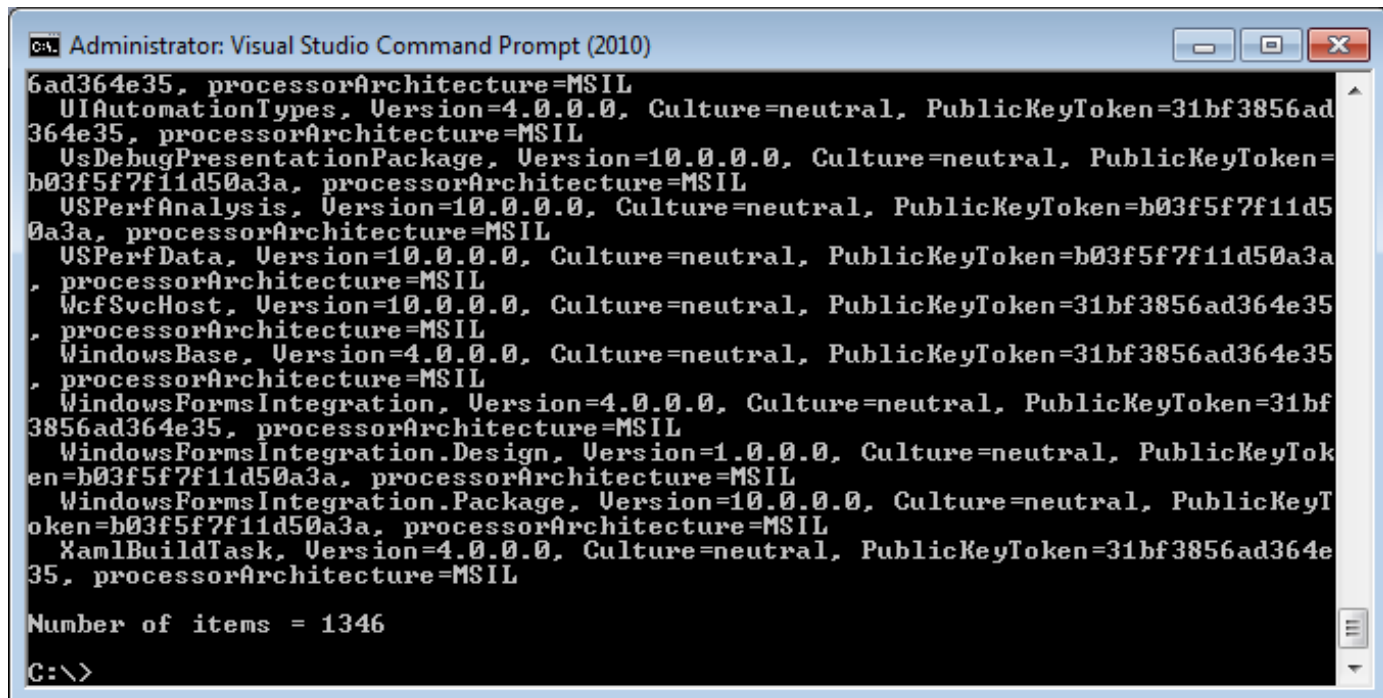
```

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
```



```

Administrator: Visual Studio Command Prompt (2010)

6ad364e35, processorArchitecture=MSIL
  UIAutomationTypes, Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35, processorArchitecture=MSIL
  UsDebugPresentationPackage, Version=10.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a, processorArchitecture=MSIL
  USPerfAnalysis, Version=10.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a, processorArchitecture=MSIL
  USPerfData, Version=10.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a, processorArchitecture=MSIL
  WcfSvcHost, Version=10.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35, processorArchitecture=MSIL
  WindowsBase, Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35, processorArchitecture=MSIL
  WindowsFormsIntegration, Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35, processorArchitecture=MSIL
  WindowsFormsIntegration.Design, Version=1.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a, processorArchitecture=MSIL
  WindowsFormsIntegration.Package, Version=10.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a, processorArchitecture=MSIL
  XamlBuildTask, Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35, processorArchitecture=MSIL

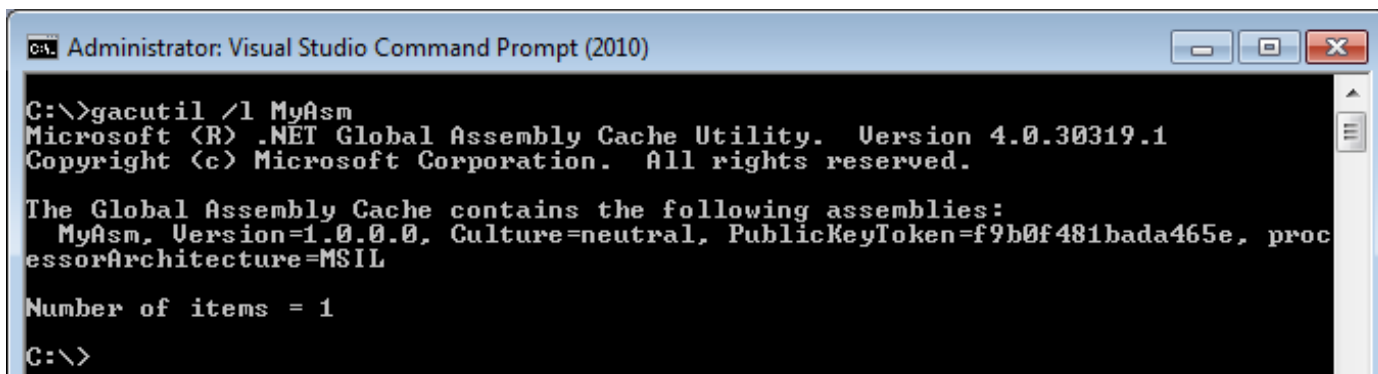
Number of items = 1346

C:\>

```

همان‌طور که مشاهده می‌کنید دستور فوق فهرستی بسیار طولانی از تمام اسمبلی‌های نصب شده در GAC را به همراه 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
```



```

Administrator: Visual Studio Command Prompt (2010)

C:\>gacutil /l MyAsm
Microsoft (R) .NET Global Assembly Cache Utility. Version 4.0.30319.1
Copyright (c) Microsoft Corporation. All rights reserved.

The Global Assembly Cache contains the following assemblies:
  MyAsm, Version=1.0.0.0, Culture=neutral, PublicKeyToken=f9b0f481bada465e, processorArchitecture=MSIL

Number of items = 1

C:\>

```

نکته : داتانت از GAC تنها در زمان اجرا استفاده می‌کند. بنابراین کامپایلر C# به صورت 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 به صورت 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 به کار 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 در صورتی که 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: ۲۲:۱ ۱۳۹۱/۰۸/۲۹

فقط می‌تونم بگم عالیه

ممنون

نویسنده: یوسف نژاد

تاریخ: ۱۶:۳۹ ۱۳۹۲/۰۲/۰۹

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

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

پارامتر T حتما باید با حرف بزرگ وارد شود.

نحوه ایجاد یک External Tools در VS2012 جهت تهیه Public Key Token

عنوان:

نویسنده: محمد باقر سیف اللهی

تاریخ:

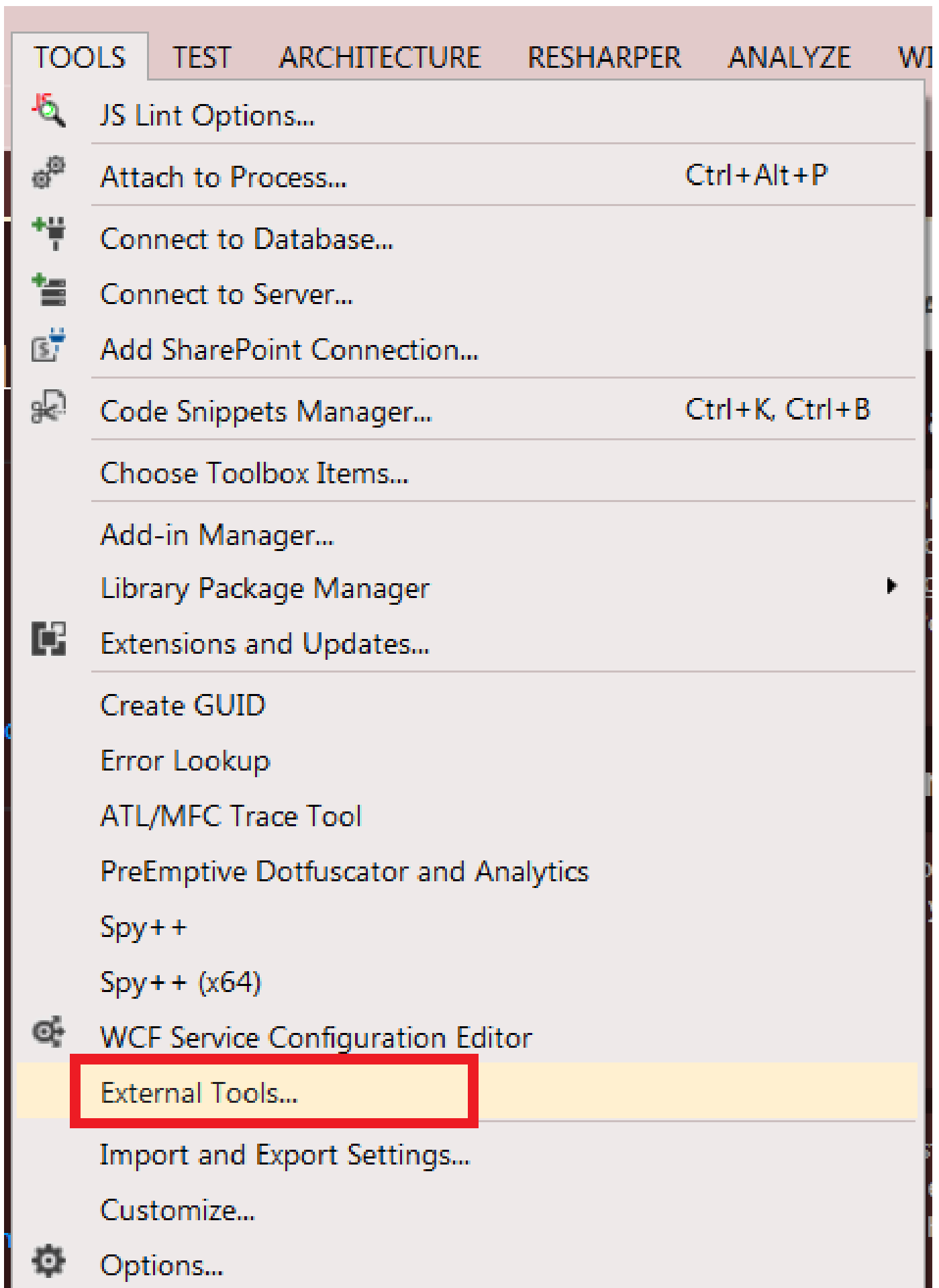
۱۱:۳۵ ۱۳۹۱/۱۰/۱۴

آدرس:

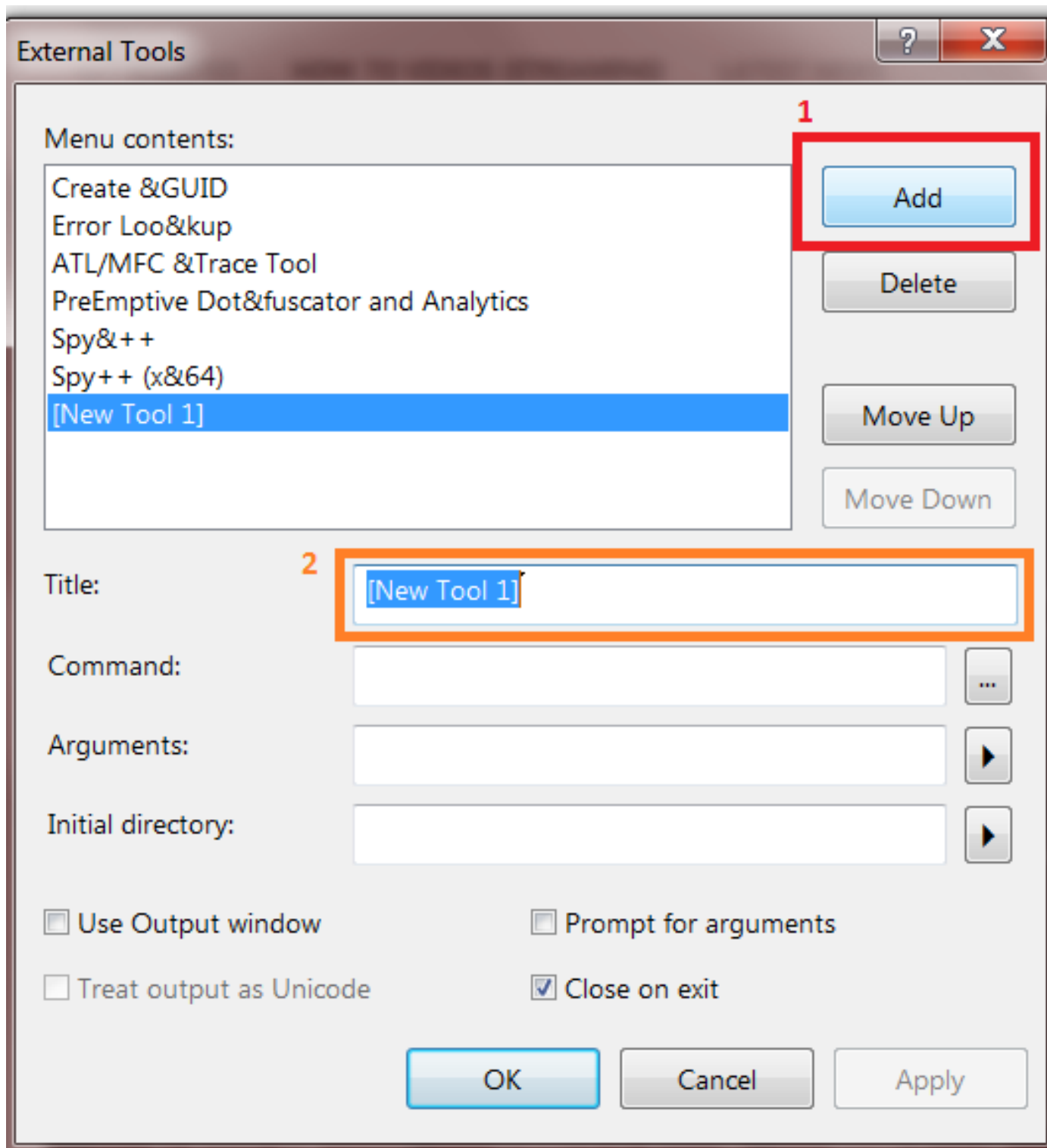
www.dotnettips.info

برچسب‌ها: Tools, VisualStudio.NET, .NET, Visual Studio, Strong Name

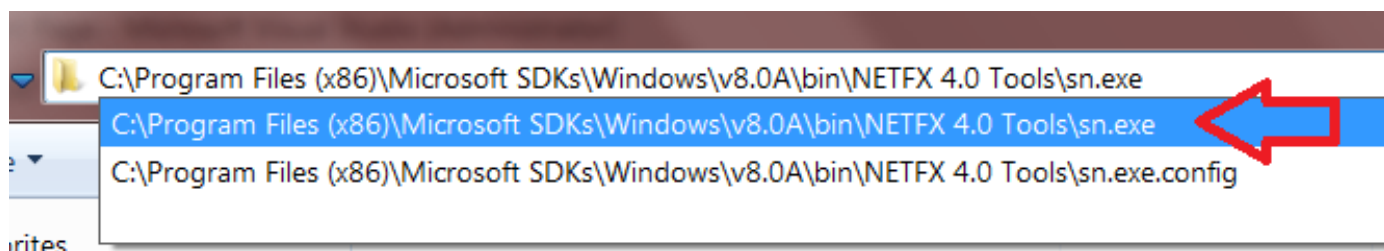
ایجاد Strong Name به اسمبلی برای داشتن یک هویت منحصر به فرد برای آن اسمبلی کمک می‌کند و یکی از پارامترهای آن داشتن Public Key Token برای اسمبلی است ([بیشتر](#)). در این پست قصد دارم به کمک ابزارهای جانبی Visual Studio 2012 که البته در 2010 نیز امکان پذیر است روشی برای تهیه آسان‌تر این Key ارائه کنم .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



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



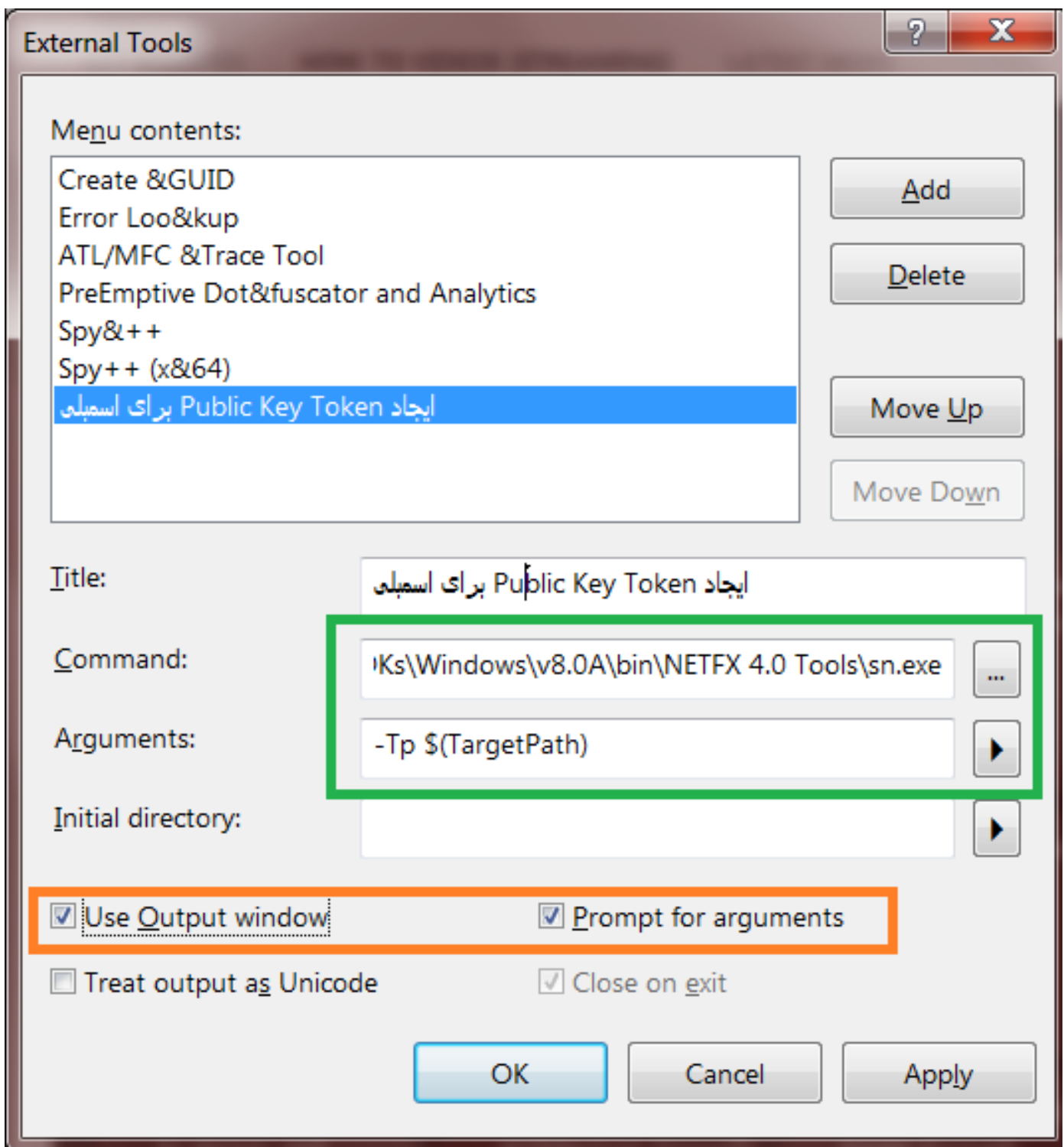
سپس مسیر فایل sn.exe را کپی کرده و در فیلد Command قرار دهید .



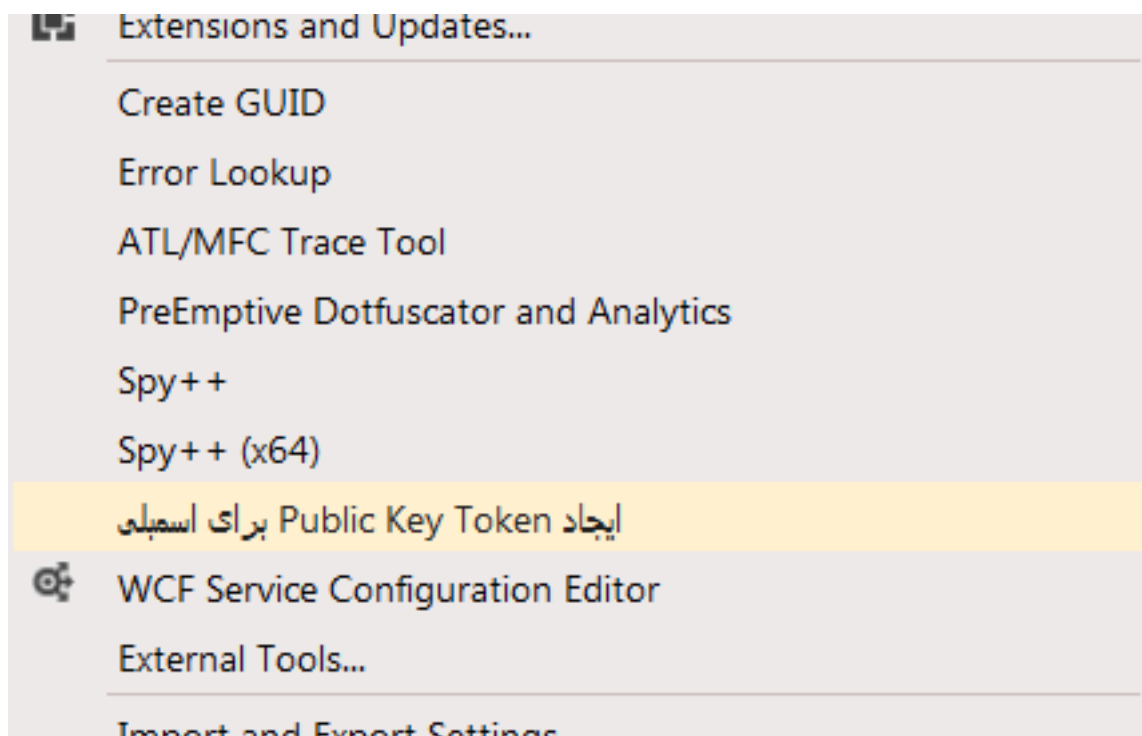
برای پارامتر از عبارت زیراستفاده کرده تا Public Key اسمبلی جاری را به شما بدهد . برای اطلاعات بیشتر در مورد آرگومانها به [اینجا](#) مراجعه کنید

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

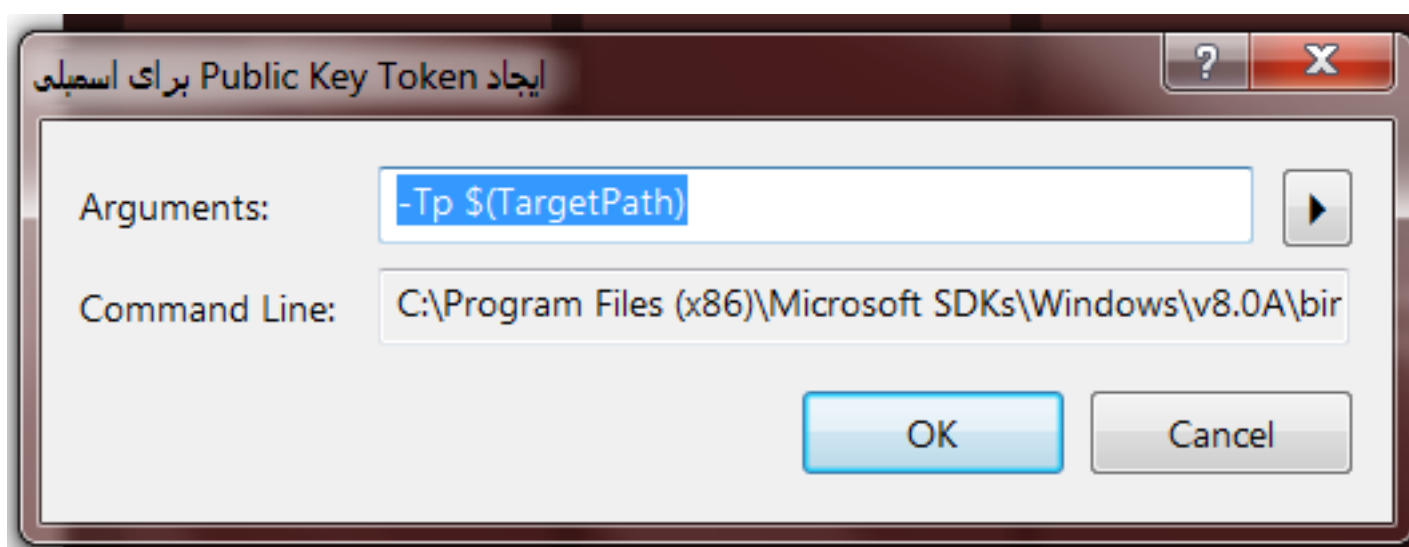
همچنین گزینه‌های Prompt for Arguments را برای دریافت آرگومان دلخواه شما (مثلا مواردی که مایلید برای یک اسمبلی دیگر key استخراج کنید) و Use Output window برای نمایش خروجی را علامت بزنید



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):
00240000048000000940000000060200000024000005253413100040000010001002dad9517360f66
c7e3968137a71d1c0fcfeb7264034b0ccb57528b66e557050568c78cce59087bfe25f4a012209a
d85c69657e823170306c46c2f99e2678e16f131dc865072165df730d2380b87a62e72dd3b2dde9
18c173785ebf08782cd457cf867822c5bb607bdf7c708ed6c1aab317262b951d54ea02167fc162
e4708aa8

Public key token is 25ce05e4457e7848
```

[موفق باشید](#)

اگر بازار هدف یک محصول شامل چندین کشور، منطقه یا زبان مختلف باشد، طراحی و پیاده سازی آن برای پشتیبانی از ویژگی‌های چندزبانه یک فاکتور مهم به حساب می‌آید. یکی از بهترین روش‌های پیاده سازی این ویژگی در دات نت استفاده از فایل‌های Resource است. درواقع هدف اصلی استفاده از فایل‌های Resource نیز Globalization است. Globalization برابر است با Internationalization + Localization که به اختصار به آن g11n می‌گویند. در تعریف، Internationalization (یا به اختصار i18n) به فرایند طراحی یک محصول برای پشتیبانی از فرهنگ(culture)ها و زبانهای مختلف و Localization (یا L10n) یا بومی‌سازی به شخصی‌سازی یک برنامه برای یک فرهنگ یا زبان خاص گفته میشود. (اطلاعات بیشتر در [اینجا](#)).

استفاده از این فایل‌ها محدود به پیاده سازی ویژگی چندزبانه نیست. شما میتوانید از این فایل‌ها برای نگهداری تمام رشته‌های موردنیاز خود استفاده کنید. نکته دیگری که باید بدان اشاره کرد این است که تقریباً تمامی منابع مورد استفاده در یک محصول را میتوان درون این فایل‌ها ذخیره کرد. این منابع در حالت کلی شامل موارد زیر است:

- انواع رشته‌های مورد استفاده در برنامه چون لیبل‌ها و پیغام‌ها و یا مسیرها (مثلاً نشانی تصاویر یا نام کنترلرها و اکشنها) و یا حتی برخی تنظیمات ویژه برنامه (که نمیخواهیم براحتی قابل نمایش یا تغییر باشد و یا اینکه بخواهیم با تغییر زبان تغییر کنند مثل direction و امثال آن)
- تصاویر و آیکونها و یا فایل‌های صوتی و انواع دیگر فایل‌ها
- و ...

نحوه بهره برداری از فایل‌های Resource در دات نت، پیاده سازی نسبتاً آسانی را در اختیار برنامه نویس قرار میدهد. برای استفاده از این فایل‌ها نیز روش‌های متنوعی وجود دارد که در مطلب جاری به چگونگی استفاده از آنها در پروژه‌های ASP.NET MVC پرداخته میشود.

Globalization در دات نت

فرمت نام یک culture دات نت (که در کلاس [CultureInfo](#) پیاده شده است) بر اساس استاندارد RFC 4646 ([^](#) و [^](#)) است. (در [اینجا](#) اطلاعاتی راجع به RFC یا Request for Comments آورده شده است). در این استاندارد نام یک فرهنگ (کالچر) ترکیبی از نام زبان به همراه نام کشور یا منطقه مربوطه است. نام زبان برپایه استاندارد ISO 639 که یک عبارت دوحرفی با حروف کوچک برای معرفی زبان است مثل fa برای فارسی و en برای انگلیسی و نام کشور یا منطقه نیز برپایه استاندارد ISO 3166 که به عبارت دوحرفی با حروف بزرگ برای معرفی یک کشور یا یک منطقه است مثل IR برای ایران یا US برای آمریکا است. برای نمونه میتوان به fa-IR برای زبان فارسی کشور ایران و یا en-US برای زبان انگلیسی آمریکایی اشاره کرد. البته در این روش نامگذاری یکی دو مورد استثنا هم وجود دارد (اطلاعات کامل کلیه زبانها: [National Language Support \(NLS\) API Reference](#)). یک فرهنگ خنثی (Neutral Culture) نیز تنها با استفاده از دو حرف نام زبان و بدون نام کشور یا منطقه معرفی میشود. مثل fa برای فارسی یا de برای آلمانی. در این بخش نیز دو استثنا وجود دارد ([^](#)).

در دات نت دو نوع culture وجود دارد: **Culture** و **UICulture**. هر دوی این مقادیر در هر Thread مقداری منحصر به فرد دارند. مقدار Culture بر روی توابع وابسته به فرهنگ (مثل فرمت رشته‌های تاریخ و اعداد و پول) تاثیر میگذارد. اما مقدار UICulture تعیین میکند که سیستم مدیریت منابع دات نت (Resource Manager) از کدام فایل Resource برای بارگذاری داده‌ها استفاده کند. درواقع در دات نت با استفاده از پراپرتی‌های موجود در کلاس استاتیک Thread برای ثرد جاری (که عبارتند از CurrentCulture و CurrentUICulture) برای فرمت کردن و یا انتخاب Resource مناسب تصمیم گیری میشود. برای تعیین کالچر جاری به صورت دستی میتوان بصورت زیر عمل کرد:

```
Thread.CurrentThread.CurrentUICulture = new CultureInfo("fa-IR");
Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("fa-IR");
```

در اینجا باید اشاره کنم که کار انتخاب Resource مناسب با توجه به کالچر ثرد جاری توسط ResourceProviderFactory پیشفرض دات نت انجام میشود. در مطالب بعدی به نحوه تعریف یک پرووایدر شخصی سازی شده هم خواهیم پرداخت.

پشتیبانی از زبانهای مختلف در MVC

برای استفاده از ویژگی چندزبانه در MVC دو روش کلی وجود دارد.

1. استفاده از فایل‌های Resource برای تمامی رشته‌های موجود

2. استفاده از View‌های مختلف برای هر زبان

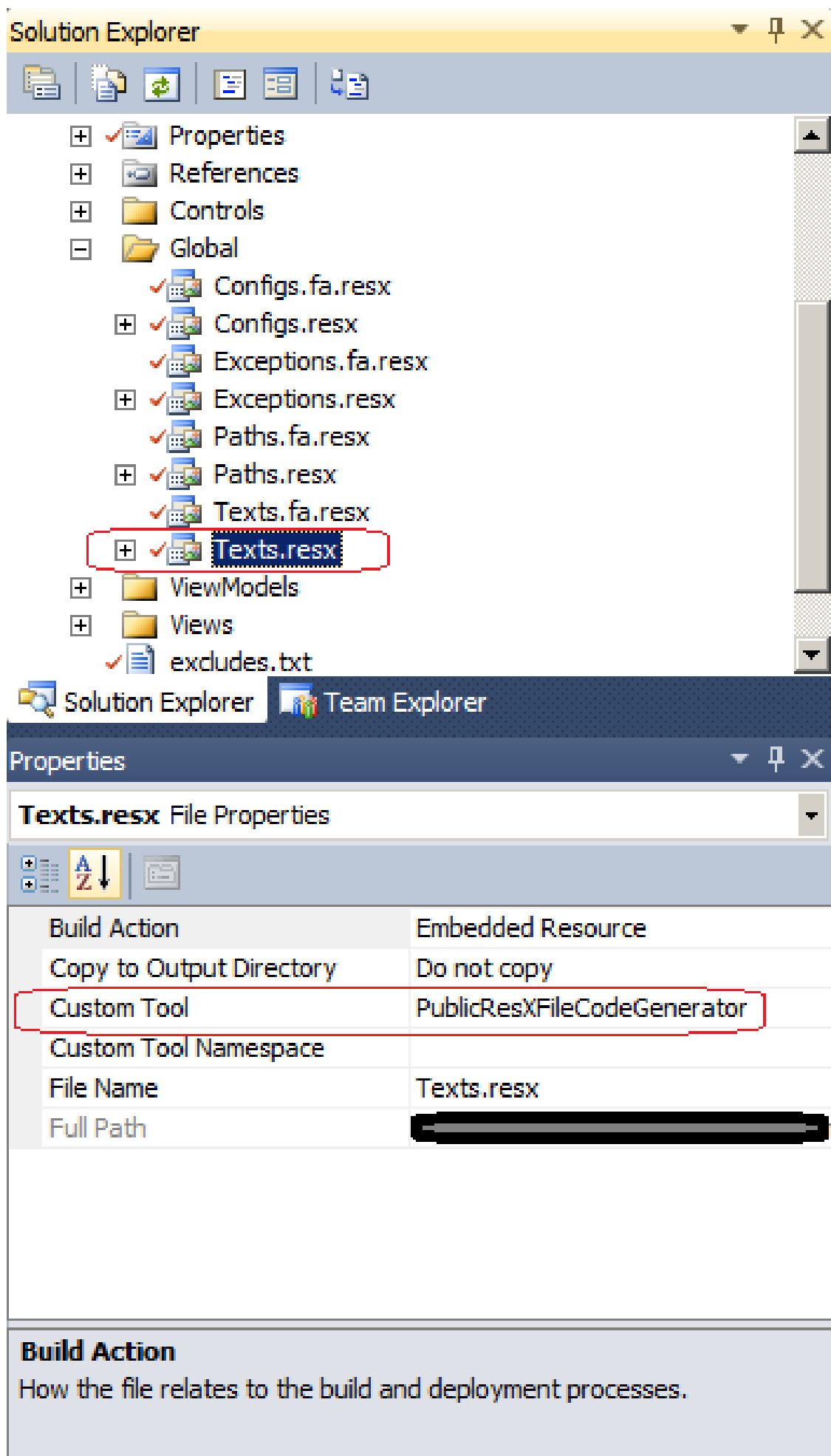
البته روش سومی هم که از ترکیب این دو روش استفاده میکند نیز وجود دارد. انتخاب روش مناسب کمی به سلیقه‌ها و عادات برنامه نویسی بستگی دارد. اگر فکر میکنید که استفاده از ویوهای مختلف به دلیل جداسازی مفاهیم درگیر در کالچرها (مثل جانمایی اجزای مختلف ویوها یا بحث Direction) باعث مدیریت بهتر و کاهش هزینه‌های پشتیبانی میشود بهتر است از روش دوم یا ترکیبی از این دو روش استفاده کنید. خودم به شخصه سعی میکنم از روش اول استفاده کنم. چون معتقدم استفاده از ویوهای مختلف باعث افزایش بیش از اندازه حجم کار میشود. اما در برخی موارد استفاده از روش دوم یا ترکیبی از دو روش میتواند بهتر باشد.

تولید فایل‌های Resource

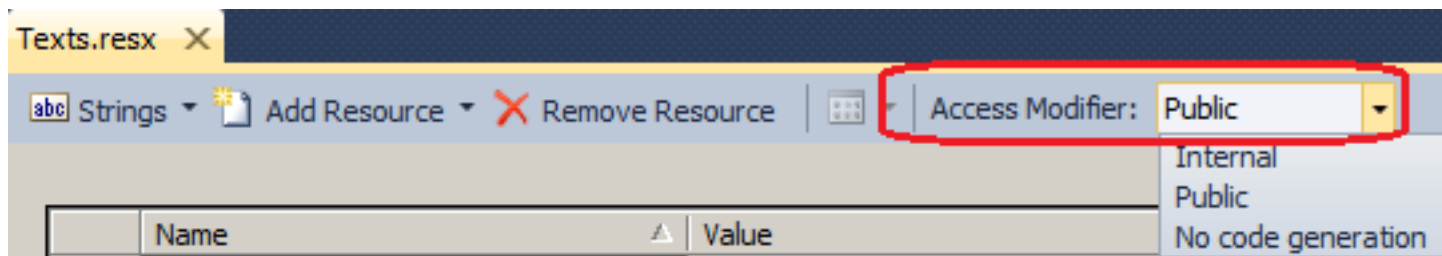
بهترین مکان برای نگهداری فایل‌های Resource در یک پروژه جداگانه است. در پروژه‌های از نوع وبسایت پوشه‌هایی با نام App_GlobalResources یا App_LocalResources وجود دارد که میتوان از آنها برای نگهداری و مدیریت این نوع فایل‌ها استفاده کرد. اما همانطور که در [اینجا](#) توضیح داده شده است این روش مناسب نیست. بنابراین ابتدا یک پروژه مخصوص نگهداری فایل‌های Resource ایجاد کنید و سپس اقدام به تهیه این فایل‌ها نمایید. سعی کنید که عنوان این پروژه به صورت زیر باشد. برای کسب اطلاعات بیشتر درباره نحوه نامگذاری اشیای مختلف در دات نت به [این مطلب](#) رجوع کنید.

SolutionName>.Resources>

برای افزودن فایل‌های Resource به این پروژه ابتدا برای انتخاب زبان پیش فرض محصول خود تصمیم بگیرید. پیشنهاد میکنم که از زبان انگلیسی (en-US) برای اینکار استفاده کنید. ابتدا یک فایل Resource (با پسوند .resx) مثلا با نام Texts.resx به این پروژه اضافه کنید. با افزودن این فایل به پروژه، ویژوال استودیو به صورت خودکار یک فایل cs حاوی کلاس متناظر با این فایل را به پروژه اضافه میکند. این کار توسط ابزار توکاری به نام ResXFileCodeGenerator انجام میشود. اگر به پراپرتی‌های این فایل .resx رجوع کنید میتوانید این عنوان را در پراپرتی Custom Tool ببینید. البته ابزار دیگری برای تولید این کلاسها نیز وجود دارد. این ابزارهای توکار برای سطوح دسترسی مختلف استفاده میشوند. ابزار پیش فرض در ویژوال استودیو یعنی همان ResXFileCodeGenerator، این کلاسها را با دسترسی internal تولید میکند که مناسب کار ما نیست. ابزار دیگری که برای اینکار درون ویژوال استودیو وجود دارد PublicResXFileCodeGenerator است و همانطور که از نامش پیداست از سطح دسترسی public استفاده میکند. برای تغییر این ابزار کافی است تا عنوان آن را دقیقا در پراپرتی Custom Tool تایپ کنید.



نکته: درباره پراپرتی مهم Build Action این فایلها در مطالب بعدی بیشتر بحث میشود. برای تعیین سطح دسترسی Resource موردنظر به روشی دیگر، میتوانید فایل Resource را باز کرده و Access Modifier آن را به Public تغییر دهید.



سپس برای پشتیبانی از زبانی دیگر، یک فایل دیگر Resource به پروژه اضافه کنید. نام این فایل باید همانم فایل اصلی به همراه نام کالچر موردنظر باشد. مثلاً برای زبان فارسی عنوان فایل باید Texts.fa-IR.resx یا به صورت ساده‌تر برای کالچر خنثی (بدون نام کشور) Texts.fa.resx باشد. دقت کنید اگر نام فایل را در همان پنجره افزودن فایل وارد کنید ویژوال استودیو این همانمی را به صورت هوشمند تشخیص داده و تغییراتی را در پراپرتی‌های پیش فرض فایل Resource ایجاد میکند.

نکته: این هوشمندی مرتبه نسبتاً بالایی دارد. بدین صورت که تنها در صورتیکه عبارت بعد از نام فایل اصلی Resource (رشته بعد از نقطه مثلاً fa در اینجا) متعلق به یک کالچر معتبر باشد این تغییرات اعمال خواهد شد.

مهمترین این تغییرات این است که ابزاری را برای پراپرتی Custom Tool این فایلها انتخاب نمیکند! اگر به پراپرتی فایل Texts.fa.resx مراجعه کنید این مورد کاملاً مشخص است. در نتیجه دیگر فایل cs حاوی کلاسی جداگانه برای این فایل ساخته نمیشود. همچنین اگر فایل Resource جدید را باز کنید میبینید که برای Access Modifier آن گزینه No Code Generation انتخاب شده است.

در ادامه شروع به افزودن عناوین موردنظر در این دو فایل کنید. در اولی (بدون نام زبان) رشته‌های مربوط به زبان انگلیسی و در دومی رشته‌های مربوط به زبان فارسی را وارد کنید. سپس در هرجایی که یک لیبل یا یک رشته برای نمایش وجود دارد از این کلیدهای Resource استفاده کنید مثل:

```
SolutionName>.Resources.Texts.Save>
SolutionName>.Resources.Texts.Cancel>
```

استفاده از Resource در ویومدل ها

دو خاصیت معروفی که در ویومدلها استفاده میشوند عبارتند از: DisplayName و Required. پشتیبانی از کلیدهای Resource به صورت توکار در خاصیت Required وجود دارد. برای استفاده از آنها باید به صورت زیر عمل کرد:

```
[Required(ErrorMessageResourceName = "ResourceKeyName", ErrorMessageResourceType =
typeof(<SolutionName>.Resources.<ResourceClassName>))]
```

در کد بالا باید از نام فایل Resource اصلی (فایل اول که بدون نام کالچر بوده و به عنوان منبع پیشفرض به همراه یک فایل cs حاوی کلاس مربوطه نیز هست) برای معرفی ErrorMessageResourceType استفاده کرد. چون ابزار توکار ویژوال استودیو از نام این فایل برای تولید کلاس مربوطه استفاده میکند.

متأسفانه خاصیت DisplayName که در فضای نام System.ComponentModel (در فایل System.dll) قرار دارد قابلیت استفاده از کلیدهای Resource را به صورت توکار ندارد. در دات نت 4 خاصیت دیگری در فضای نام System.ComponentModel.DataAnnotations به نام Display (در فایل System.ComponentModel.DataAnnotations.dll) وجود دارد که این امکان را به صورت توکار دارد. اما قابلیت استفاده از این خاصیت تنها در MVC 3 وجود دارد. برای نسخه‌های قدیمیتر

MVC امکان استفاده از این خاصیت حتی اگر نسخه فریمورک هدف 4 باشد وجود ندارد، چون هسته این نسخه‌های قدیمی امکان استفاده از ویژگی‌های جدید فریمورک با نسخه بالاتر را ندارد. برای رفع این مشکل میتوان کلاس خاصیت DisplayName را برای استفاده از خاصیت Display به صورت زیر توسعه داد:

```
public class LocalizationDisplayNameAttribute : DisplayNameAttribute
{
    private readonly DisplayAttribute _display;
    public LocalizationDisplayNameAttribute(string resourceName, Type resourceType)
    {
        _display = new DisplayAttribute { ResourceType = resourceType, Name = resourceName };
    }
    public override string DisplayName
    {
        get
        {
            try
            {
                return _display.GetName();
            }
            catch (Exception)
            {
                return _display.Name;
            }
        }
    }
}
```

در این کلاس با ترکیب دو خاصیت نامبرده امکان استفاده از کلیدهای Resource فراهم شده است. در پیاده سازی این کلاس فرض شده است که نسخه فریمورک هدف حداقل برابر 4 است. اگر از نسخه‌های پایین‌تر استفاده میکنید در پیاده سازی این کلاس باید کاملاً به صورت دستی کلید موردنظر را از Resource معرفی شده بدست آورید. مثلاً به صورت زیر:

```
public class LocalizationDisplayNameAttribute : DisplayNameAttribute
{
    private readonly PropertyInfo nameProperty;
    public LocalizationDisplayNameAttribute(string displayNameKey, Type resourceType = null)
        : base(displayNameKey)
    {
        if (resourceType != null)
            nameProperty = resourceType.GetProperty(base.DisplayName, BindingFlags.Static |
BindingFlags.Public);
    }
    public override string DisplayName
    {
        get
        {
            if (nameProperty == null) base.DisplayName;
            return (string)nameProperty.GetValue(nameProperty.DeclaringType, null);
        }
    }
}
```

برای استفاده از این خاصیت جدید میتوان به صورت زیر عمل کرد:

```
[LocalizationDisplayName("ResourceKeyName", typeof(<SolutionName>.Resources.<ResourceClassName>))]
```

البته بیشتر خواص متداول در ویومدلها از ویژگی موردبحث پشتیبانی میکنند.

نکته: به کار گیری این روش ممکن است در پروژه‌های بزرگ کمی گیج کننده و دردسرساز بوده و باعث پیچیدگی بی‌مورد کد و نیز افزایش بیش از حد حجم کدنویسی شود. در مقاله آقای فیل هک ([Model Metadata and Validation Localization using Conventions](#)) روش بهتر و تمیزتری برای مدیریت پیامهای این خاصیت‌ها آورده شده است.

پشتیبانی از ویژگی چند زبانه

مرحله بعدی برای چندزبانه کردن پروژه‌های MVC تغییراتی است که برای مدیریت Culture جاری برنامه باید پیاده شوند. برای

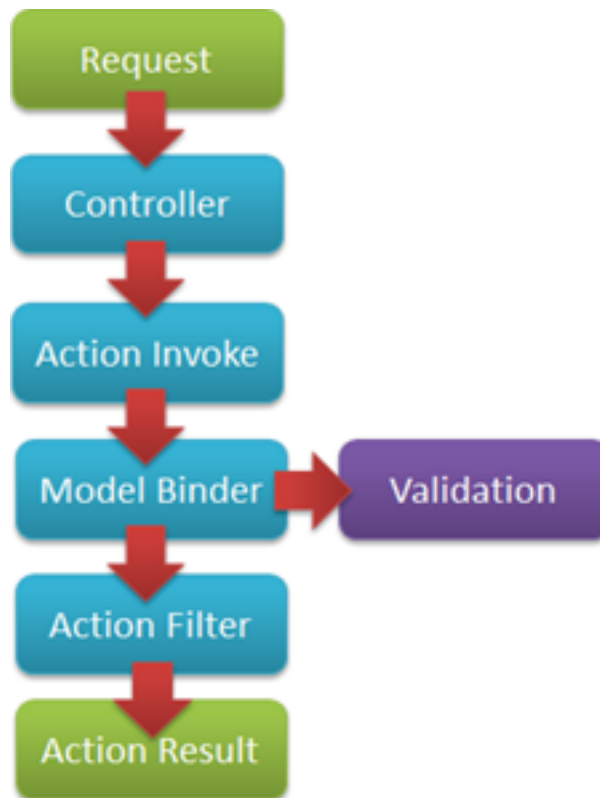
اینکار باید خاصیت `CurrentUICulture` در ثرد جاری کنترل و مدیریت شود. یکی از مکانهایی که برای نگهداری زبان جاری استفاده میشود کوکی است. معمولاً برای اینکار از کوکی‌های دارای تاریخ انقضای طولانی استفاده میشود. میتوان از تنظیمات موجود در فایل کانفیگ برای ذخیره زبان پیش فرض سیستم نیز استفاده کرد. روشی که معمولاً برای مدیریت زبان جاری میتوان از آن استفاده کرد پیاده سازی یک کلاس پایه برای تمام کنترلرها است. کد زیر راه حل نهایی را نشان میدهد:

```
public class BaseController : Controller
{
    private const string LanguageCookieName = "MyLanguageCookieName";
    protected override void ExecuteCore()
    {
        var cookie = HttpContext.Request.Cookies[LanguageCookieName];
        string lang;
        if (cookie != null)
        {
            lang = cookie.Value;
        }
        else
        {
            lang = ConfigurationManager.AppSettings["DefaultCulture"] ?? "fa-IR";
            var httpCookie = new HttpCookie(LanguageCookieName, lang) { Expires = DateTime.Now.AddYears(1) };
            HttpContext.Response.SetCookie(httpCookie);
        }
        Thread.CurrentThread.CurrentUICulture = CultureInfo.CreateSpecificCulture(lang);
        base.ExecuteCore();
    }
}
```

راه حل دیگر استفاده از یک `ActionFilter` است که نحوه پیاده سازی یک نمونه از آن در زیر آورده شده است:

```
public class LocalizationActionFilterAttribute : ActionFilterAttribute
{
    private const string LanguageCookieName = "MyLanguageCookieName";
    public override void OnActionExecuting(ActionExecutingContext filterContext)
    {
        var cookie = filterContext.HttpContext.Request.Cookies[LanguageCookieName];
        string lang;
        if (cookie != null)
        {
            lang = cookie.Value;
        }
        else
        {
            lang = ConfigurationManager.AppSettings["DefaultCulture"] ?? "fa-IR";
            var httpCookie = new HttpCookie(LanguageCookieName, lang) { Expires = DateTime.Now.AddYears(1) };
        }
        filterContext.HttpContext.Response.SetCookie(httpCookie);
        Thread.CurrentThread.CurrentUICulture = CultureInfo.CreateSpecificCulture(lang);
        base.OnActionExecuting(filterContext);
    }
}
```

نکته مهم: تعیین زبان جاری (یعنی همان مقداردهی پراپرتی `CurrentCulture` ثرد جاری) در یک اکشن فیلتر بدرستی عمل نمیکند. برای بررسی بیشتر این مسئله ابتدا به تصویر زیر که ترتیب رخ دادن رویدادهای مهم در ASP.NET MVC را نشان میدهد دقت کنید:



همانطور که در تصویر فوق مشاهده میکنید رویداد `OnActionExecuting` که در یک اکشن فیلتر به کار میرود بعد از عملیات مدل بایندینگ رخ میدهد. بنابراین قبل از تعیین کالچر جاری، عملیات `validation` و یافتن متن خطاها از فایل‌های `Resource` انجام میشود که منجر به انتخاب کلیدهای مربوط به کالچر پیشفرض سرور (و نه آنچه که کاربر تنظیم کرده) خواهد شد. بنابراین استفاده از یک اکشن فیلتر برای تعیین کالچر جاری مناسب نیست. راه حل مناسب استفاده از همان کنترلر پایه است، زیرا متد `ExecuteCore` قبل از تمامی این عملیات صدا زده میشود. بنابراین همیشه کالچر تنظیم شده توسط کاربر به عنوان مقدار جاری آن در ثرد ثبت میشود.

امکان تعیین/تغییر زبان توسط کاربر

برای تعیین یا تغییر زبان جاری سیستم نیز روشهای گوناگونی وجود دارد. استفاده از زبان تنظیم شده در مرورگر کاربر، استفاده از عنوان زبان در آدرس صفحات درخواستی و یا تعیین زبان توسط کاربر در تنظیمات برنامه/سایت و ذخیره آن در کوکی یا دیتابیس و مواردی از این دست روشهایی است که معمولاً برای تعیین زبان جاری از آن استفاده میشود. در کدهای نمونه ای که در بخشهای قبل آورده شده است فرض شده است که زبان جاری سیستم درون یک کوکی ذخیره میشود بنابراین برای استفاده از این روش میتوان از قطعه کدی مشابه زیر (مثلاً در فایل `_Layout.cshtml`) برای تعیین و تغییر زبان استفاده کرد:

```

<select id="langs" onchange="languageChanged()">
  <option value="fa-IR">فارسی</option>
  <option value="en-US">انگلیسی</option>
</select>
<script type="text/javascript">
  function languageChanged() {
    setCookie("MyLanguageCookieName", $('#langs').val(), 365);
    window.location.reload();
  }
  document.ready = function () {
    $('#langs').val(getCookie("MyLanguageCookieName"));
  };
  function setCookie(name, value, exdays, path) {
    var exdate = new Date();
    exdate.setDate(exdate.getDate() + exdays);
    var newValue = escape(value) + ((exdays == null) ? "" : "; expires=" + exdate.toUTCString()) +
    ((path == null) ? "" : "; path=" + path);
    document.cookie = name + "=" + newValue;
  }
</script>

```

```
function getCookie(name) {
    var i, x, y, cookies = document.cookie.split(";");
    for (i = 0; i < cookies.length; i++) {
        x = cookies[i].substr(0, cookies[i].indexOf("="));
        y = cookies[i].substr(cookies[i].indexOf("=") + 1);
        x = x.replace(/^\s+|\s+$/g, "");
        if (x == name) {
            return unescape(y);
        }
    }
}
</script>
```

متدهای `getCookie` و `setCookie` جاوا اسکریپتی در کد بالا از [اینجا](#) گرفته شده اند البته پس از کمی تغییر.

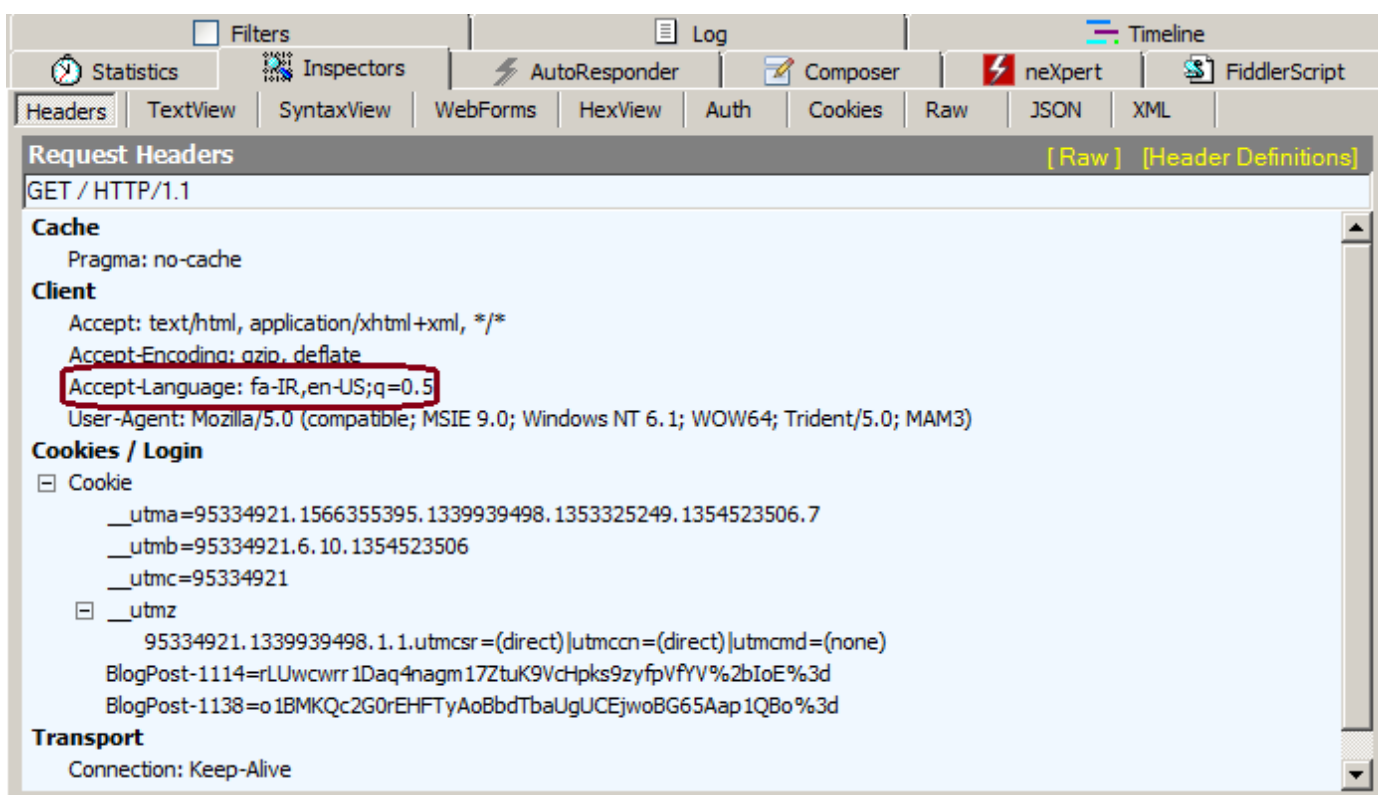
نکته : مطلب `Cookie` ها بحثی نسبتاً مفصل است که در جای خودش باید به صورت کامل آورده شود. اما در اینجا تنها به همین نکته اشاره کنم که عدم توجه به پراپرتی `path` کوکی‌ها در این مورد خاص برای خود من بسیار گیج‌کننده و دردسرساز بود.

به عنوان راهی دیگر میتوان به جای روش ساده استفاده از کوکی، تنظیماتی در اختیار کاربر قرار داد تا بتواند زبان تنظیم شده را درون یک فایل یا دیتابیس ذخیره کرد البته با در نظر گرفتن مسائل مربوط به کش کردن این تنظیمات.

راه حل بعدی میتواند استفاده از تنظیمات مرورگر کاربر برای دریافت زبان جاری تنظیم شده است. مرورگرها تنظیمات مربوط به زبان را در قسمت `Accept-Languages` در `HTTP Header` درخواست ارسالی به سمت سرور قرار میدهند. بصورت زیر:

```
GET http://www.dotnettips.info HTTP/1.1
...
Accept-Language: fa-IR,en-US;q=0.5
...
```

این هم تصویر مربوط به [Fiddler](#) آن:

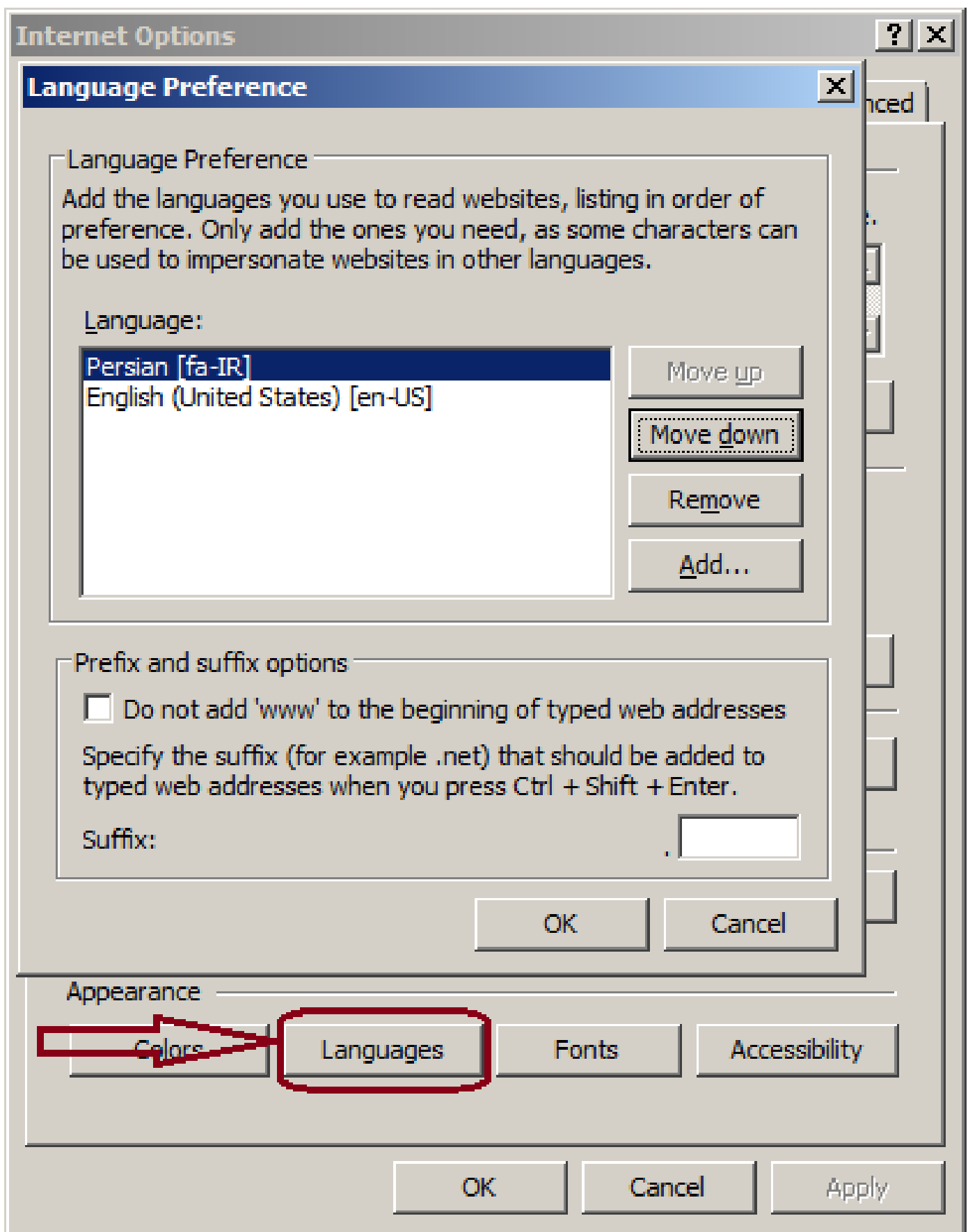


نکته: پارامتر `q` در عبارت مشخص شده در تصویر فوق `relative quality factor` نام دارد و به نوعی مشخص کننده اولویت زبان مربوطه است. مقدار آن بین 0 و 1 است و مقدار پیش فرض آن 1 است. هرچه مقدار این پارامتر بیشتر باشد زبان مربوطه اولویت

بالاتری دارد. مثلاً عبارت زیر را در نظر بگیرید:

```
Accept-Language: fa-IR, fa;q=0.8,en-US;q=0.5,ar-BH;q=0.3
```

در این حالت اولویت زبان fa-IR برابر 1 و fa برابر 0.8 (fa;q=0.8) است. اولویت دیگر زبانهای تنظیم شده نیز همانطور که نشان داده شده است در مراتب بعدی قرار دارند. در تنظیم نمایش داده شده برای تغییر این تنظیمات در IE میتوان همانند تصویر زیر اقدام کرد:



در تصویر بالا زبان فارسی اولویت بالاتری نسبت به انگلیسی دارد. برای اینکه سیستم g11n دات نت به صورت خودکار از این مقادیر جهت زبان ثرد جاری استفاده کند میتوان از تنظیم زیر در فایل کانفیگ استفاده کرد:

```
<system.web>
  <globalization enableClientBasedCulture="true" uiCulture="auto" culture="auto"></globalization>
</system.web>
```

در سمت سرور نیز برای دریافت این مقادیر تنظیم شده در مرورگر کاربر میتوان از کدهای زیر استفاده کرد. مثلاً در یک اکشن فیلتر:

```
var langs = filterContext.HttpContext.Request.UserLanguages;
```

پراپرتی UserLanguages از کلاس Request حاوی آرایه‌ای از استرینگ است. این آرایه درواقع از Split کردن مقدار Accept-Languages با کاراکتر ',' بدست می‌آید. بنابراین اعضای این آرایه رشته‌ای از نام زبان به همراه پارامتر q مربوطه خواهند بود (مثل "fa;q=0.8").

راه دیگر مدیریت زبانها استفاده از عنوان زبان در مسیر درخواستی صفحات است. مثلاً آدرسی شبیه به `www.MySite.com/fa/employees` نشان میدهد کاربر درخواست نسخه فارسی از صفحه Employees را دارد. نحوه استفاده از این عناوین و نیز موقعیت فیزیکی این عناوین در مسیر صفحات درخواستی کاملاً به سلیقه برنامه نویس و یا کارفرما بستگی دارد. روش کلی بهره برداری از این روش در تمام موارد تقریباً یکسان است.

برای پیاده سازی این روش ابتدا باید یک route جدید در فایل Global.asax.cs اضافه کرد:

```
routes.MapRoute(
    "Localization", // Route name
    "{lang}/{controller}/{action}/{id}", // URL with parameters
    new { controller = "Home", action = "Index", id = UrlParameter.Optional } // Parameter defaults
);
```

دقت کنید که این route باید قبل از تمام route‌های دیگر ثبت شود. سپس باید کلاس پایه کنترلر را به صورت زیر پیاده سازی کرد:

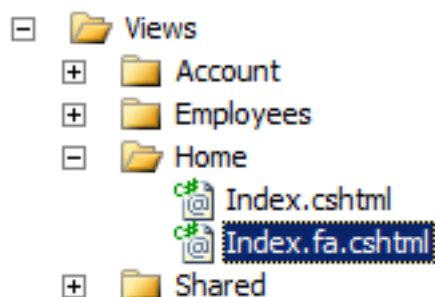
```
public class BaseController : Controller
{
    protected override void ExecuteCore()
    {
        var lang = RouteData.Values["lang"];
        if (lang != null && !string.IsNullOrEmpty(lang.ToString()))
        {
            Thread.CurrentThread.CurrentUICulture = CultureInfo.CreateSpecificCulture(lang.ToString());
        }
        base.ExecuteCore();
    }
}
```

این کار را در یک اکشن فیلتر هم میتوان انجام داد اما با توجه به توضیحاتی که در قسمت قبل داده شد استفاده از اکشن فیلتر برای تعیین زبان جاری کار مناسبی نیست.

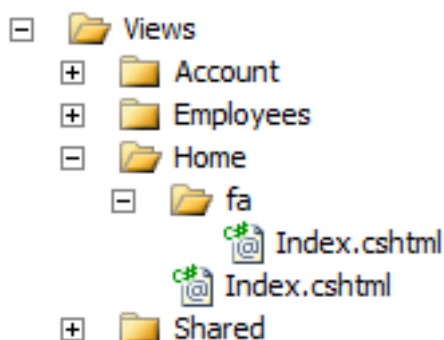
نکته: به دلیل آوردن عنوان زبان در مسیر درخواستها باید کنترلر دقیقتری بر کلیه مسیرهای موجود داشت!

استفاده از ویوهای جداگانه برای زبانهای مختلف

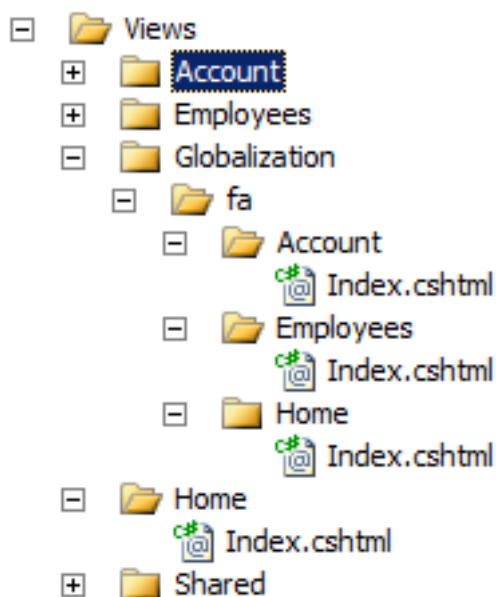
برای اینکار ابتدا ساختار مناسبی را برای نگهداری از ویوهای مختلف خود در نظر بگیرید. مثلاً میتوانید همانند نامگذاری فایل‌های Resource از نام زبان یا کالچر به عنوان بخشی از نام فایل‌های ویو استفاده کنید و تمام ویوها را در یک مسیر ذخیره کنید. همانند تصویر زیر:



البته اینکار ممکن است به مدیریت این فایلها را کمی مشکل کند چون به مرور زمان تعداد فایلهای ویو در یک فولدر زیاد خواهد شد. روش دیگری که برای نگهداری این ویوها میتوان به کار برد استفاده از فولدرهای جداگانه با عناوین زبانهای موردنظر است. مانند تصویر زیر:



روش دیگری که برای نگهداری و مدیریت بهتر ویوهای زبانهای مختلف از آن استفاده میشود به شکل زیر است:



استفاده از هرکدام از این روشها کاملاً به سلیقه و راحتی مدیریت فایلها برای برنامه نویس بستگی دارد. در هر صورت پس از

انتخاب یکی از این روشها باید اپلیکشن خود را طوری تنظیم کنیم که با توجه به زبان جاری سیستم، ویوی مربوطه را جهت نمایش انتخاب کند.

مثلا برای روش اول نامگذاری ویوها میتوان از روش دستکاری متد `OnActionExecuted` در کلاس پایه کنترلر استفاده کرد:

```
public class BaseController : Controller
{
    protected override void OnActionExecuted(ActionExecutedContext context)
    {
        var view = context.Result as ViewResultBase;
        if (view == null) return; // not a view
        var viewName = view.ViewName;
        view.ViewName = GetGlobalizationViewName(viewName, context);
        base.OnActionExecuted(context);
    }
    private static string GetGlobalizationViewName(string viewName, ControllerContext context)
    {
        var cultureName = Thread.CurrentThread.CurrentUICulture.Name;
        if (cultureName == "en-US") return viewName; // default culture
        if (string.IsNullOrEmpty(viewName))
            return context.RouteData.Values["action"] + "." + cultureName; // "Index.fa"
        int i;
        if ((i = viewName.IndexOf('.')) > 0) // ex: Index.cshtml
            return viewName.Substring(0, i + 1) + cultureName + viewName.Substring(i); // "Index.fa.cshtml"
        return viewName + "." + cultureName; // "Index" ==> "Index.fa"
    }
}
```

همانطور که قبلا نیز شرح داده شد، چون متد `ExecuteCore` قبل از `OnActionExecuted` صدا زده میشود بنابراین از تنظیم درست مقدار کالچر در ثرد جاری اطمینان داریم.

روش دیگری که برای مدیریت انتخاب ویوهای مناسب استفاده از یک ویوانجین شخصی سازی شده است. مثلا برای روش سوم نامگذاری ویوها میتوان از کد زیر استفاده کرد:

```
public sealed class RazorGlobalizationViewEngine : RazorViewEngine
{
    protected override IView CreatePartialView(ControllerContext controllerContext, string partialPath)
    {
        return base.CreatePartialView(controllerContext, GetGlobalizationViewPath(controllerContext, partialPath));
    }
    protected override IView CreateView(ControllerContext controllerContext, string viewPath, string masterPath)
    {
        return base.CreateView(controllerContext, GetGlobalizationViewPath(controllerContext, viewPath), masterPath);
    }
    private static string GetGlobalizationViewPath(ControllerContext controllerContext, string viewPath)
    {
        //var controllerName = controllerContext.RouteData.GetRequiredString("controller");
        var request = controllerContext.HttpContext.Request;
        var lang = request.Cookies["MyLanguageCookie"];
        if (lang != null && !string.IsNullOrEmpty(lang.Value) && lang.Value != "en-US")
        {
            var localizedViewPath = Regex.Replace(viewPath, "^~/Views/",
            string.Format("~/Views/Globalization/{0}/", lang.Value));
            if (File.Exists(request.MapPath(localizedViewPath))) viewPath = localizedViewPath;
        }
        return viewPath;
    }
}
```

و برای ثبت این ViewEngine در فایل `Global.asax.cs` خواهیم داشت:

```
protected void Application_Start()
{
    ViewEngines.Engines.Clear();
    ViewEngines.Engines.Add(new RazorGlobalizationViewEngine());
}
```

محتوای یک فایل Resource

ساختار یک فایل .resx به صورت XML استاندارد است. در زیر محتوای یک نمونه فایل Resource با پسوند .resx را مشاهده میکنید:

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <!--
    Microsoft ResX Schema ...
  -->
  <xsd:schema id="root" xmlns="" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    ...
  </xsd:schema>
  <resheader name="resmimetype">
    <value>text/microsoft-resx</value>
  </resheader>
  <resheader name="version">
    <value>2.0</value>
  </resheader>
  <resheader name="reader">
    <value>System.Resources.ResXResourceReader, System.Windows.Forms, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089</value>
  </resheader>
  <resheader name="writer">
    <value>System.Resources.ResXResourceWriter, System.Windows.Forms, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089</value>
  </resheader>
  <data name="RightToLeft" xml:space="preserve">
    <value>>false</value>
    <comment>RightToLeft is false in English!</comment>
  </data>
</root>
```

در قسمت ابتدایی تمام فایل‌های .resx که توسط ویژوال استودیو تولید میشود کامنتی طولانی وجود دارد که به صورت خلاصه به شرح محتوا و ساختار یک فایل Resource میپردازد. در ادامه تگ نسبتاً طولانی xsd:schema قرار دارد. از این قسمت برای معرفی ساختار داده‌ای فایل‌های XML استفاده میشود. برای آشنایی بیشتر با XSD (یا XML Schema) به [اینجا](#) مراجعه کنید. به صورت خلاصه میتوان گفت که XSD برای تعیین ساختار داده‌ها یا تعیین نوع داده‌ای اطلاعات موجود در یک فایل XML به کار میرود. درواقع تگهای XSD به نوعی فایل XML ما را Strongly Typed میکند. با توجه به اطلاعات این قسمت، فایل‌های .resx شامل 4 نوع گره اصلی هستند که عبارتند از: metadata و assembly و data و resheader. در تعریف هر یک از گره‌ها در این قسمت مشخصاتی چون نام زیر گره‌های قابل تعریف در هر گره و نام و نوع خاصیت‌های هر یک معرفی شده است.

بخش موردنظر ما در این مطلب قسمت انتهایی این فایل‌هاست (تگهای resheader و data). همانطور در بالا مشاهده میکنید تگهای reheader شامل تنظیمات مربوط به فایل .resx با ساختاری ساده به صورت name/value است. یکی از این تنظیمات resmimetype فایل resource را معرفی میکند که درواقع مشخص کننده نوع محتوای (Content Type) فایل XML است ([^](#)). برای فایل‌های .resx این مقدار برابر text/microsoft-resx است. تنظیم بعدی نسخه مربوط به فایل .resx (یا Microsoft ResX Schema) را نشان میدهد. در حال حاضر نسخه جاری (در VS 2010) برابر 2.0 است. تنظیم بعدی مربوط به کلاسهای reader و writer تعریف شده برای استفاده از این فایل‌هاست. به نوع این کلاسهای خواننده و نویسنده فایل‌های .resx و مکان فیزیکی و فضای نام آنها دقت کنید که در مطالب بعدی از آنها برای ویرایش و بروزرسانی فایل‌های resource در زمان اجرا استفاده خواهیم کرد.

در پایان نیز تگهای data که برای نگهداری داده‌ها از آنها استفاده میشود. هر گره data شامل یک خاصیت نام (name) و یک زیرگره مقدار (value) است. البته امکان تعیین یک کامنت در زیرگره comment نیز وجود دارد که اختیاری است. هر گره data میتواند شامل خاصیت type و یا mimetype نیز باشد. خاصیت type مشخص کننده نوعی است که تبدیل text/value را با استفاده از ساختار [TypeConverter](#) پشتیبانی میکند. البته اگر در نوع مشخص شده این پشتیبانی وجود نداشته باشد، داده موردنظر پس از سریالایز شدن با فرمت مشخص شده در خاصیت mimetype ذخیره میشود. این mimetype اطلاعات موردنیاز را برای کلاس خواننده این فایل‌ها (ResXResourceReader به صورت پیشفرض) جهت چگونگی بازیابی آبجکت موردنظر فراهم میکند. مشخص کردن این دو خاصیت برای انواع رشته‌ای نیاز نیست. انواع mimetype قابل استفاده عبارتند از:

- application/x-microsoft.net.object.binary.base64: آبجکت موردنظر باید با استفاده از کلاس System.Runtime.Serialization.Formatter.Binary.BinaryFormatter سریالایز شده و سپس با فرمت base64 به یک رشته انکد شود (راجع به انکدینگ base64 و [^](#)).
- application/x-microsoft.net.object.soap.base64: آبجکت موردنظر باید با استفاده از کلاس

شود. `System.Runtime.Serialization.Formatters.Soap.SoapFormatter` سریالایز شده و سپس با فرمت base64 به یک رشته انکد

- application/x-microsoft.net.object.bytearray.base64: آبجکت ابتدا باید با استفاده از یک `System.ComponentModel.TypeConverter` به آرایه ای از بایت سریالایز شده و سپس با فرمت base64 به یک رشته انکد شود. **نکته:** امکان جاسازی کردن (embed) فایل‌های resx. در یک اسمبلی یا کامپایل مستقیم آن به یک سَتلایت اسمبلی (ترجمه مناسبی برای [satellite assembly](#) پیدا نکردم، چیزی شبیه به اسمبلی قمری یا وابسته و از این قبیل ...) وجود ندارد. ابتدا باید این فایل‌های resx. به فایل‌های resources. تبدیل شوند. اینکار با استفاده از ابزار Resource File Generator (نام فایل اجرایی آن resgen.exe است) انجام میشود ([^](#) و [^](#)). سپس میتوان با استفاده از Assembly Linker ستلایت اسمبلی مربوطه را تولید کرد ([^](#)). کل این عملیات در ویژوال استودیو با استفاده از ابزار msbuild به صورت خودکار انجام میشود!

نحوه یافتن کلیدهای Resource در بین فایل‌های مختلف Resx توسط پرووایدر پیش فرض در دات نت

عملیات ابتدا با بررسی خاصیت `CurrentUICulture` از ثرد جاری آغاز میشود. سپس با استفاده از عنوان استاندارد کالچر جاری، فایل مناسب Resource یافته میشود. در نهایت بهترین گزینه موجود برای کلید درخواستی از منابع موجود انتخاب میشود. مثلاً اگر کالچر جاری fa-IR و کلید درخواستی از کلاس Texts باشد ابتدا جستجو برای یافتن فایل Texts.fa-IR.resx آغاز میشود و اگر فایل موردنظر یا کلید درخواستی در این فایل یافته نشد جستجو در فایل Texts.fa.resx ادامه می‌یابد. اگر باز هم یافته نشد در نهایت این عملیات جستجو در فایل resource اصلی خاتمه می‌یابد و مقدار کلید منبع پیش فرض به عنوان نتیجه برگشت داده میشود. یعنی در تمامی حالات سعی میشود تا دقیقترین و بهترین و نزدیکترین نتیجه انتخاب شود. البته در صورتیکه از یک پرووایدر شخصی سازی شده برای کار خود استفاده میکنید باید چنین الگوریتمی را جهت یافتن کلیدهای منابع خود از فایل‌های Resource (یا هر منبع دیگر مثل دیتابیس یا حتی یک وب سرویس) در نظر بگیرید.

Globalization در کلاینت (javascript g11n)

یکی دیگر از موارد استفاده g11n در برنامه نویسی سمت کلاینت است. با وجود استفاده گسترده از جاوا اسکریپت در برنامه نویسی سمت کلاینت در وب اپلیکیشن‌ها، متأسفانه تا همین اواخر عملاً ابزار یا کتابخانه مناسبی برای مدیریت g11n در این زمینه وجود نداشته است. یکی از اولین کتابخانه‌های تولید شده در این زمینه کتابخانه jQuery Globalization است که توسط مایکروسافت توسعه داده شده است (برای آشنایی بیشتر با این کتابخانه به [^](#) و [^](#) مراجعه کنید). این کتابخانه بعداً تغییر نام داده و اکنون با عنوان Globalize شناخته میشود. Globalize یک کتابخانه کاملاً مستقل است که وابستگی به هیچ کتابخانه دیگر ندارد (یعنی برای استفاده از آن نیازی به jQuery نیست). این کتابخانه حاوی کالچرهای بسیاری است که عملیات مختلفی چون فرمت و parse انواع داده‌ها را نیز در سمت کلاینت مدیریت میکند. همچنین با فراهم کردن منابعی حاوی جفت‌های key/culture میتوان از مزایایی مشابه مواردی که در این مطلب بحث شد در سمت کلاینت نیز بهره برد. نشانی این کتابخانه در github [اینجا](#) است. با اینکه خود این کتابخانه ابزار کاملی است اما در بین کالچرهای موجود در فایل‌های آن متأسفانه پشتیبانی کاملی از زبان فارسی نشده است. ابزار دیگری که برای اینکار وجود دارد پلاگین [jquery localize](#) است که برای بحث g11n رشته‌ها پیاده‌سازی بهتر و کاملتری دارد.

در مطالب بعدی به مباحث تغییر مقادیر کلیدهای فایل‌های resource در هنگام اجرا با استفاده از روش مستقیم تغییر محتوای فایل‌ها و کامپایل دوباره توسط ابزار msbuild و نیز استفاده از یک ResourceProvider شخصی سازی شده به عنوان یک راه حل بهتر برای اینکار میپردازم.

در تهیه این مطلب از منابع زیر استفاده شده است: [Localization in ASP.NET MVC – 3 Days Investigation, 1 Day Job](#)

[ASP.NET MVC 3 Internationalization](#)

[Localization and skinning in ASP.NET MVC 3 web applications](#) [Simple ASP.Net MVC Globalization with Graceful](#)

[Fallback](#)

[Globalization, Internationalization and Localization in ASP.NET MVC 3, JavaScript and jQuery - Part 1](#)

نظرات خوانندگان

نویسنده: امیرحسین مرجانی
تاریخ: ۲۳:۵ ۱۳۹۱/۱۰/۲۱

سلام آقای یوسف نژاد
من بعد از تلاش‌های زیاد توی پروژه‌های مختلف این مطالبی که شما نوشته اید رو پیاده سازی کردم ، ولی خیلی پراکنده.
ولی حالا می‌بینم شما به زیبایی این مطالب رو کنار هم قرار دادید.
می‌خواستم بابت مطلب خوب و مفیدتون و همچنین وقتی که گذاشتید تشکر کنم.
ممنونم بابت زحمات شما

اگر ممکنه برچسب MVC رو هم به مطلبتون اضافه کنید.

نویسنده: یوسف نژاد
تاریخ: ۲۳:۲۴ ۱۳۹۱/۱۰/۲۱

با سلام و تشکر بابت نظر لطف شما.
البته باید بگم که همه دوستانی که اینجا به عنوان نویسنده کمک میکنند هدفشون اشتراک مطالبی هست که یاد گرفته اند تا سایر دوستان هم استفاده کنند.

برچسب MVC هم اضافه شد. با تشکر از دقت نظر شما.

نویسنده: امیرحسین جلوداری
تاریخ: ۱:۳ ۱۳۹۱/۱۰/۲۲

کاملا مشخصه که مطلب از روی تجربه‌ی کاریه و بسیار عالی جمع آوری شده ... ممنون ... به طرز عجیبی منتظر قسمت بعدم :دی

نویسنده: پندار
تاریخ: ۲۱:۳۸ ۱۳۹۱/۱۲/۰۸

گویا در MVC 4 این روش پاسخ نمیدهد. لطفا در این مورد برای MVC 4 راه حلی بدهید

نویسنده: محسن
تاریخ: ۲۳:۶ ۱۳۹۱/۱۲/۰۸

MVC 4 فقط یک سری افزونه بیشتر از MVC3 داره. مثلا razor آن بهبود پیدا کرده، فشرده سازی فایل‌های CSS به اون اضافه شده یا Web API رو به صورت یکپارچه داره. از لحاظ کار با فایل‌های منبع فرقی نکرده.

نویسنده: پندار
تاریخ: ۹:۲۱ ۱۳۹۱/۱۲/۰۹

متن نشانی زیر را مطالعه کنید

<http://geekswithblogs.net/shaunxu/archive/2012/09/04/localization-in-asp.net-mvc-ndash-upgraded.aspx>

نویسنده: محسن
تاریخ: ۹:۴۳ ۱۳۹۱/۱۲/۰۹

مطلبی که لینک دادی در مورد آپدیت یک helper شخصی توسعه داده شده توسط شخص ثالث است از MVC2 به MVC4. اگر کسی

از این راه حل شخصی و خاص استفاده نکرده باشه، اصول فوق فرقی نکرده.

نویسنده: صابر فتح الهی
تاریخ: ۱۶:۵ ۱۳۹۱/۱۲/۱۴

مطلب خیلی خوبی بود کلی استفاده کردیم.
مهندس کالچر زبان کردی چی میشه؟ توی لیست منابعی که دادین گیر نیاوردم

نویسنده: وحید نصیری
تاریخ: ۱۷:۲۲ ۱۳۹۱/۱۲/۱۴

kur هست [مطابق استاندارد](#) .

نویسنده: صابر فتح الهی
تاریخ: ۲:۱۱ ۱۳۹۱/۱۲/۱۷

سلام
اما مهندس کلاس Culture Info این مقدار قبول نمی‌کنه

نویسنده: وحید نصیری
تاریخ: ۹:۳ ۱۳۹۱/۱۲/۱۷

می‌تونید کلاس [فرهنگ سفارشی](#) را ایجاد و [استفاده](#) کنید.

نویسنده: صابر فتح الهی
تاریخ: ۱۰:۱۲ ۱۳۹۱/۱۲/۱۷

اما روش گفته شده نیاز به دسترسی مدیریت دارد که روی سرورهای اشتراکی ممکن نیست

نویسنده: وحید نصیری
تاریخ: ۱۱:۱۹ ۱۳۹۱/۱۲/۱۷

نحوه توسعه اکثر برنامه‌ها و کتابخانه‌ها در طول زمان، بر اساس تقاضا و پیگیری مصرف کننده است. اگر بعد از بیش از 10 سال، چنین فرهنگی اضافه نشده یعنی درخواستی نداشته. مراجعه کنید به [محل پیگیری این نوع مسایل](#) .

نویسنده: صابر فتح الهی
تاریخ: ۱۰:۱۴ ۱۳۹۱/۱۲/۱۹

سلام مهندس یوسف نژاد (ابتدا ممنونم از پست خوب شما)
با پیروی از پست شما
ابتدا فایل‌های ریسورس در پروژه جاری فولدر App_GlobalResources گذاشتم و پروژه در صفحات aspx با قالب زیر به راحتی
تغییر زبان داده میشد:

```
<asp:Literal ID="Literal1" Text='<%%$ Resources:resource, Title %>' runat="server" />
```

اما بعدش فایل هارو توی یک پروژه کتابخانه ای جدید گذاشتم و Build Action فایل‌های ریسورس روی Embedded Resource
تنظیم کردم، پروژه با موفقیت اجرا شد و در سمت سرور با کد زیر راحت به مقادیر دسترسی دارم:

```
Literal1.Text=ResourceManager.Resource.Title;
```


اما در سمت صفحات aspx با کد قبلی به شکل زیر نمایش نمیده و خطا صادر میشه:

```
<asp:Literal ID="Literal1" runat="server" Text='<%$ ResourceManager.Resource:resource, Title %>' />
```

و خطای زیر صادر میشه:

Parser Error

Description: An error occurred during the parsing of a resource required to service this request. Please review the following specific parse error details and modify your source file appropriately.

Parser Error Message: The expression prefix 'ResourceManager.Resource' was not recognized. Please correct the prefix or register the prefix in the <expressionBuilders> section of configuration.

Source Error:

مراحل این [یست](#) روی هم دنبال کردم اما باز نمشد.
چه تنظیماتی ست نکردم ؟

نویسنده: یوسف نژاد
تاریخ: ۱۳۹۲/۰۱/۳۱ ۱۲:۴۴

ببخشید یه چند وقتی فعال نبودم و پاسخ این سوال رو دیر دارم میدم.
امکان استفاده از کلیدهای Resource برای مقداردهی خواص سمت سرور کنترلها در صفحات aspx به صورت مستقیم وجود ندارد. بنابراین برای استفاده از این کلیدها همانند روش پیش فرض موجود در ASP.NET باید از یکسری ExpressionBuilder استفاده شود که کار Parse عبارت وارده برای این خواص را در سمت سرور انجام میدهد. کلاس پیش فرض برای اینکار در ASP.NET Web Form که از پیشوند Resources استفاده میکند تنها برای Resourceهای محلی (Local) موجود در فولدرهای پیش فرض (App_GlobalResources و App_LocalResources) کاربرد دارد و برای استفاده از Resourceهای موجود در منابع ریفرنس داده شده به پروژه باید از روشی مثل اونچه که خود شما لینکش رو دادین استفاده کرد.
من این روش رو استفاده کردم و پیاده سازی موفق داشتم. نمیدونم مشکل شما چیه...

نویسنده: یوسف نژاد
تاریخ: ۱۳۹۲/۰۱/۳۱ ۱۲:۵۲

اگر مشکلی در پیاده سازی روش بالا دارین، تمام مراحل که من طی کردم دقیقا اینجا میارم:
ابتدا کلاس ExpressionBuilder رو به صورت زیر مثلا در خود پروژه Resources اضافه میکنیم:

```
using System.Web.Compilation;
using System.CodeDom;
namespace Resources
{
    [ExpressionPrefix("MyResource")]
    public class ResourceExpressionBuilder : ExpressionBuilder
    {
        public override System.CodeDom.CodeExpression GetCodeExpression(System.Web.UI.BindPropertyEntry entry, object parsedData, System.Web.Compilation.ExpressionBuilderContext context)
        {
            return new CodeSnippetExpression(entry.Expression);
        }
    }
}
```

سپس تنظیمات زیر رو به Web.config اضافه میکنیم:

```
<compilation debug="true" targetFramework="4.0">
  <expressionBuilders>
    <add expressionPrefix="MyResource" type="Resources.ResourceExpressionBuilder, Resources" />
  </expressionBuilders>
</compilation>
```

```
</expressionBuilders>
</compilation>
```

در نهایت به صورت زیر میتوان از این کلاس استفاده کرد:

```
<asp:Literal ID="Literal1" runat="server" Text="<%"$ MyResource: Resources.Resource1.String2 %"> />
```

هرچند ظاهراً مقدار پیشوند معرفی شده در Attribute کلاس ResourceExpressionBuilder اهمیت چندانی ندارد! امیدوارم مشکلتون حل بشه.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۲/۰۱ ۲:۲۹

ممنونم از پاسخ شما
همون روش شمارو دنبال کردم پاسخ گرفتم، اشکال از خودم بود
با تشکر از شما

نویسنده: صادق نجاتی
تاریخ: ۱۳۹۲/۱۲/۲۷ ۱۲:۰۰

با سلام
ضمن تشکر از مطلب بسیار خوبتون
خاصیت DisplayFormat قابلیت استفاده از کلیدهای Resource را ندارد !
لطفا راهنمایی فرمایید که چطور میشه از این خاصیت برای DisplayFormat استفاده کرد؟
من می‌خواهم برای تاریخ در زبانه فارسی از فرمت {yyyy-MM-dd} و در زبانه انگلیسی از {yyyy-dd-MM} استفاده کنم.
با سپاس فراوان

عنوان:	MSBuild
نویسنده:	یوسف نژاد
تاریخ:	۲۰:۵۰ ۱۳۹۱/۱۱/۰۸
آدرس:	www.dotnettips.info
برچسب‌ها:	NET, MSBuild.

MSBuild

به عنوان یک تعریف کلی، مایکروسافت بیلد (Microsoft Build)، پلتفرمی برای ساخت اپلیکیشن‌هاست. در این پلتفرم (که با عنوان MSBuild شناخته میشود) کلیه تنظیمات لازم برای تولید و ساخت یک اپلیکیشن درون یک فایل XML ذخیره میشود، که به آن **فایل پروژه** میگویند. ویژوال استودیو نیز از این ابزار برای تولید تمامی اپلیکیشن‌ها استفاده می‌کند، اما MSBuild به ویژوال استودیو وابسته نیست و کاملاً مستقل از آن است.

این ابزار به همراه دات نت فریمورک (البته نسخه کامل آن و نه نسخه‌های سبکتری چون Client Profile) نصب میشود. بنابراین با استفاده از فایل اجرایی این ابزار (msbuild.exe) میتوان فرایند بیلد را برای پروژه و یا سولوشن‌های خود، بدون نیاز به نصب ویژوال استودیو اجرا کرد. استفاده مستقیم از MSBuild در شرایط زیر نیاز میشود:

- ویژوال استودیو در دسترس نباشد.

- نسخه 64 بیتی این ابزار که در ویژوال استودیو در دسترس نیست. البته در بیشتر مواقع این مورد پیش نخواهد آمد مگر اینکه برای فرایند بیلد به حافظه بیشتری نیاز باشد.

- اجرای فرایند بیلد در بیش از یک پراسس (برای رسیدن به سرعت بالاتر). این امکان در تولید پروژه‌های C++ در ویژوال استودیو موجود است. همچنین از نسخه 2012 این امکان برای پروژه‌های C# نیز فراهم شده است.

- سفارشی‌سازی فرایند بیلد

- و ...

همچنین یکی دیگر از بخشهای مهم فرایند تولید اپلیکیشن که همانند ویژوال استودیو از این ابزار بصورت مستقیم استفاده میکند Team Foundation Build است.

با استفاده از خط فرمان این ابزار تنظیمات فراوانی را برای سفارشی سازی عملیات بیلد میتوان انجام داد که شرح آنها بحثی مفصل میطلبد. تنظیمات بسیار دیگری هم در فایل پروژه قابل اعمال است (توضیحات بیشتر در [اینجا](#)). منابع برای مطالعه بیشتر:

[MSBuild Reference](#)

[\(Visual Studio Integration \(MSBuild](#)

[Walkthrough: Using MSBuild](#)

Microsoft Build API

در دات‌نت فریمورک فضای نامی با عنوان Microsoft.Build نیز وجود دارد که امکانات این ابزار را در اختیار برنامه نویسی قرار میدهد. برای استفاده از این کتابخانه باید ارجاعی به اسمبلی آن داد، که به همین نام بوده و به همراه دات‌نت فریمورک نصب میشود. کد زیر نحوه استفاده اولیه از این کتابخانه را نشان میدهد:

```
private static void TestMSBuild(string projectFullPath)
{
    var pc = new ProjectCollection();
    var globalProperties = new Dictionary<string, string>() { { "Configuration", "Debug" }, { "Platform", "AnyCPU" } };
    var buildRequest = new BuildRequestData(projectFullPath, globalProperties, null, new string[] { "Build" }, null);
    var buildResult = BuildManager.DefaultBuildManager.Build(new BuildParameters(pc), buildRequest);
}
```

با اینکه ارائه مقداری غیرنال برای آرگومان globalProperties اجباری است اما پرکردن آن کاملاً اختیاری است، زیرا تمام تنظیمات ممکن را میتوان در خود فایل پروژه ثبت کرد.

برای مطالعه بیشتر منابع زیر پیشنهاد میشود: [Microsoft.Build](#)

[NET 4.0 MSBuild API introduction.](#)

استفاده از msbuild.exe

ابزار msbuild به صورت یک فایل exe در دسترس است و برای استفاده از آن میتوان از خط فرمان ویندوز استفاده کرد. مسیر فایل اجرایی آن (MSBuild.exe) در ریشه مسیر دات نت فریمورک است، بصورت زیر:

نسخه 32 بیتی:

C:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe

نسخه 64 بیتی:

C:\Windows\Microsoft.NET\Framework64\v4.0.30319\MSBuild.exe

برای استفاده از آن میتوان مسیر فایل پروژه یا سولوشن (فایل با پسوند .csproj یا .vbproj یا .sln) را به آن داد تا سایر عملیات تولید را به صورت خودکار تا آخر به انجام برساند. کاری که عینا در ویژوال استودیو در زمان Build انجام میشود! برای بهره برداری از آن در کد میتوان از کلاس Process استفاده کرد. برای مسیر این فایل هم میتوان از نشانی‌هایی که در بالا معرفی شد استفاده کرد یا برای راحتی و امنیت بیشتر از کلید رجیستری مربوطه که در کد زیر نشان داده شده استفاده کرد:

```
private static void TestMSBuild1(string projectPath)
{
    var regKey = Registry.LocalMachine.OpenSubKey(@"SOFTWARE\Microsoft\MSBuild\ToolsVersions\4.0");
    if (regKey == null) return;
    var msBuildExeFilePath = Path.Combine(regKey.GetValue("MSBuildToolsPath").ToString(), "MSBuild.exe");
    var startInfo = new ProcessStartInfo
    {
        FileName = msBuildExeFilePath,
        Arguments = projectPath,
        WindowStyle = ProcessWindowStyle.Hidden
    };
    var process = Process.Start(startInfo);
    process.WaitForExit();
}
```

بدین ترتیب عملیاتی مشابه عملیات Build در ویژوال استودیو انجام میشود و با توجه به تنظیمات موجود در فایل پروژه، پوشه‌های خروجی (مثلا bin و obj در حالت پیش فرض پروژه‌های ویژوال استودیو) نیز در مسیرهای مربوطه ایجاد میگردد.

چند وقت پیش زمانی که قصد داشتم از یک Portable Class Library که تحت دات نت 4 بود توی پروژه ام استفاده کنم متوجه شدم که این نوع Class Library از فضای نام System.Threading.Task پشتیبانی نمی‌کند. قصد داشتم که از این فضای نام برای بحث TPL توی پروژه ام استفاده کنم و چند تا متد Async بنویسم. زمانی که سعی کردم با استفاده از AsyncAwaitCTP:Nuget نصب کنم با خطای زیر روبرو شدم.

دستور مورد نظر در Nuget :

```
PM> install-package AsyncAwaitCtp
```

و اما خطا

```
Install failed. Rolling back...
install-package : Could not install package 'AsyncAwaitCTP 1.0'. You are trying to install this package into a project that targets '.NETPortable,Version=v4.0,Profile=Profile3', but the package does not contain any assembly references that are compatible with that framework. For more information, contact the package author.
At line:1 char:1
```

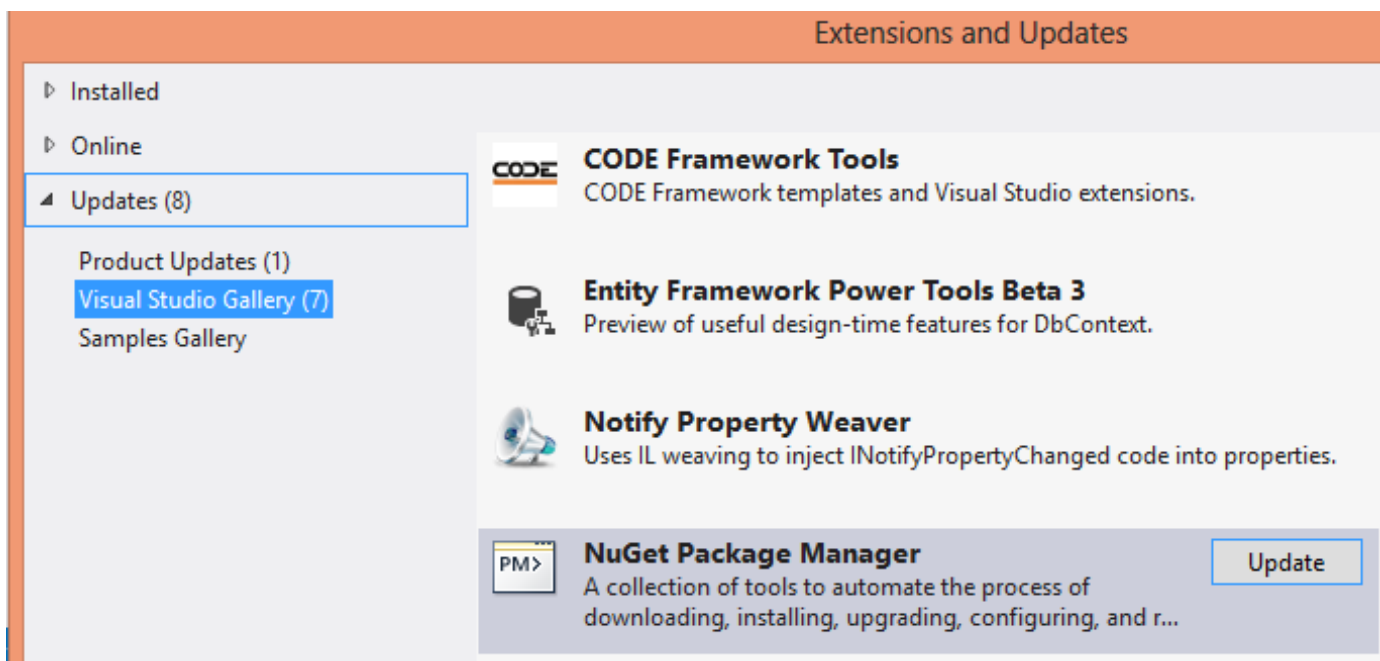
دستور بالا برای نصب AsyncAwaitCTP در Class Library تحت دات نت 4.5 استفاده می‌شه. خلاصه بعد از یکم جستجو متوجه شدم که باید از دستور زیر برای نصب TPL توی Portable Class Library تحت دات نت 4 استفاده کنم.

```
PM > install-Package Microsoft-BCL-Async -Pre
```

که بازم با خطای زیر مواجه شدم.

```
Install failed. Rolling back...
install-package : Could not install package 'Microsoft.Bcl 1.0.16-rc'. You are trying to install this package into a project that targets '.NETPortable,Version=v4.0,Profile=Profile3', but the package does not contain any assembly references that are compatible with that framework. For more information, contact the package author.
At line:1 char:1
+ install-package Microsoft.BCL.Async -Pre
+ ~~~~~
+ CategoryInfo          : NotSpecified: (:) [Install-Package], InvalidOperationException
+ FullyQualifiedErrorId : NuGetCmdletUnhandledException,NuGet.PowerShell.Commands.InstallPackageCommand
```

از اونجا که از دستور بالا مطمئن بودم و می‌دونستم که باید درست کار کنه فهمیدم اشکال کار از یه جای دیگه است. خلاصه بعد از یه جستجوی 25 دقیقه ای متوجه شدم که Nuget نصب شده روی سیستم من Update نیست. برای همین به روش زیر عمل کردم. از منوی Tools گزینه Extension And Updates رو انتخاب کردم. بعد از صفحه مورد نظر گزینه Updates روز از منوی سمت چپ انتخاب کردم و در نهایت گزینه Nuget Package Manager. مثل شکل زیر:



بعد از اتمام عملیات Update دوباره دستورات مورد نظر رو وارد کردم که به خوبی عملیات نصب CTP به اتمام رسید.

عنوان: ایجاد رشته Alphanumeric تصادفی در سی شارپ

نویسنده: امیر هاشم زاده

تاریخ: ۱۹:۴۵ ۱۳۹۲/۰۲/۲۰

آدرس: www.dotnettips.info

برچسب‌ها: C#, .NET, Random, alphanumeric

برای ایجاد یک رشته تصادفی [Alphanumeric](#) (شامل حرف و عدد) روشهای زیادی وجود دارد ولی در اینجا به تشریح 2 روش آن اکتفا می‌کنیم.

روش کلی: ابتدا بازه رشته تصادفی مورد نظر را تعیین می‌کنیم. سپس به اندازه طول رشته، اندیس تصادفی ایجاد می‌کنیم و بوسیله آنها کاراکتر تصادفی را از بازه بدست می‌آوریم و در انتها کاراکترهای تصادفی را با هم ادغام کرده تا رشته نهایی حاصل شود. روش اول:

ابتدا بازه (char) رشته را مشخص می‌کنیم.

```
var chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
```

سپس بوسیله LINQ آن را به اندازه طول رشته دلخواه (در این مثال 8 کاراکتر) تکرار می‌کنیم و برای انتخاب تصادفی یک کاراکتر در هر بازه (char) تکرار شده از کلاس جهت بدست آوردن اندیس تصادفی بازه استفاده می‌کنیم.

```
var random = new Random();
var result = new string(
    Enumerable.Repeat(chars, 8)
        .Select(s => s[random.Next(s.Length)])
        .ToArray());
```

توجه: از این روش برای هیچ کدام از موارد مهم و کلیدی مانند ساخت کلمه عبور و توکن استفاده نکنید. روش دوم:

همانند روش اول ابتدا بازه رشته را تعیین می‌کنیم.

```
char[] chars = new char[62];
chars="abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890".ToCharArray();
```

بعد از تعریف بازه، یک سری اعداد تصادفی غیر صفر را بوسیله کلاس [RNGCryptoServiceProvider](#) و متد [GetNonZeroBytes](#) آن، در متغیری که قرار است بعداً در ایجاد رشته‌ی تصادفی نیاز است پر می‌کنیم.

```
byte[] data = new byte[maxSize];
RNGCryptoServiceProvider crypto = new RNGCryptoServiceProvider();
crypto.GetNonZeroBytes(data);
```

در این مرحله به تعداد طول رشته تصادفی مورد نظر عدد تصادفی بین 0 تا 255 ذخیره شده در متغیر data داریم، برای ایجاد اندیس تصادفی از باقیمانده عدد تصادفی ایجاد شده در مرحله قبل (byte) به طول بازه (chars.Length) استفاده می‌کنیم سپس کاراکترهای تصادفی را کنار یکدیگر قرار می‌دهیم.

```
StringBuilder result = new StringBuilder(maxSize);
foreach (byte b in data)
{
    result.Append(chars[b % (chars.Length)]);
}
```

و در نهایت متد ما جهت ایجاد رشته Alphanumeric در روش دوم به شکل زیر خواهد بود:

```
public static string GetRandomAlphaNumeric (int maxSize)
{
    char[] chars = new char[62];
    chars = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890".ToCharArray();
    RNGCryptoServiceProvider crypto = new RNGCryptoServiceProvider();
    byte[] data = new byte[maxSize];
    crypto.GetNonZeroBytes(data);
    StringBuilder result = new StringBuilder(maxSize);
    foreach (byte b in data)
    {
        result.Append(chars[b % (chars.Length)]);
    }
    return result.ToString();
}
```

لازم به یادآوری است که رشته ایجاد شده در 2 روش بیان شده **منحصر بفرد** نیست بلکه **تصادفی** است، درواقع تصادفی بودن با منحصر بودن متفاوت است، برای ایجاد رشته‌های منحصر بفرد روشهایی (البته این روشها 100 درصد نیستند ولی از قابلیت اطمینان بالایی برخوردار هستند) وجود دارد که پست‌های بعدی به آنها اشاره خواهیم کرد.

تعامل MATLAB (متلب) با دات نت - قسمت اول

عنوان:

نویسنده: مسعود مشهدی

تاریخ: ۱۵:۰ ۱۳۹۲/۰۳/۱۴

آدرس: www.dotnettips.info

برچسب‌ها: NET, C#.NET, MATLAB.

متلب (MATLAB) یکی از پرکاربردترین نرم افزارهای محاسباتی در حوزه مهندسی بویژه برق، ریاضیات، مکانیک و ... می‌باشد. بدون شک تعامل نرم افزارهای مختلف با هم در جهت کاربردی‌تر کردن یک پروژه کمک بسزایی به کاربران نهایی می‌کند. قطعاً استفاده از علوم روز همچون شبکه‌های عصبی، منطق فازی و الگوریتم‌های تکاملی همچون ژنتیک بدون استفاده از متلب بسیار سخت و پیچیده خواهد بود. دستورات و تابع‌های (functions) آماده و ساده در متلب در جهت استفاده از این علوم تقریباً هر پژوهشگر و کاربری را ترغیب به استفاده از متلب می‌کند. طبقاً استفاده از کتابخانه‌های دانت در متلب کمک بسیاری به توسعه دهندگان این حوزه می‌کند.

در این سری از مطالب سعی بر بررسی این تعامل شده است.

بطور کلی دو نوع تعامل در این زمینه وجود دارد :

1- استفاده از اسمبلی‌های دات نت در متلب تحت عنوان MATLAB .NET Interface

2- استفاده از پکیج تابع‌های متلب در پروژه‌های مبتنی بر دات نت تحت عنوان MATLAB Builder NE

در مورد اول از دات نت فقط در پلت فورم ویندوز استفاده می‌شود. کلیه امکانات دات نت 2 را ساپورت میکند و با ورژن‌های 3 و 3.5 سازگار است اما با ورژن 4 تنها بعضی از امکانات در دسترس است و هنوز مورد تست کلی قرار نگرفته است. کلیه امکانات دات نت در C# در متلب بجز یک سری از موارد که در جدول زیر ذکر شده است در دسترس است.

Features Not Supported in MATLAB
Cannot use <code>ClassName.propertyname</code> syntax to set static properties. Use NET.setStaticProperty instead.
Unloading an assembly
Passing a structure array, sparse array, or complex number to a .NET property or method
Subclassing .NET classes from MATLAB
Accessing nonpublic class members
Displaying generic methods using <code>methods</code> or <code>methodsview</code> functions. For a workaround, see Display .NET Generic Methods Using Reflection .
Creating an instance of a nested class. For a workaround, see Working With Nested Classes .
Saving (serializing) .NET objects into a MAT-file
Creating .NET arrays with a specific lower bound
Concatenating multiple .NET objects into an array
Implementing interface methods
Hosting .NET controls in figure windows
Casting operations
Calling constructors with <code>ref</code> or <code>out</code> type arguments
Using <code>System.Console.WriteLine</code> to write text to the command window
Pointer type arguments, function pointers, <code>Dllimport</code> keyword
.NET remoting
Using the MATLAB: (colon) operator in a <code>foreach</code> iteration
Adding event listeners to .NET events defined in static classes
Handling .NET events with signatures that do not conform to the standard signature
Creating empty .NET objects
Creating .NET objects that do not belong to a namespace

به عنوان مثال از کلاس `speech synthesizer` دات نت 3 در متلب بصورت زیر استفاده می‌کنیم :

```
function Speak(text)
    NET.addAssembly('System.Speech');
    speak = System.Speech.Synthesis.SpeechSynthesizer;
    speak.Volume = 100;
```

```
end          Speak(speak,text);
```

سپس برای رندر کردن یک متن به صوت دستور زیر را اجرا می‌کنیم :

```
Speak('You can use .NET Libraries in MATLAB');
```

در ارتباط با استفاده از توابع متلب در یک پروژه مبتنی بر دات نت در قسمت بعد توضیح داده خواهد شد.

منبع : Help متلب

نظرات خوانندگان

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۳/۱۶ ۰:۵۸

در این کدی که تهیه کردید، Speak داخل متد Speak [چطور فراخوانی شده](#) بدون ارجاع به یک شیء؟ (سطر آخر متد). بعد در متلب نیازی به new نیست؟ فقط فراخوانی NET.addAssembly باعث شناسایی System.Speech.Synthesis.SpeechSynthesizer میشه؟

نویسنده: مسعود مشهدی
تاریخ: ۱۳۹۲/۰۳/۱۶ ۹:۱۸

متلب به حروف کوچک و بزرگ حساس است. دقت کرده باشید speak داخل Speak است. در حقیقت Speak نام تابع است. در متلب تابع NET.addAssembly کار بارگذاری اسمبلی‌های یاد شده را دارد. خیر در متلب احتیاجی به new نیست. در کل برنامه نویسی در متلب کمی متفاوت با زبان‌های معمول (C# و ..) است.

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۳/۱۶ ۱۰:۲۰

اینجا در سطر آخر متد نباید نوشته می‌شد speak.Speak ؟ یا اینکه این Speak نوشته شده [با دو پارامتر](#) ، منظور تابعی دیگر هست؟

نویسنده: مسعود مشهدی
تاریخ: ۱۳۹۲/۰۳/۱۶ ۱۲:۱۸

نه دوست عزیز همونطور که گفتم ساختار متلب متفاوت هست.
منظور از Speak همون تابع است.

کافیست تابع فوق را در یک M فایل (کدها در متلب با پسوند ام فایل ذخیره می‌شوند) ذخیره کنید. سپس در داخل متلب مسیر اجرای کد را به مسیر M فایل تغییر دهید و سپس دستور رندر کردن رو که بالا گفته شد در Command Window اجرا کنید. متن فوق به تابع Speak ارجاع داده می‌شود.

نویسنده: علی
تاریخ: ۱۳۹۲/۰۳/۲۳ ۲۲:۳۷

استفاده از Matlab در دات نت نیاز به نصب کلی خود Matlab داره؟

نویسنده: مسعود مشهدی
تاریخ: ۱۳۹۲/۰۳/۲۴ ۲۱:۳۰

در این مورد در قسمت بعد توضیح خواهم داد. بله احتیاج است. اگر تصمیم به نوشتن کدها و کامپایل کردن اونها دارید. خیر، اگر نویسنده کدها یک پکیج از پروژه متلب در اختیار شما قرار بدهد.

دسترسی به داده‌ها پیش شرط انجام همه‌ی منطق‌های اکثر نرم افزارهای تجاری می‌باشد. داده‌های ممکن در حافظه ، پایگاه داده ، فایل‌های فیزیکی و هر منبع دیگری قرار گرفته باشند. هنگامی که حجم داده‌ها کم باشد شاید روش دسترسی و الگوریتم مورد استفاده اهمیتی نداشته باشد اما با افزایش حجم داده‌ها روش‌های بهینه‌تر تاثیر مستقیم در کارایی برنامه دارند. در این مثال سعی بر این است که در یک سناریوی خاص تفاوت بین Dictionary و List را بررسی کنیم : فرض کنید 2 کلاس Student و Grade موجود است که وظیفه‌ی نگهداری اطلاعات دانش آموز و نمره را بر عهده دارند.

```
public class Grade
{
    public Guid StudentId { get; set; }
    public string Value { get; set; }

    public static IEnumerable<Grade> GetData()
    {
        for (int i = 0; i < 10000; i++)
        {
            yield return new Grade
            {
                StudentId = GuidHelper.ListOfIds[i], Value = "Value " + i
            };
        }
    }
}

public class Student
{
    public Guid Id { get; set; }
    public string Name { get; set; }
    public string Grade { get; set; }

    public static IEnumerable<Student> GetStudents()
    {
        for (int i = 0; i < 10000; i++)
        {
            yield return new Student
            {
                Id = GuidHelper.ListOfIds[i],
                Name = "Name " + i
            };
        }
    }
}
```

از کلاس GuidHelper برای تولید و نگهداری شناسه‌های یکتا برای دانش آموز کمک گرفته شده است :

```
public class GuidHelper
{
    public static List<Guid> ListOfIds=new List<Guid>();

    static GuidHelper()
    {
        for (int i = 0; i < 10000; i++)
        {
            ListOfIds.Add(Guid.NewGuid());
        }
    }
}
```

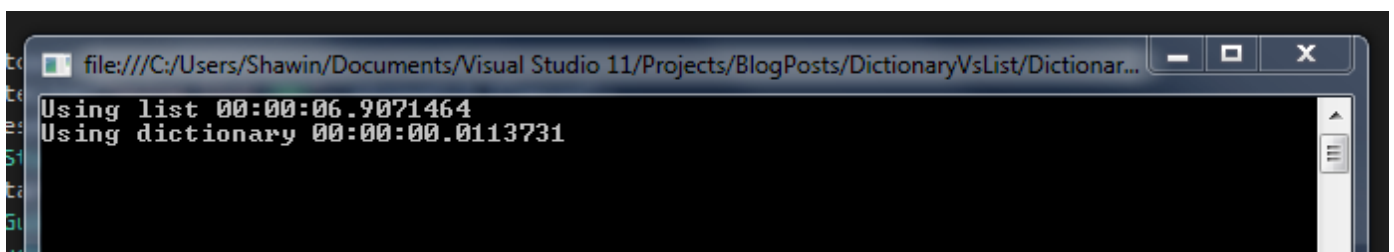
سپس لیستی از دانش آموزان و نمرات را درون حافظه ایجاد کرده و با یک حلقه نمره‌ی هر دانش آموز به Property مورد نظر مقدار داده می‌شود.

ابتدا از LINQ روی لیست برای پیدا کردن نمره‌ی مورد نظر استفاده کرده و در روش دوم برای پیدا کردن نمره‌ی هر دانش آموز از Dictionary استفاده شده :

```
internal class Program
{
    private static void Main(string[] args)
    {
        var stopwatch = new Stopwatch();
        List<Grade> grades = Grade.GetData().ToList();
        List<Student> students = Student.GetStudents().ToList();

        stopwatch.Start();
        foreach (Student student in students)
        {
            student.Grade = grades.Single(x => x.StudentId == student.Id).Value;
        }
        stopwatch.Stop();
        Console.WriteLine("Using list {0}", stopwatch.Elapsed);
        stopwatch.Reset();
        students = Student.GetStudents().ToList();
        stopwatch.Start();
        Dictionary<Guid, string> dictionary = Grade.GetData().ToDictionary(x => x.StudentId, x =>
x.Value);
        foreach (Student student in students)
        {
            student.Grade = dictionary[student.Id];
        }
        stopwatch.Stop();
        Console.WriteLine("Using dictionary {0}", stopwatch.Elapsed);
        Console.ReadKey();
    }
}
```

نتیجه‌ی مقایسه در سیستم من اینگونه می‌باشد :



همانگونه که مشاهده می‌شود در این سناریو خواندن نمره از روی Dictionary بر اساس 'کلید' بسیار سریع‌تر از انجام یک پرس و جوی LINQ روی لیست است.

زمانی که از LINQ on list

```
student.Grade = grades.Single(x => x.StudentId == student.Id).Value;
```

برای پیدا کردن مقدار مورد نظر یک به یک روی اعضا لیست حرکت می‌کند تا به مقدار مورد نظر برسد در نتیجه پیچیدگی زمانی آن $O(n)$ هست. پس هر چه میزان داده‌ها بیشتر باشد این روش کندتر می‌شود.

زمانی که از Dictionary

```
student.Grade = dictionary[student.Id];
```

برای پیدا کردن مقدار استفاده می‌شود با اولین تلاش مقدار مورد نظر یافت می‌شود پس پیچیدگی زمانی آن 1 0 می‌باشد.

در نتیجه اگر نیاز به پیدا کردن اطلاعات بر اساس یک مقدار یکتا یا کلید باشد تبدیل اطلاعات به Dictionary و خواندن از آن بسیار به صرفه‌تر است.

تفاوت این 2 روش وقتی مشخص می‌شود که میزان داده‌ها زیاد باشد.

در همین رابطه ([1](#) ، [2](#))

[DictionaryVsList.zip](#)

نظرات خوانندگان

نویسنده: حسین مرادی نیا
تاریخ: ۱۳۹۲/۰۳/۱۷ ۲۱:۳۵

یه نگاهی هم به این بندازید. جالبه: <http://stackoverflow.com/questions/1009107/what-net-collection-provides-the-fastest-search>

نویسنده: مهدی فرزاد
تاریخ: ۱۳۹۲/۰۳/۱۸ ۰:۲

با تشکر از دوست خوبم ، یک سؤال مطرح میشه شما این نتیجه رو از روی داده‌های موجود در حافظه انجام دادید ، اگر این داده‌ها در دیتا بیس باشه و با استفاده از یک ORM مثل EF به داده‌ها دسترسی داشته باشیم برای استفاده از Dictionary ابتدا تمام داده‌ها یک بار واکنشی شده و در نتیجه جستجو میشه؟ آیا این مطلب درسته؟ اگر آره پس نتیجه به نفع Linq تغییر میکنه

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۳/۱۸ ۰:۲۴

نه. ToList یا ToDictionary اصطلاحاً یک نوع Projection هستند و پس از دریافت اطلاعات مطابق کوئری لینک شما اعمال خواهند شد (شکل دادن به اطلاعات دریافت شده از بانک اطلاعاتی؛ فرضاً 100 رکورد دریافت شده، حالا شما خواستید از این رکوردها برای استفاده، List درست کنید یا دیکشنری یا حالت‌های دیگر).

در [قسمت قبل](#) در مورد استفاده دات نت در متلب توضیح داده شد. در این قسمت به نحوه استفاده توابع متلب در دات نت بصورت ساده می‌پردازیم.

فرض کنید تیم برنامه‌نویس متلب و تیم برنامه‌نویس دات نت در تعامل با یکدیگر هستند. وظیفه تیم برنامه‌نویس متلب به شرح زیر می‌باشد :

- 1- نوشتن توابع در متلب و تست کردن آنها جهت توسعه و ارائه مناسب به تیم مقابل
- 2- درست کردن کامپوننت دات نت در متلب با استفاده از محیط Deployment Tool GUI (با اجرای دستور deploytool در متلب)
- 3- استفاده از یک پکیج بسته‌بندی شده از فایل‌های قابل ارائه به تیم مقابل (اختیاری)
- 4- کپی پکیج در محل از قبل تعیین شده توسط دو تیم یا ارائه آن به تیم مقابل جهت استفاده

برای مثال M فایل (اصطلاح فایل‌ها در متلب همانند کلاس در دات نت) makesquare.m را که در مسیر

```
matlabroot\toolbox\dotnetbuilder\Examples\VS8\NET\MagicSquareExample\MagicSquareComp
```

است را در نظر بگیرید :

```
function y = makesquare(x)
%MAKESQUARE Magic square of size x.
% Y = MAKESQUARE(X) returns a magic square of size x.
% This file is used as an example for the MATLAB
% Builder NE product.

% Copyright 2001-2012 The MathWorks, Inc.

y = magic(x);
```

تابع magic یک ماتریس در ابعاد x در x درست می‌کند که درایه‌های آن اعداد صحیح از 1 تا x^2 بوده و مجموع سطر و ستون‌های آن با هم برابر است. x باید بزرگتر یا مساوی 3 باشد.

در صورتی که x برابر 5 انتخاب شود خروجی متلب بصورت زیر خواهد بود :

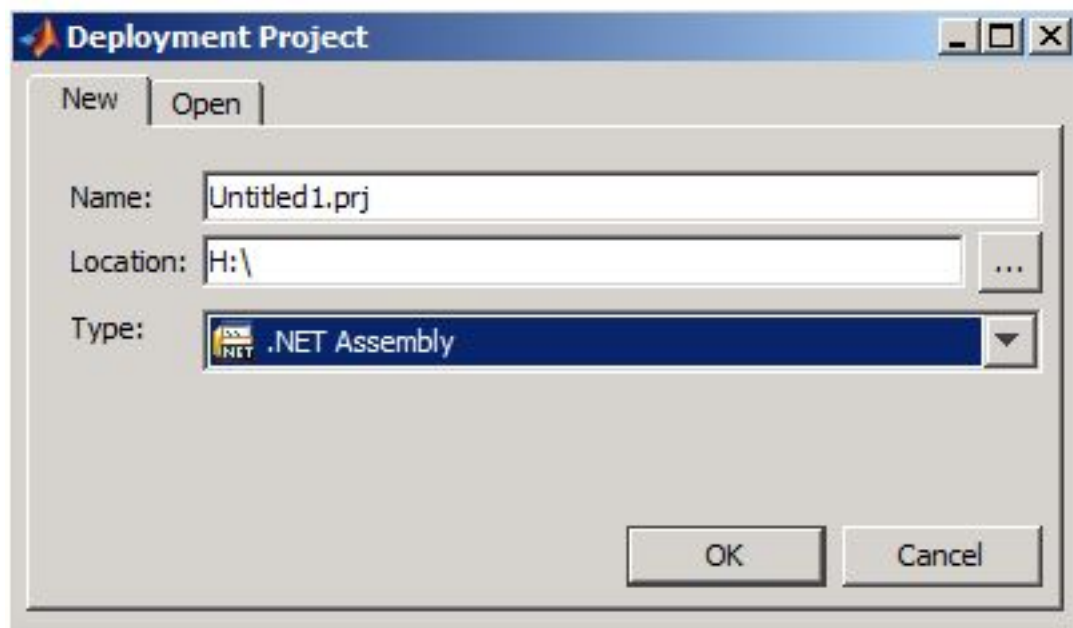
```
17 24 1 8 15
23 5 7 14 16
4 6 13 20 22
10 12 19 21 3
11 18 25 2 9
```

در قسمت تهیه یک کامپوننت دات نت اطلاعات زیر را در نظر بگیرید :

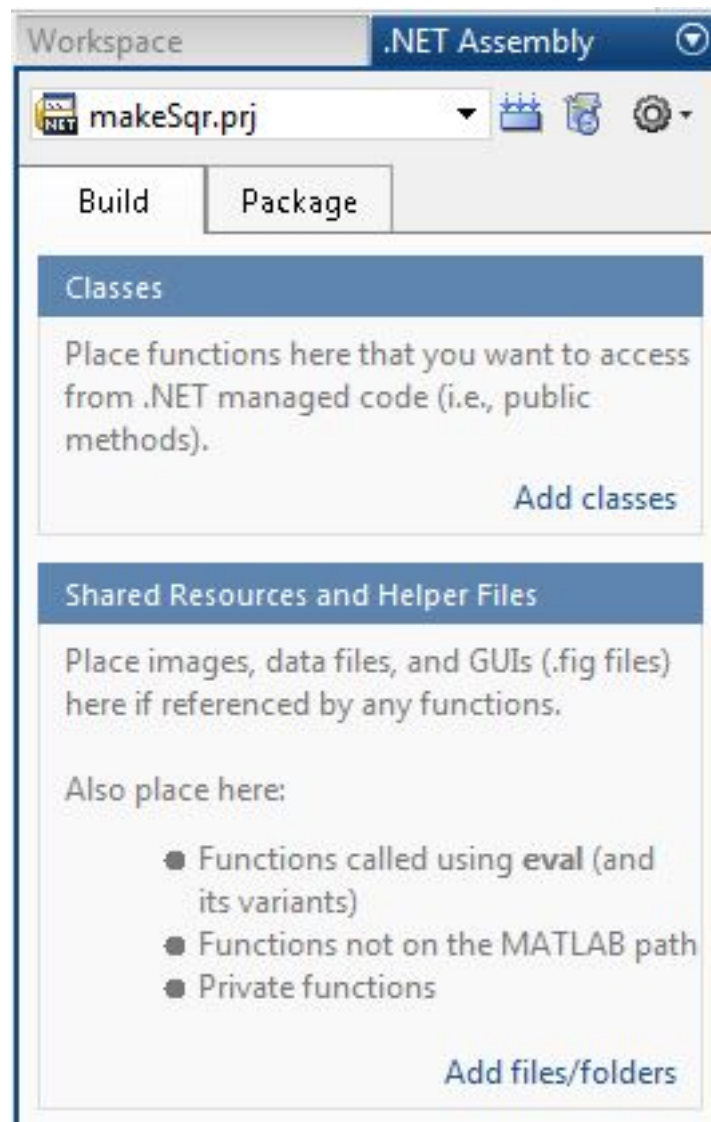
Project Name	makeSqr
Class Name	MLTestClass
File to compile	makesquare.m

سپس برای درست کردن کامپوننت در محیط Deployment Tool GUI برنامه متلب را اجرا کرده و در پنجره command دستور deploytool را اجرا کنید تا پنجره زیر باز شود :

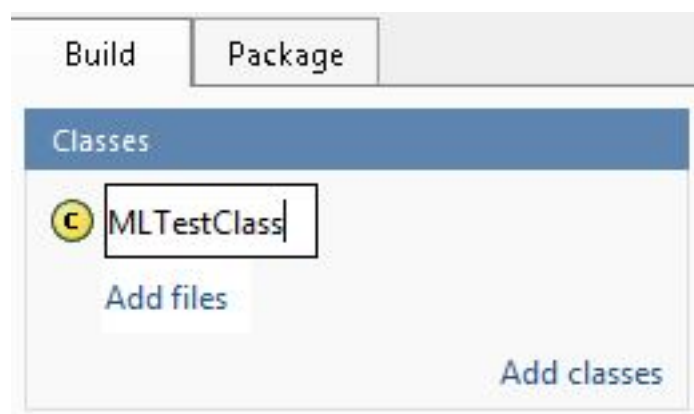
The Deployment Project Dialog Box



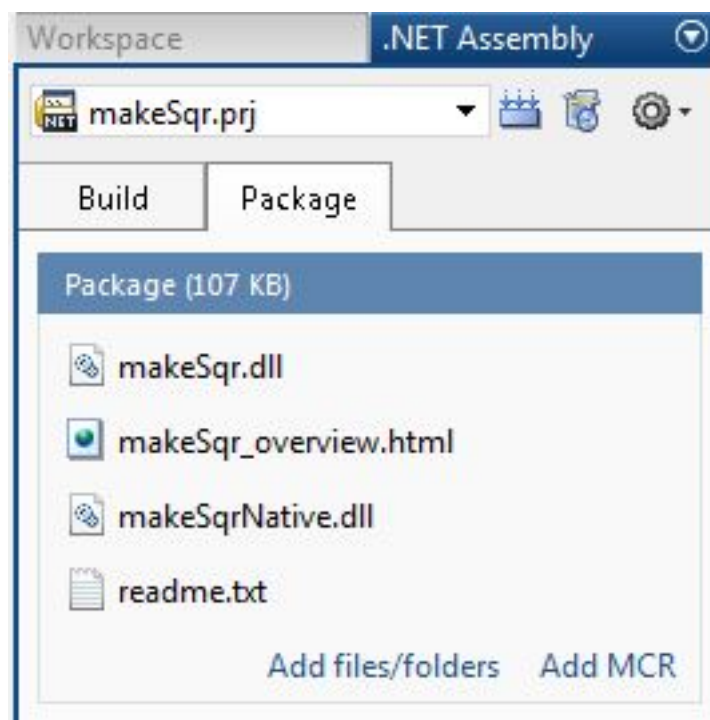
نام و مسیر پروژه را تعیین کنید سپس از منوی کشویی نوع پروژه، که دات نت اسمبلی باشد را انتخاب کنید. پنجره‌ای در به شکل زیر مشاهده خواهد شد :



در تب build اگر قصد استفاده از اپلیکیشن COM را دارید و یا فایل‌هایی جهت تکمیل پروژه قصد پیوست دارید را در قسمت پایین Add files را انتخاب کنید. و اگر قصد استفاده از اپلیکیشن دات نت را دارید قسمت بالایی Add classes را انتخاب کنید و نام کلاس را وارد کنید.

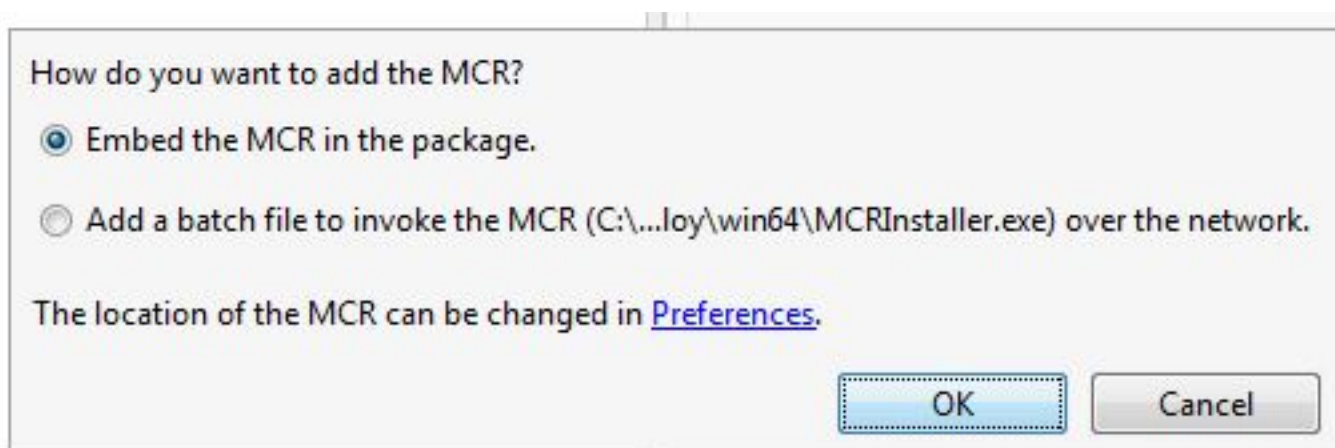


سپس برای کلاس مورد نظر فایل‌های متلبی که قصد کامپایل کردن آنها را دارید از قسمت Add files پیوست کنید. در صورتیکه قصد اضافه کردن کلاس اضافی را داشتید مجدداً مراحل را طی کنید. در انتها دکمه build را زده تا عملیات کامپایل آغاز شود. اما برای استفاده تیم برنامه‌نویسی دات نت احتیاج به کامپایلر متلب می‌باشد که این مهم در پکیجی که به این تیم ارائه خواهد شد مد نظر قرار خواهد گرفت. در قسمت تب Package گزینه Add MCR را انتخاب نمائید :



بعد از انتخاب، دو گزینه برای انتخاب وجود دارد که بطور خلاصه گزینه اول فایل‌های کامپایلر متلب در داخل پروژه جهت ارائه قرار می‌گیرد. همچنین این گزینه جهت استفاده در مواقع درون شبکه‌ای، مواردی که فضای دیسک و عملکرد و چندان اهمیت ندارد مورد استفاده قرار می‌گیرد. اما گزینه دوم عکس قضیه بالا عمل می‌کند و برای تعداد یوزر بالا و شبکه‌ای و ... مورد استفاده می‌باشد.

در اینجا گزینه اول را انتخاب می‌کنیم. در صورتیکه فایل‌های دیگری جهت ضمیمه به پکیج احتیاج است به آن اضافه می‌کنیم.



سپس کلید پکیج را زده تا پکیج مورد نظر آماده شود. دقت داشته باشید که بعد از انتخاب کامپایلر متلب، حجم پکیج نزدیک به 400 مگابایت خواهد شد. پکیج مورد نظر بصورت یک فایل exe فشرده خواهد شد. معمولاً پکیج شامل فایل‌های زیر باید باشد :

componentName.xml	Documentation files
componentName.pdb (if Debug option is selected)	Program Database File, which contains debugging information
componentName.dll	Component assembly file
MCR Installer	MCR Installer (if not already installed on the target machine).

بعد از طی مراحل فوق نوبت به تیم برنامه‌نویسی دات نت می‌رسد. بعد از دریافت پکیج از تیم برنامه‌نویسی متلب در صورتیکه بر روی سیستم هدف کامپایلر متلب و یا خود متلب نصب نیست باید از داخل پکیج این کامپایلر نصب شود. دقت داشته باشید که ورژن کامپایلر بر روی سیستم باید با ورژن پکیج دریافتی یکی باشد. در VS یک پروژه کنسول ایجاد کنید و از فولدر پکیج پروژه دریافتی در زیرفولدر distrib فایل makeSqr.dll را به رفرنس برنامه VS اضافه کنید. در ادامه از مسیر نصب کامپایلر فایل MArray.dll را هم به رفرنس پروژه اضافه کنید. این فایل جهت تبادل داده اپلیکیشن با کامپایلر متلب مورد استفاده قرار می‌گیرد.

installation_folder\toolbox\dotnetbuilder\bin\architecture\framework_version

اسمبلی‌های زیر را به کلاس Program برنامه اضافه کنید :

```
using System;
using MathWorks.MATLAB.NET.Arrays;
using MyComponentName;
```

سپس کدهای زیر را به کلاس فوق اضافه نمایید :

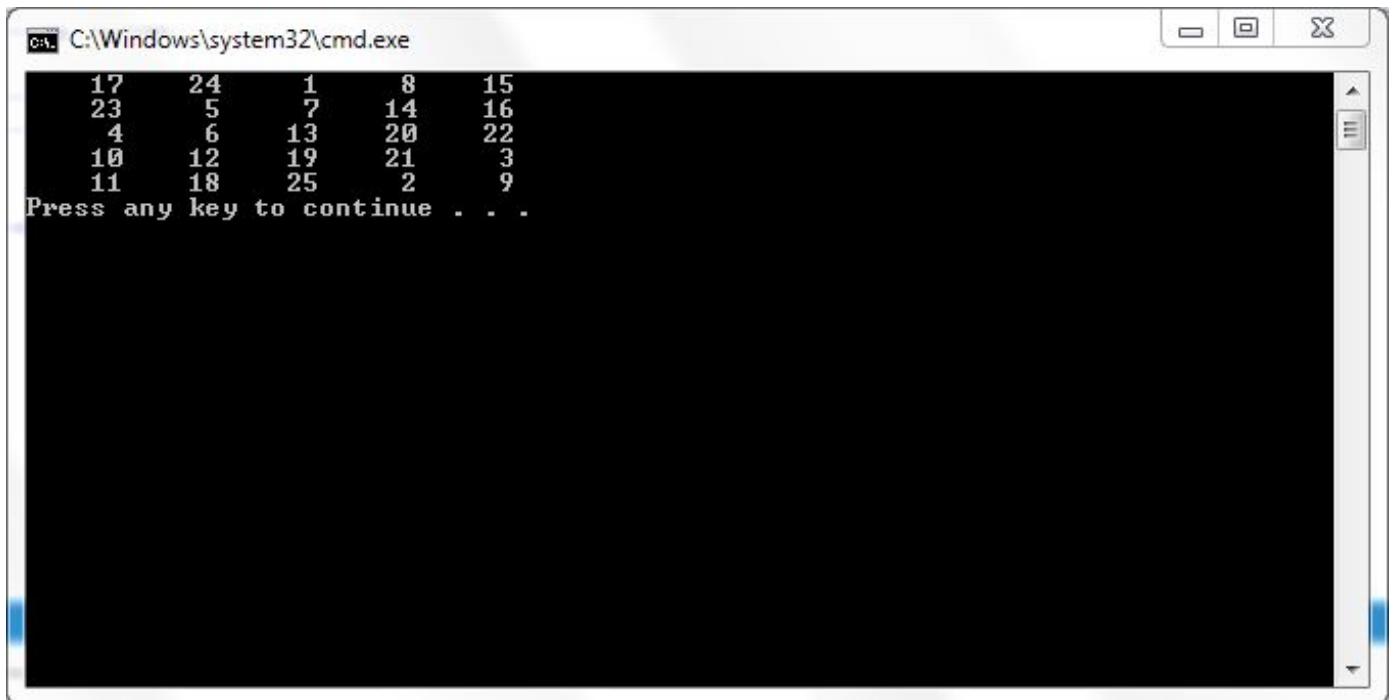
```
static void Main(string[] args)
{
    MLTestClass obj = new MLTestClass();
    MArray[] result = obj.makesquare(1, 5);

    MNumericArray output = (MNumericArray)result[0];
    Console.WriteLine(output);
}
```

توضیحات کدهای فوق :

- 1- MNumericArray یک اینترفیس جهت تعیین و نمایش نوع آرایه‌های عددی در متلب است.
- 2- MArray یک کلاس abstract جهت دسترسی، فرمت‌دهی و مدیریت آرایه‌های متلب می‌باشد.
- 3- عدد 1 مشخص کننده تعداد خروجی تابع متلب و عدد 5 ورودی تابع می‌باشد.

خروجی برنامه همانند خروجی متلب بصورت زیر خواهد بود :



```
C:\Windows\system32\cmd.exe

17    24    1    8    15
23    5    7    14   16
 4    6   13   20   22
10   12   19   21    3
11   18   25    2    9
Press any key to continue . . .
```

نکته:

ورژن فریمورک دات نت در هنگام کامپایل با ورژن Mwarrray.dll باید یکی باشد.

نظرات خوانندگان

نویسنده: سید امیر سجادی
تاریخ: ۱۱:۲۶ ۱۳۹۲/۰۸/۲۸

بسیار عالی بود. ممنون

نویسنده: بهار
تاریخ: ۱۳:۸ ۱۳۹۳/۰۵/۱۲

آموزشتون عالی بود. ممنون

ولی من نمی‌تونم برنامه را اجرا کنم. با پیغام زیر مواجه می‌شم

Could not load file or assembly 'MWMArray, Version=2.11.0.0, Culture=neutral, PublicKeyToken=e1d84a0da19db86f' or one of its dependencies

برنامه را با VS 2010 و متلب R2011a در ویندوز 7 ، 64 بیتی نوشتم. از اینکه راهنمایی می‌کنید سپاسگزارم.

سوال دیگه ای که دارم، اینه که، چه جوری می‌تونم به جای برنامه ماتریس جادویی از شبکه عصبی استفاده کنم؟ آیا تغییر این دو برنامه فقط در تب Bulid است و نیازی به تغییر در تب package ندارد؟

نویسنده: محسن خان
تاریخ: ۱۵:۲۰ ۱۳۹۳/۰۵/۱۲

برنامه‌ی نوشته شده اگر کد خالص دات نتی هست، [مشکلی با 64 بیت و 32 بیت نداره](#) . اما اگر داخلش ناخالصی native وجود داره، مثلا از یک DLL بومی ویندوز استفاده می‌کنه که دات نت نیست و همچنین این DLL از نوع 32 بیتی هست و برنامه روی Any CPU تنظیم شده، حتما کرش می‌کنه با خطایی که گفتید. راه حلش اینه که در خواص پروژه، any CPU را به X86 تغییر بدید.

نویسنده: مسعود مشهدی
تاریخ: ۱۷:۲۳ ۱۳۹۳/۰۵/۱۲

یک نکته دیگه رو هم دقت کنید که ورژن فریمورک دات نت در هنگام کامپایل با ورژن Mwmarray.dll باید یکی باشد. در مورد ماتریس یا شبکه‌های عصبی کلیت تفاوتی نمی‌کند همانطور که این مثال یک تابع ماتریس magic هست شبکه‌های عصبی هم همان تابع هست با این تفاوت که از توابع تو در تو تشکیل شده است.

نویسنده: بهار
تاریخ: ۲۲:۱۷ ۱۳۹۳/۰۵/۱۲

ممنون از اینکه پاسخ دادید. مشکل حل شد. به جای X86 از Any CPU استفاده کردم و برنامه اجرا شد

رشته، مجموعه‌ای از کاراکترهاست که پشت سرهم، در مکانی از حافظه قرار گرفته‌اند. هر کاراکتر حاوی یک شماره سریال در جدول [یونیکد](#) هست. به طور پیش فرض دات نت برای هر کاراکتر (نوع داده char) شانزده بیت در نظر گرفته است که برای 65536 کاراکتر کافی است.

برای نگهداری از رشته‌ها و انجام عملیات بر روی آنها در دات نت از نوع system.string استفاده می‌کنیم:

```
string greeting = "Hello, C#";
```

که در این حالت مجموعه‌ای از کاراکترها را ایجاد خواهد کرد:

H	e	l	l	o	,		C	#
---	---	---	---	---	---	--	---	---

اتفاقاتی که در داخل کلاس string رخ می‌دهد بسیار ساده است و ما را از تعریف char[] بی‌نیاز می‌کند تا مجبور نشویم خانه‌های آرایه را به ترتیب پر کنیم. از معایب استفاده از آرایه char میتوان موارد زیر را برشمرد: خانه‌های آن یک ضرب پر نمیشوند بلکه به ترتیب، خانه به خانه پر می‌شوند. قبل از انتساب متن باید از طول متن مطمئن شویم تا بتوانیم تعداد خانه‌ها را بر اساس آن ایجاد کنیم. همه عملیات آرایه‌ها از پر کردن ابتدای کار گرفته تا هر عملی، نیاز است به صورت دستی صورت بگیرد و تعداد خطوط کد برای هر کاری هم بالا می‌رود.

البته استفاده از string هم راه حل نهایی برای کار با متون نیست. در انتهای این مطلب مورد دیگری را نیز بررسی خواهیم کرد. از ویژگی دیگر رشته‌ها این است که آن‌ها شباهت زیادی به آرایه‌ای از کاراکترها دارند؛ ولی اصلاً شبیه آن‌ها نیستند و نمی‌توانید به صورت یک آرایه آن‌ها را مقداردهی کنید. البته کلاس string امکاناتی را با استفاده از indexer [] مهیا کرده است که می‌توانید بر اساس اندیس‌ها به کاراکترها به صورت جداگانه دسترسی داشته باشید ولی نمی‌توانید آن‌ها را مقدار دهی کنید. این اندیس‌ها از 0 تا طول آن length-1 ادامه دارند.

```
string str = "abcde";
char ch = str[1]; // ch == 'b'
str[1] = 'a'; // Compilation error!
ch = str[50]; // IndexOutOfRangeException
```

همانطور که میدانیم برای مقداردهی رشته‌ها از علامت‌های نقل قول "" استفاده می‌کنیم که باعث میشود اگر بخواهیم علامت " را در رشته‌ها داشته باشیم نتوانیم. برای حل این مشکل از علامت \ استفاده می‌کنیم که البته باعث استفاده از بعضی کاراکترهای خاص دیگر هم می‌شود:

```
string a="Hello \"C#\"";
string b="Hello \r\n C#"; // مساوی با اینتر
string c="C:\\a.jpg"; // چاپ خود علامت \ -مسیردهی
```

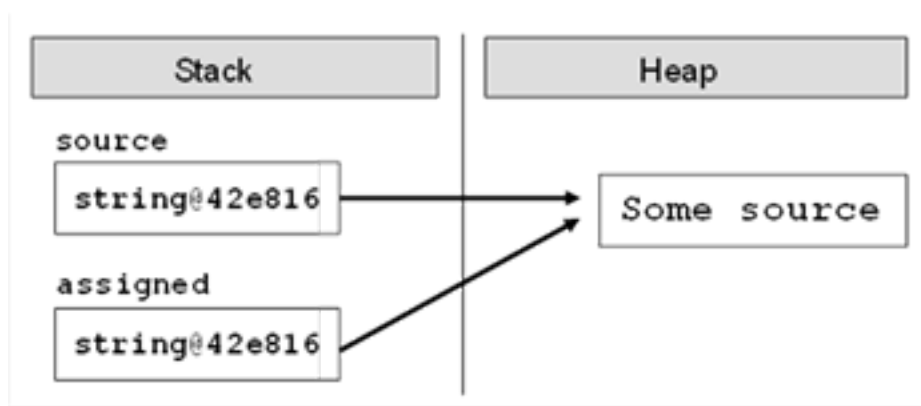
البته اگر از علامت @ در قبل از رشته استفاده شود علامت \ بی اثر خواهد شد.

```
string c=@"C:\a.jpg"; // == "C:\\a.jpg"
```

مقداردهی رشته‌ها و پایدار (تغییر ناپذیر) بودن آنها Immutable

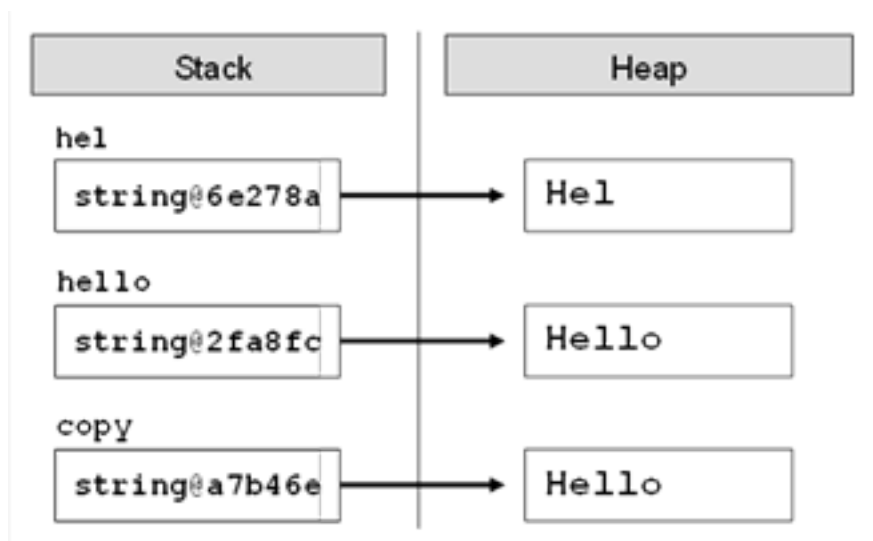
رشته‌ها ساختاری پایدار هستند؛ به این معنی که به صورت reference مقداردهی می‌شوند. موقعی که شما مقداری را به یک رشته انتساب می‌دهید، مقدار متغیر در String pool یا [لینک](#) در Heap ذخیره می‌شوند و اگر همین متغیر را به یک متغیر دیگر انتساب دهیم، متغیر جدید مقدار آن را دیگر در حافظه پویا (داینامیک) Heap به عنوان مقدار جدید ذخیره نخواهد کرد؛ بلکه تنها یک pointer خواهد بود که به آدرس حافظه متغیر اولی اشاره می‌کند. به مثال زیر دقت کنید. متغیر source مقدار some source را ذخیره می‌کند و بعد همین متغیر، به متغیر assigned انتساب داده می‌شود؛ ولی مقداری جابجا نمی‌شود. بلکه متغیر assign به آدرسی در حافظه اشاره می‌کند که متغیر source اشاره می‌کند. هرگاه که در یکی از متغیرها، تغییری رخ دهد، همان متغیری که تغییر کرده است، به آدرس جدید با محتوای تغییر داده شده اشاره می‌کند.

```
string source = "Some source";
string assigned = source;
```

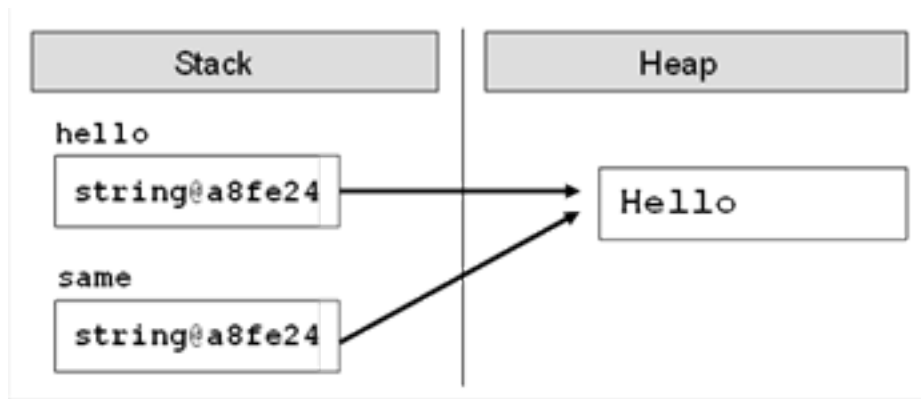


این ویژگی نوع reference فقط برای ساختارهای Immutable به معنی پایدار رخ می‌دهد و نه برای ساختارهای ناپایدار (تغییر پذیر) mutable؛ به این خاطر که آن‌ها مقادیرشان را مستقیماً تغییر می‌دهند و اشاره‌ای در حافظه صورت نمی‌گیرد.

```
string hel = "Hel";
string hello = "Hello";
string copy = hel + "lo";
```




```
string hello = "Hello";
string same = "Hello";
```



برای اطلاعات بیشتر در این زمینه این [لینک](#) را مطالعه نمایید.

مقایسه رشته‌ها

برای مقایسه دو رشته می‌توان از علامت == یا از متد Equals استفاده نماییم. در این حالت به خاطر اینکه کد حروف کوچک و بزرگ متفاوت است، مقایسه حروف هم متفاوت خواهد بود. برای اینکه حروف کوچک و بزرگ تاثیری بر مقایسه ما نگذارند و C# با برابر بدانند باید از متد Equals به شکل زیر استفاده کنیم:

```
Console.WriteLine(word1.Equals(word2,
    StringComparison.CurrentCultureIgnoreCase));
```

برای اینکه بزرگی و کوچکی اعداد را مشخص کنیم از علامت‌های < و > استفاده می‌کنیم ولی برای رشته‌ها از متد CompareTo بهره می‌بریم که چیش قرارگیری آن‌ها را بر اساس حروف الفبا مقایسه می‌کند و سه عدد، می‌تواند خروجی آن باشند. اگر 0 باشد یعنی برابر هستند، اگر 1- باشد رشته اولی قبل از رشته دومی است و اگر 1 باشد رشته دومی قبل از رشته اولی است.

```
string score = "sCore";
string scary = "scary";

Console.WriteLine(score.CompareTo(scary));
Console.WriteLine(scary.CompareTo(score));
Console.WriteLine(scary.CompareTo(scary));

// Console output:
// 1
// -1
// 0
```

اینبار هم برای اینکه حروف کوچک و بزرگ، دخالتی در کار نداشته باشند، می‌توانید از داده شمارشی StringComparison در متد ایستای string.Compare(s1,s2,StringComparison) استفاده نمایید؛ یا از نوع داده‌ای boolean برای تعیین نوع مقایسه استفاده کنید.

```
string alpha = "alpha";
string score1 = "sCorE";
string score2 = "score";

Console.WriteLine(string.Compare(alpha, score1, false));
Console.WriteLine(string.Compare(score1, score2, false));
Console.WriteLine(string.Compare(score1, score2, true));
```

```
Console.WriteLine(string.Compare(score1, score2,
    StringComparison.CurrentCultureIgnoreCase));
// Console output:
// -1
// 1
// 0
// 0
```

نکته : برای مقایسه برابری دو رشته از متد Equals یا == استفاده کنید و فقط برای تعیین کوچک یا بزرگ بودن از compare استفاده نمایید. دلیل آن هم این است که برای مقایسه از فرهنگ culture فعلی سیستم استفاده میشود و نظم جدول یونیکد را رعایت نمی کنند و ممکن است بعضی رشته های نابرابر با یکدیگر برابر باشند. برای مثال در زبان آلمانی دو رشته "SS" و "ß" با یکدیگر برابر هستند.

عبارات با قاعده Regular Expression

این عبارات الگوهایی هستند که قرار است عبارات مشابه الگویی را در رشته ها پیدا کنند. برای مثال الگوی [A-Z0-9]+ مشخص می کند که رشته مورد نظر نباید خالی باشد و حداقل با یکی از حروف بزرگ یا اعداد پر شده باشد. این الگوها میتوانند برای واکنشی داده ها یا قالب های خاص در رشته ها به کار بروند. برای مثال شماره تماس ها، [پست الکترونیکی](#) و ... در [اینجا](#) میتواند نحوه ی الگوسازی را بیاموزید. کد زیر بر اساس یک الگو، شماره تماس های مورد نظر را یافته و البته با فیلتر گذاری آن ها را نمایش می دهد:

```
string doc = "Smith's number: 0898880022\nFranky can be " +
    "found at 0888445566.\nSteven's mobile number: 0887654321";
string replacedDoc = Regex.Replace(
    doc, "(08)[0-9]{8}", "$1*****");
Console.WriteLine(replacedDoc);
// Console output:
// Smith's number: 08*****
// Franky can be found at 08*****.
// Steven' mobile number: 08*****
```

سه شماره تماس در رشته ی بالا با الگوی ما همخوانی دارند که بعد با استفاده از متد replace در شی Regex عبارات دلخواه خودمان را جایگزین شماره تماس ها خواهیم کرد. الگوی بالا شماره تماس هایی را میابد که با 08 آغاز شده اند و بعد از آن 8 عدد دیگر از 0 تا 9 قرار گرفته اند. بعد از اینکه متن مطابق الگو یافت شد، ما آن را با الگوی \$1***** جایگزین می کنیم که علامت \$ یک placeholder برای یک گروه است. هر عبارت () در عبارات با قاعده یک گروه حساب میشود و اولین پرانتز \$1 و دومین پرانتز یا گروه میشود \$2 که در عبارت بالا (08) میشود \$1 و به جای مابقی الگو، 8 علامت ستاره نمایش داده میشود.

اتصال رشته ها در Loop

برای اتصال رشته ها ما از علامت + یا متد ایستای string.concat استفاده می کنیم ولی استفاده ی از آن در داخل یک حلقه باعث کاهش کارایی برنامه خواهد شد. برای همین بیاید ببینم در حین انتقال رشته ها در حافظه چه اتفاقی رخ میدهد. ما در اینجا دو رشته str1 و str2 داریم که عبارات "super" و "star" را نگه داری می کنند و در واقع دو متغیر هستند که به حافظه ی پویای Heap اشاره می کنند. اگر این دو را با هم جمع کنیم و نتیجه را در متغیر result قرار دهیم، سه متغیر میشوند که هر کدام به حافظه ای جداگانه در heap اشاره می کنند. در واقع برای این اتصال، قسمت جدیدی از حافظه تخصیص داده شده و مقدار جدید در آن نشسته است. در این حالت یک متغیر جدید ساخته شد که به آدرس آن اشاره می کند. کل این فرآیند یک فرآیند کاملاً زمانبر است که با تکرار این عمل موجب از دست دادن کارایی برنامه می شود؛ به خصوص اگر در یک حلقه این کار صورت بگیرد. سیستم دات نت همانطور که میدانید شامل [GC](#) یا سیستم خودکار پاکسازی حافظه است که برنامه نویس را از dispose کردن بسیاری از اشیاء بی نیاز می کند. موقعی که متغیری به قسمتی از حافظه اشاره می کند که دیگر بلا استفاده است، سیستم GC به صورت خودکار آنها را پاکسازی می کند که این عمل زمان بر هم خودش موجب کاهش کارایی می شود. همچنین انتقال رشته ها از یک مکان حافظه به مکانی دیگر، باز خودش یک فرآیند زمانبر است؛ به خصوص اگر رشته مورد نظر طولانی هم باشد. **مثال عملی:** در تکه کد زیر قصد داریم اعداد 1 تا 20000 را در یک رشته الحاق کنیم:

```
DateTime dt = DateTime.Now;
string s = "";
for (int index = 1; index <= 20000; index++)
{
    s += index.ToString();
}
Console.WriteLine(s);
```

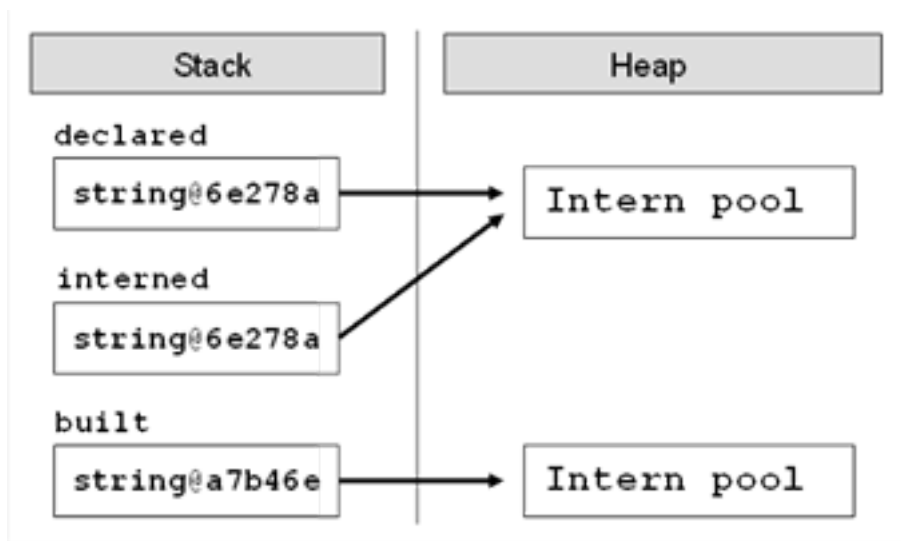
```
Console.WriteLine(dt);
Console.WriteLine(DateTime.Now);
Console.ReadKey();
```

کد بالا تا زمان نمایش کامل، بسته به قدرت سیستم ممکن است یکی دو ثانیه طول بکشد. حالا عدد را به 200000 تغییر دهید (یک صفر اضافه تر). برنامه را اجرا کنید و مجدداً تست بزنید. در این حالت چند دقیقه ای بسته به قدرت سیستم زمان خواهد برد؛ مثلاً دو دقیقه یا سه دقیقه یا کمتر و بیشتر. عملیاتی که در حافظه صورت میگیرد این چند گام را طی میکند: قسمتی از حافظه به طور موقت برای این دور جدید حلقه، گرفته میشود که به آن بافر میگوییم. رشته قبلی به بافر انتقال میابد که بسته به مقدار آن زمان بر و کند است؛ 5 کیلو یا 5 مگابایت یا 50 مگابایت و ... شماره تولید شده جدید به بافر چسبانده میشود. بافر به یک رشته تبدیل میشود و جایی برای خود در حافظه Heap میگیرد. حافظه رشته قدیمی و بافر دیگر بلا استفاده شده اند و توسط GC پاکسازی میشوند که ممکن است عملیاتی زمان بر باشد.

String Builder

این کلاس ناپایدار و تغییر پذیر است. به کد و شکل زیر دقت کنید:

```
string declared = "Intern pool";
string built = new StringBuilder("Intern pool").ToString();
```



این کلاس دیگر مشکل الحاق رشته ها یا دیگر عملیات پردازشی را ندارد. بیایید مثال قبل را برای این کلاس هم بررسی نماییم:

```
StringBuilder sb = new StringBuilder();
sb.Append("Numbers: ");

DateTime dt = DateTime.Now;
for (int index = 1; index <= 200000; index++)
{
    sb.Append(index);
}
Console.WriteLine(sb.ToString());
Console.WriteLine(dt);
Console.WriteLine(DateTime.Now);
Console.ReadKey();
```

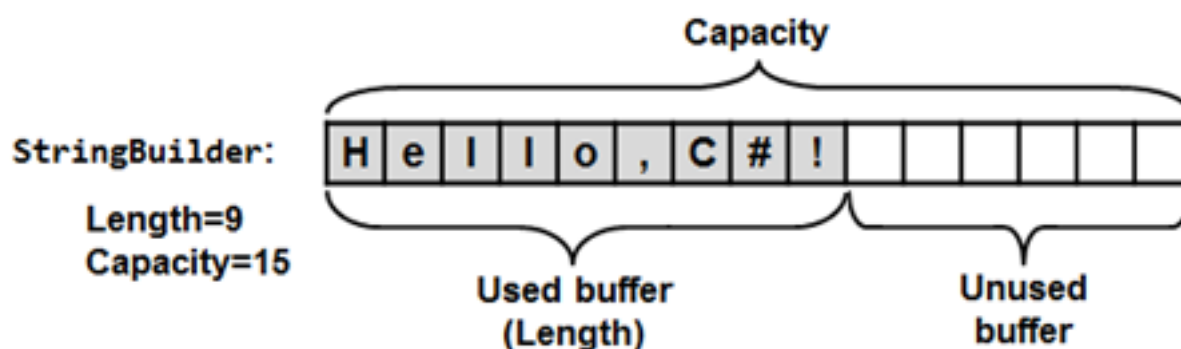
اکنون همین عملیات چند دقیقه‌ای قبل، در زمانی کمتر، مثلاً دو ثانیه انجام میشود. حال این سوال پیش می‌آید مگر کلاس *stringbuilder* چه میکند که زمان پردازش آن قدر کوتاه است؟

همانطور که گفتیم این کلاس *mutable* یا تغییر پذیر است و برای انجام عملیات‌های ویرایشی نیازی به ایجاد شیء جدید در حافظه ندارد؛ در نتیجه باعث کاهش انتقال غیرضروری داده‌ها برای عملیات پایه‌ای چون الحاق رشته‌ها میگردد.

stringbuilder شامل یک بافر با ظرفیتی مشخص است (به طور پیش فرض 16 کاراکتر). این کلاس آرایه‌هایی از کاراکترها را پیاده سازی میکند که برای عملیات و پردازش‌هایش از یک رابط کاربرپسند برای برنامه نویسان استفاده می‌کند. اگر تعداد کاراکترها کمتر از 16 باشد مثلاً 5، فقط 5 خانه آرایه استفاده میشود و مابقی خانه‌ها خالی میماند و با اضافه شدن یک کاراکتر جدید، دیگر شیء جدیدی در حافظه درست نمی‌شود؛ بلکه در خانه ششم قرار می‌گیرد و اگر تعداد کاراکترهایی که اضافه می‌شوند باعث شود از 16 کاراکتر رد شود، مقدار خانه‌ها دو برابر میشوند؛ هر چند این عملیات دو برابر شدن *resizing* عملیاتی کند است ولی این اتفاق به ندرت رخ می‌دهد.

کد زیر یک آرایه 15 کاراکتری ایجاد می‌کند و عبارت *Hello C#* را در آن قرار می‌دهد.

```
StringBuilder sb = new StringBuilder(15);
sb.Append("Hello, C#!");
```



در شکل بالا خانه‌هایی خالی مانده است *Unused* و جا برای کاراکترهای جدید به اندازه خانه‌های *unused* هست و اگر بیشتر شود همانطور که گفتیم تعداد خانه‌ها 2 برابر می‌شوند که در اینجا میشود 30.

استفاده از متد ایستای *string.Format*

از این متد برای نوشتن یک متن به صورت قالب و سپس جایگزینی مقادیر استفاده می‌شود:

```
DateTime date = DateTime.Now;
string name = "David Scott";
string task = "Introduction to C# book";
string location = "his office";

string formattedText = String.Format(
    "Today is {0:MM/dd/yyyy} and {1} is working on {2} in {3}.",
    date, name, task, location);
Console.WriteLine(formattedText);
```

در کد بالا ابتدا ساختار قرار گرفتن تاریخ را بر اساس الگو بین *{}* مشخص می‌کنیم و متغیر *date* در آن قرار می‌گیرد و سپس برای *{1}*, *{2}*, *{3}* به ترتیب قرار گیری آن‌ها متغیرهای *name*, *last*, *location* قرار می‌گیرند. از *ToString()* هم می‌توان برای فرمت بندی خروجی استفاده کرد؛ مثل همین عبارت *MM/dd/yyyy* در خروجی نوع داده تاریخ و زمان.

نظرات خوانندگان

نویسنده: شهرز جعفری
تاریخ: ۱۸:۱۰ ۱۳۹۳/۱۱/۲۹

یک سوال منظور از Gac اینجا چیه؟

نویسنده: علی یگانه مقدم
تاریخ: ۱۸:۴۸ ۱۳۹۳/۱۱/۲۹

ممنون که گوشزد کردید؛ عذر میخوام. مطلب ویرایش شد. منظور GC بود. بنده اشتباهها نوشتم GAC.

استثناء چیست؟

واژه‌ی استثناء یا exception کوتاه شده‌ی عبارت exceptional event است. در واقع exception یک نوع رویداد است که در طول اجرای برنامه رخ می‌دهد و در نتیجه، جریان عادی برنامه را مختل می‌کند. زمانیکه خطایی درون یک متد رخ دهد، یک شیء (exception object) حاوی اطلاعاتی درباره‌ی خطا ایجاد خواهد شد. به فرآیند ایجاد یک exception object و تحویل دادن آن به سیستم runtime، اصطلاحاً throwing an exception یا صدور استثناء گفته می‌شود که در ادامه به آن خواهیم پرداخت. بعد از اینکه یک متد استثنائی را صادر می‌کند، سیستم runtime سعی در یافتن روشی برای مدیریت آن خواهد کرد. خوب اکنون که با مفهوم استثناء آشنا شدید اجازه دهید دو سناریو را با هم بررسی کنیم.

- سناریوی اول:

فرض کنید یک فایل XML از پیش تعریف شده (برای مثال یک لیست از محصولات) قرار است در کنار برنامه‌ی شما باشد و باید این لیست را درون برنامه‌ی خود نمایش دهید. در این حالت برای خواندن این فایل انتظار دارید که فایل وجود داشته باشد. اگر این فایل وجود نداشته باشد برنامه‌ی شما با اشکال روبرو خواهد شد.

- سناریوی دوم:

فرض کنید یک فایل XML از آخرین محصولات مشاهده شده توسط کاربران را به صورت cache در برنامه‌تان دارید. در این حالت در اولین بار اجرای برنامه توسط کاربر انتظار داریم که این فایل موجود نباشد و اگر فایل وجود نداشته باشد به سادگی می‌توانیم فایل مربوط را ایجاد کرده و محصولات را که توسط کاربر مشاهده شده، درون این فایل اضافه کنیم. در واقع استثناءها بستگی به حالت‌های مختلفی دارد. در مثال اول وجود فایل حیاتی است ولی در حالت دوم وجود فایل نیز برنامه می‌تواند به کار خود ادامه داده و فایل مورد نظر را از نو ایجاد کند. استثناءها مربوط به زمانی هستند که این احتمال وجود داشته باشد که برنامه طبق انتظار پیش نرود. برای حالت اول کد زیر را داریم:

```
public IEnumerable<Product> GetProducts()
{
    using (var stream = File.Read(Path.Combine(Environment.CurrentDirectory, "products.xml")))
    {
        var serializer = new XmlSerializer();
        return (IEnumerable<Product>)serializer.Deserialize(stream);
    }
}
```

همانطور که عنوان شد در حالت اول انتظار داریم که فایلی بر روی دیسک موجود باشد. در نتیجه نیازی نیست هیچ استثنایی را مدیریت کنیم (زیرا در واقع اگر فایل موجود نباشد هیچ روشی برای ایجاد آن نداریم). در مثال دوم می‌دانیم که ممکن است فایل از قبل موجود نباشد. بنابراین می‌توانیم موجود بودن فایل را با یک شرط بررسی کنیم:

```
public IEnumerable<Product> GetCachedProducts()
{
    var fullPath = Path.Combine(Environment.CurrentDirectory, "ProductCache.xml");
    if (!File.Exists(fullPath))
        return new Product[0];

    using (var stream = File.Read(fullPath))
    {
        var serializer = new XmlSerializer();
        return (IEnumerable<Product>)serializer.Deserialize(stream);
    }
}
```

چه زمانی باید استثناءها را مدیریت کنیم؟

زمانیکه بتوان متدهایی که خروجی مورد انتظار را بر می‌گردانند ایجاد کرد. اجازه دهید دوباره از مثال‌های فوق استفاده کنیم:

```
IEnumerable<Product> GetProducts()
```

همانطور که از نام آن پیداست این متد باید همیشه لیستی از محصولات را برگرداند. اگر می‌توانید اینکار را با استفاده از catch کردن یک استثنا انجام دهید در غیر اینصورت نباید درون متد اینکار را انجام داد.

```
IEnumerable<Product> GetCachedProducts()
```

در متد فوق می‌توانستیم از FileNotFoundException برای فایل موردنظر استفاده کنیم؛ اما مطمئن بودیم که فایل در ابتدا وجود ندارد.

در واقع استثناها حالت‌هایی هستند که غیرقابل پیش‌بینی هستند. این حالت‌ها می‌توانند یک خطای منطقی از طرف برنامه‌نویس و یا چیزی خارج کنترل برنامه‌نویس باشند (مانند خطاهای سیستم‌عامل، شبکه، دیسک). یعنی در بیشتر مواقع این نوع خطاها را نمی‌توان مدیریت کرد.

اگر می‌خواهید استثناءها را catch کرده و آنها را لاگ کنید در بالاترین لایه اینکار را انجام دهید.

چه استثناءهایی باید مدیریت شوند و کدام‌ها خیر؟

مدیریت صحیح استثناءها می‌تواند خیلی مفید باشد. همانطور که عنوان شد یک استثناء زمانی رخ می‌دهد که یک حالت استثناء در برنامه اتفاق بیفتد. این مورد را بخاطر داشته باشید، زیرا به شما یادآوری می‌کند که در همه جا نیازی به استفاده از try/catch نیست. در اینجا ذکر این نکته خیلی مهم است: تنها استثناءهایی را catch کنید که بتوانید برای آن راه‌حلی ارائه دهید. به عنوان مثال اگر در لایه‌ی دسترسی به داده، خطایی رخ دهد و استثنای SQLException صادر شود، می‌توانیم آن را catch کرده و درون یک استثناء عمومی‌تر قرار دهیم:

```
public class UserRepository : IUserRepository
{
    public IList<User> Search(string value)
    {
        try
        {
            return CreateConnectionAndACommandAndReturnAList("WHERE value=@value",
Parameter.New("value", value));
        }
        catch (SQLException err)
        {
            var msg = String.Format("Ohh no! Failed to search after users with '{0}' as search
string", value);
            throw new DataSourceException(msg, err);
        }
    }
}
```

همانطور که در کد فوق مشاهده می‌کنید به محض صدور استثنای SQLException آن را درون قسمت catch به صورت یک استثنای عمومی‌تر همراه با افزودن یک سری اطلاعات جدید صادر می‌کنیم. اما همانطور که عنوان شد کار لاگ کردن استثناءها را بهتر است در لایه‌های بالاتر انجام دهیم. اگر مطمئن نیستید که تمام استثناءها توسط شما مدیریت شده‌اند، می‌توانید در حالت‌های زیر، دیگر استثناءها را مدیریت کنید: ASP.NET می‌توانید Application_Error را پیاده‌سازی کنید. در اینجا فرصت خواهید داشت تا تمامی خطاهای مدیریت نشده را هندل کنید.

WinForms: استفاده از رویدادهای Application.ThreadException و AppDomain.CurrentDomain.UnhandledException

WCF: پیاده‌سازی اینترفیس IErrorHandler

ASMX: ایجاد یک Soap Extension سفارشی

[ASP.NET WebAPI](#)

چه زمان‌هایی باید یک استثناء صادر شود؟

صادر کردن یک استثناء به تنهایی کار ساده‌ایی است. تنها کافی است throw را همراه شیء exception (exception object) فراخوانی کنیم. اما سوال اینجاست که چه زمانی باید یک استثناء را صادر کنیم؟ چه داده‌هایی را باید به استثناء اضافه کنیم؟ در ادامه به این سوالات خواهیم پرداخت. همانطور که عنوان گردید استثناءها زمانی باید صادر شوند که یک استثناء اتفاق بیفتد.

اعتبارسنجی آرگومان‌ها

ساده‌ترین مثال، آرگومان‌های مورد انتظار یک متد است:

```
public void PrintName(string name)
{
    Console.WriteLine(name);
}
```

در حالت فوق انتظار داریم مقداری برای پارامتر name تعیین شود. متد فوق با آرگومان null نیز به خوبی کار خواهد کرد؛ یعنی مقدار خروجی یک خط خالی خواهد بود. از لحاظ کدنویسی متد فوق به خوبی کار خود را انجام می‌دهد اما خروجی مورد انتظار کاربر نمایش داده نمی‌شود. در این حالت نمی‌توانیم تشخیص دهیم مشکل از کجا ناشی می‌شود. مشکل فوق را می‌توانیم با صدور استثنای ArgumentNullException رفع کنیم:

```
public void PrintName(string name)
{
    if (name == null) throw new ArgumentNullException("name");
    Console.WriteLine(name);
}
```

خوب، name باید دارای طول ثابت و همچنین ممکن است حاوی عدد و حروف باشد:

```
public void PrintName(string name)
{
    if (name == null) throw new ArgumentNullException("name");
    if (name.Length < 5 || name.Length > 10) throw new ArgumentOutOfRangeException("name", name, "Name must be between 5 or 10 characters long");
    if (name.Any(x => !char.IsAlphaNumeric(x)) throw new ArgumentOutOfRangeException("name", name, "May only contain alpha numerics");
    Console.WriteLine(name);
}
```

برای حالت فوق و همچنین جلوگیری از تکرار کدهای داخل متد PrintName می‌توانید یک متد Validator برای کلاسی با نام Person ایجاد کنید.

حالت دیگر صدور استثناء، زمانی است که متدی خروجی مورد انتظارمان را نتواند تحویل دهد. یک مثال بحث‌برانگیز متدی با امضای زیر است:

```
public User GetUser(int id)
{
}
```

کاملاً مشخص است که متدی همانند متد فوق زمانیکه کاربری را پیدا نکند، مقدار null را برمی‌گرداند. اما این روش درستی است؟ خیر؛ زیرا همانطور که از نام این متد پیداست باید یک کاربر به عنوان خروجی برگردانده شود. با استفاده از بررسی null کدهایی شبیه به این را در همه جا خواهیم داشت:

```
var user = datasource.GetUser(userId);
if (user == null)
    throw new InvalidOperationException("Failed to find user: " + userId);
// actual logic here
```


به این چنین کدهایی معمولاً The null cancer گفته می‌شود (سرطان نال!) زیرا اجازه داده‌ایم متد، خروجی null را بازگشت دهد. به جای کد فوق می‌توانیم از این روش استفاده کنیم:

```
public User GetUser(int id)
{
    if (id <= 0) throw new ArgumentOutOfRangeException("id", id, "Valid ids are from 1 and above. Do you have a parsing error somewhere?");

    var user = db.Execute<User>("WHERE Id = ?", id);
    if (user == null)
        throw new EntityNotFoundException("Failed to find user with id " + id);

    return user;
}
```

نکته‌ای که باید به آن توجه کنید این است که در هنگام صدور یک استثناء اطلاعات کافی را نیز به آن پاس دهید. به عنوان مثال در EntityNotFoundException مثال فوق پاس دادن "Failed to find user with id " + id کار دیباگ را برای مصرف کننده، راحت‌تر خواهد کرد.

خطاهای متداول حین کار با استثناءها

صدور مجدد استثناء و از بین بردن stacktrace

کد زیر را در نظر بگیرید:

```
try
{
    FutileAttemptToResist();
}
catch (BorgException err)
{
    _myDearLog.Error("I'm in da cube! Ohh no!", err);
    throw err;
}
```

مشکل کد فوق قسمت throw err است. این خط کد، محتویات stacktrace را از بین برده و استثناء را مجدداً برای شما ایجاد خواهد کرد. در این حالت هرگز نمی‌توانیم تشخیص دهیم که منبع خطا از کجا آمده است. در این حالت پیشنهاد می‌شود که تنها از throw استفاده شود. در این حالت استثناء اصلی مجدداً صادر گردیده و مانع حذف شدن محتویات stacktrace خواهد شد ([+](#)). اضافه نکردن اطلاعات استثناء اصلی به استثناء جدید

یکی دیگر از خطاهای رایج اضافه نکردن استثناء اصلی حین صدور استثناء جدید است:

```
try
{
    GreaseTinMan();
}
catch (InvalidOperationException err)
{
    throw new TooScaredLion("The Lion was not in the m00d", err); //<---- استثناء به استثناء
    جدید پاس داده شود
}
```

ارائه ندادن context information

در هنگام صدور یک استثناء بهتر است اطلاعات دقیقی را به آن ارسال کنیم تا دیباگ کردن آن به راحتی انجام شود. به عنوان مثال کد زیر را در نظر داشته باشید:

```
try
{
```

```

    socket.Connect("somethingawful.com", 80);
}
catch (SocketException err)
{
    throw new InvalidOperationException("Socket failed", err);
}

```

هنگامی که کد فوق با خطا مواجه شود نمی‌توان تنها با متن Socket failed تشخیص داد که مشکل از چه چیزی است. بنابراین پیشنهاد می‌شود اطلاعات کامل و در صورت امکان به صورت دقیق را به استثناء ارسال کنید. به عنوان مثال در کد زیر سعی شده است تا حد امکان context information کاملی برای استثناء ارائه شود:

```

void IncreaseStatusForUser(int userId, int newStatus)
{
    try
    {
        var user = _repository.Get(userId);
        if (user == null)
            throw new UpdateException(string.Format("Failed to find user #{0} when trying to increase status to {1}", userId, newStatus));

        user.Status = newStatus;
        _repository.Save(user);
    }
    catch (DataSourceException err)
    {
        var errMsg = string.Format("Failed to find modify user #{0} when trying to increase status to {1}", userId, newStatus);
        throw new UpdateException(errMsg, err);
    }
}

```

نحوه‌ی طراحی استثناءها

برای ایجاد یک استثناء سفارشی می‌توانید از کلاس Exception ارث‌بری کنید و چهار سازنده‌ی آن را اضافه کنید:

```

public NewException()
public NewException(string description )
public NewException(string description, Exception inner)
protected or private NewException(SerializationInfo info, StreamingContext context)

```

سازنده اول به عنوان default constructor شناخته می‌شود. اما پیشنهاد می‌شود که از آن استفاده نکنید، زیرا یک استثناء بدون context information از ارزش کمی برخوردار خواهد بود.

سازنده‌ی دوم برای تعیین description بوده و همانطور که عنوان شد ارائه دادن context information از اهمیت بالایی برخوردار است. به عنوان مثال فرض کنید استثناء KeyNotFoundException که توسط کلاس Dictionary صادر شده است را دریافت کرده‌اید. این استثناء زمانی صادر خواهد شد که بخواهید به عنصری که درون دیکشنری پیدا نشده است دسترسی داشته باشید. در این حالت پیام زیر را دریافت خواهید کرد:

```
"The given key was not present in the dictionary."
```

حالا فرض کنید اگر پیام به صورت زیر باشد چقدر باعث خوانایی و عیب‌یابی ساده‌تر خطا خواهد شد:

```
"The key 'abracadabra' was not present in the dictionary."
```

در نتیجه تا حد امکان سعی کنید که context information شما کاملتر باشد.

سازنده‌ی سوم شبیه به سازنده‌ی قبلی عمل می‌کند با این تفاوت که توسط پارامتر دوم می‌توانیم یک استثناء دیگر را catch کرده یک استثناء جدید صادر کنیم.

سازنده‌ی سوم زمانی مورد استفاده قرار می‌گیرد که بخواهید از Serialization پشتیبانی کنید (به عنوان مثال ذخیره‌ی استثناءها درون فایل و...) در این حالت:

خوب، برای یک استثناء سفارشی حداقل باید کدهای زیر را داشته باشیم:

```
public class SampleException : Exception
{
    public SampleException(string description)
        : base(description)
    {
        if (description == null) throw new ArgumentNullException("description");
    }

    public SampleException(string description, Exception inner)
        : base(description, inner)
    {
        if (description == null) throw new ArgumentNullException("description");
        if (inner == null) throw new ArgumentNullException("inner");
    }

    public SampleException(SerializationInfo info, StreamingContext context)
        : base(info, context)
    {
    }
}
```

اجباری کردن ارائه‌ی Context information:

برای اجباری کردن context information کافی است یک فیلد اجباری درون سازنده تعریف کنیم. برای مثال اگر بخواهیم کاربر HTTP status code را برای استثناء ارائه دهد باید سازنده‌ها را اینگونه تعریف کنیم:

```
public class HttpException : Exception
{
    System.Net.HttpStatusCode _statusCode;

    public HttpException(System.Net.HttpStatusCode statusCode, string description)
        : base(description)
    {
        if (description == null) throw new ArgumentNullException("description");
        _statusCode = statusCode;
    }

    public HttpException(System.Net.HttpStatusCode statusCode, string description, Exception inner)
        : base(description, inner)
    {
        if (description == null) throw new ArgumentNullException("description");
        if (inner == null) throw new ArgumentNullException("inner");
        _statusCode = statusCode;
    }

    public HttpException(SerializationInfo info, StreamingContext context)
        : base(info, context)
    {
    }

    public System.Net.HttpStatusCode StatusCode { get; private set; }
}
```

همچنین بهتر است پراپرتی Message را برای نمایش پیام مناسب بازنویسی کنید:

```
public override string Message
{
    get { return base.Message + "\r\nStatus code: " + StatusCode; }
}
```

مورد دیگری که باید در کد فوق مد نظر داشت این است که status code قابلیت سریالایز شدن را ندارد. بنابراین باید متد GetObjectData را برای سریالایز کردن بازنویسی کنیم:

```
public class HttpException : Exception
{
    // [...]

    public HttpException(SerializationInfo info, StreamingContext context)
        : base(info, context)
    {

```

```
// this is new
StatusCode = (HttpStatusCode) info.GetInt32("HttpStatusCode");
}

public HttpStatusCode StatusCode { get; private set; }

public override string Message
{
    get { return base.Message + "\r\nStatus code: " + StatusCode; }
}

// this is new
public override void GetObjectData(SerializationInfo info, StreamingContext context)
{
    base.GetObjectData(info, context);
    info.AddValue("HttpStatusCode", (int) StatusCode);
}
}
```

در اینحالت فیلدهای اضافی در طول فرآیند Serialization به خوبی سریالایز خواهند شد.

در حین صدور استثناءها همیشه باید در نظر داشته باشیم که چه نوع context information را می‌توان ارائه داد، این مورد در یافتن راه‌حل خیلی کمک خواهد کرد.

طراحی پیام‌های مناسب

پیام‌های exception مختص به توسعه‌دهندگان است نه کاربران نهایی.

نوشتن این نوع پیام‌ها برای برنامه‌نویس کار خسته‌کننده‌ای است. برای مثال دو مورد زیر را در نظر داشته باشید:

```
throw new Exception("Unknown FaileType");
throw new Exception("Unecpected workingDirectory");
```

این نوع پیام‌ها حتی اگر از لحاظ نوشتاری مشکلی نداشته باشند یافتن راه‌حل را خیلی سخت خواهند کرد. اگر در زمان برنامه‌نویسی با این نوع خطاها روبرو شوید ممکن است با استفاده از debugger ورودی نامعتبر را پیدا کنید. اما در یک برنامه و خارج از محیط برنامه‌نویسی، یافتن علت بروز خطا خیلی سخت خواهد بود. توسعه‌دهندگانی که exception message را در اولویت قرار می‌دهند، معتقد هستند که از لحاظ تجربه‌ی کاربری پیام‌ها تا حد امکان باید فاقد اطلاعات فنی باشد. همچنین همانطور که پیش‌تر عنوان گردید این نوع پیام‌ها همیشه باید در بالاترین سطح نمایش داده شوند نه در لایه‌های زیرین. همچنین پیام‌هایی مانند Unknown FaileType نه برای کاربر نهایی، بلکه برای برنامه‌نویس نیز ارزش چندانی ندارد زیرا فاقد اطلاعات کافی برای یافتن مشکل است. در طراحی پیام‌ها باید موارد زیر را در نظر داشته باشیم:

- امنیت:

یکی از مواردی که از اهمیت بالایی برخوردار است مسئله امنیت است از این جهت که پیام‌ها باید فاقد مقادیر runtime باشند. زیرا ممکن است اطلاعاتی را در خصوص نحوه‌ی عملکرد سیستم آشکار سازند.

- زبان:

همانطور که عنوان گردید پیام‌های استثناء برای کاربران نهایی نیستند، زیرا کاربران نهایی ممکن است اشخاص فنی نباشند، یا ممکن است زبان آنها انگلیسی نباشد. اگر مخاطبین شما آلمانی باشند چطور؟ آیا تمامی پیام‌ها را با زبان آلمانی خواهید نوشت؟ اگر هم اینکار را انجام دهید تکلیف استثناءهایی که توسط Base Class Library و دیگر کتابخانه‌های third-party صادر می‌شوند چیست؟ اینها انگلیسی هستند.

در تمامی حالت‌هایی که عنوان شد فرض بر این است که شما در حال نوشتن این نوع پیام‌ها برای یک سیستم خاص هستید. اما اگر هدف نوشتن یک کتابخانه باشد چطور؟ در این حالت نمی‌دانید که کتابخانه‌ی شما در کجا استفاده می‌شود. اگر هدف نوشتن یک کتابخانه نباشد این نوع پیام‌هایی که برای کاربران نهایی باشند، وابستگی‌ها را در سیستم افزایش خواهند داد، زیرا در این حالت پیام‌ها به یک رابط کاربری خاص گره خواهند خورد.

خب اگر پیام‌ها برای کاربران نهایی نیستند، پس برای کسانی مورد استفاده قرار خواهند گرفت؟ در واقع این نوع پیام می‌تواند به

عنوان یک documentation برای سیستم شما باشند.

فرض کنید در حال استفاده از یک کتابخانه جدید هستید به نظر شما کدام یک از پیام‌های زیر مناسب هستند:

```
"Unexpected workingDirectory"
```

یا:

```
"You tried to provide a working directory string that doesn't represent a working directory. It's not your fault, because it wasn't possible to design the FileStore class in such a way that this is a statically typed pre-condition, but please supply a valid path to an existing directory."
```

```
"The invalid value was: "fllobdedy"."
```

یافتن مشکل در پیام اول خیلی سخت خواهد بود زیرا فاقد اطلاعات کافی برای یافتن مشکل است. اما پیام دوم مشکل را به صورت کامل توضیح داده است. در حالت اول شما قطعاً نیاز خواهید داشت تا از دیباگر برای یافتن مشکل استفاده کنید. اما در حالت دوم پیام به خوبی شما را برای یافتن راه‌حل راهنمایی می‌کند. همیشه برای نوشتن پیام‌های مناسب سعی کنید از لحاظ نوشتاری متن شما مشکلی نداشته باشد، اطلاعات کافی را درون پیام اضافه کنید و تا حد امکان نحوه‌ی رفع مشکل را توضیح دهید