عنوان: **مدیریت تغییرات در سیستم های مبتنی بر WCF**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۳/۰۲ ۱۵:۲۰
آدرس: www.dotnettips.info
برچسب‌ها: WCF, versioning

تشریح مسئله : در صورتی که بعد از انتشار برنامه؛ در نسخه بعدی مدل سمت سرور تغییر کرده باشد و امکان بروز رسانی مدل‌های سمت کلاینت وجود نداشته باشد برای حل این مسئله بهترین روش کدام است.
نکته : برای فهم بهتر مطالب آشنایی اولیه با مفاهیم WCF الزامی است.
ابتدا مدل زیر را در نظر بگیرید:

```
[DataContract]
    public class Book
    {
        [DataMember]
        public int Code { get; set; }

        [DataMember]
        public string Name { get; set; }
    }
```

حالا یک سرویس برای دریافت و ارسال اطلاعات این مدل به کلاینت می‌نویسیم.

```
[ServiceContract]
    public interface ISampleService
    {
        [OperationContract]
        IEnumerable<Book> GetAll();

        [OperationContract]
        void Save( Book book );
    }
```

و سرویسی که Contract بالا رو پیاده سازی کند.

```
public class SampleService : ISampleService
    {
        public List<Book> ListOfBook
        {
            get;
            private set;
        }

        public SampleService()
        {
            ListOfBook = new List<Book>();
        }
        public IEnumerable<Book> GetAll()
        {
            ListOfBook.AddRange( new Book[]
            {
                new Book(){Code=1 , Name="Book1"},
                new Book(){Code=2 , Name="Book2"},
            } );
            return ListOfBook;
        }

        public void Save( Book book )
        {
            ListOfBook.Add( book );
        }
    }
```

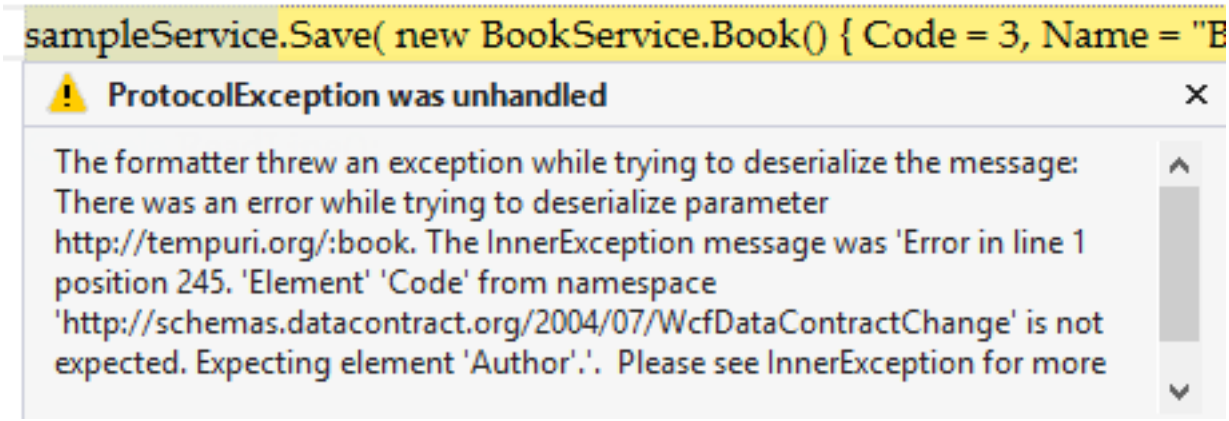
متد GetAll برای ارسال اطلاعات به کلاینت و متد Save نیز برای دریافت اطلاعات از کلاینت.
حالا یک پروژه Console Application بسازید و از روش AddServiceReference سرویس مورد نظر را به Client اضافه کنید.
برنامه را تست کنید. بدون هیچ مشکلی کار می‌کند.

حالا اگر در نسخه بعدی سیستم مجبور شویم به مدل Book یک خاصیت دیگر به نام Author را نیز اضافه کنیم و امکان Update کردن سرویس در سمت کلاینت وجود نداشته باشد چه اتفاقی خواهد افتاد.
به صورت زیر:

```
[DataContract]
    public class Book
    {
        [DataMember]
        public int Code { get; set; }
        [DataMember]
        public string Name { get; set; }

        [DataMember]
        public string Author { get; set; }
    }
```

به طور پیش فرض اگر در DataContractهای سمت سرور و کلاینت اختلاف وجود داشته باشد این موارد نادیده گرفته می‌شوند.
یعنی همیشه مقدار خاصیت Author برابر null خواهد بود.
نکته : برای Value Typeها مقادیر پیش فرض و برای Reference Typeها مقدار Null.
اگر برای DataMemberAttribute خاصیت IsRequired را برابر true کنیم از این پس برای هر درخواستی که مقدار Author آن مقدار نداشته باشد یک Protocol Exception پرتاب می‌شود. به صورت زیر:

```
[DataMember( IsRequired = true )]
public string Author { get; set; }
```

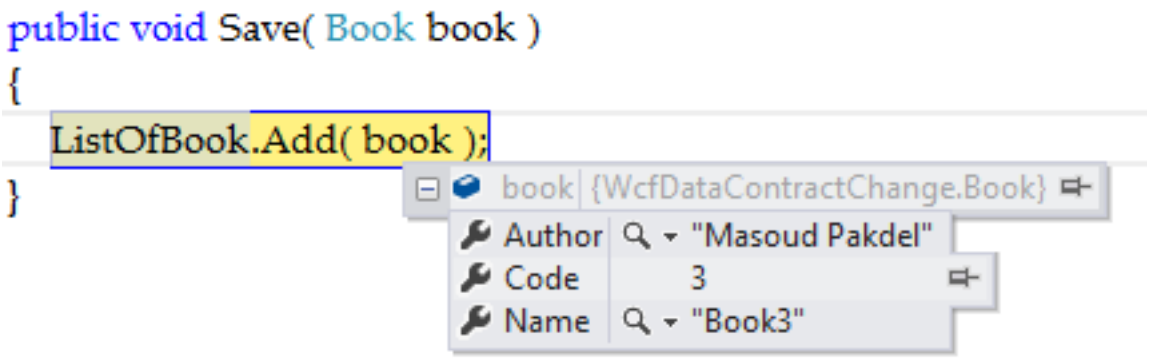


اما این همیشه راه حل مناسبی نیست.
روش دیگر این است که Desrialize کردن مدل را تغییر دهیم. بدین معنی که هر گاه مقدار Author برابر Null بود یک مقدار پیش فرض برای آن در نظر بگیریم. این کار با نوشتن یک متد و قراردادن OnDeserializingAttribute به راحتی امکان پذیر است.
کلاس Book به صورت زیر تغییر می‌کند.

```
[DataContract]
    public class Book
    {
        [DataMember]
        public int Code { get; set; }
        [DataMember]
        public string Name { get; set; }

        [DataMember( IsRequired = true )]
        public string Author { get; set; }

        [OnDeserializing]
        private void OnDeserializing( StreamingContext context )
        {
            if ( string.IsNullOrEmpty( Author ) )
            {
```

```
                Author = "Masoud Pakdel";
            }
        }
    }
```

حال اگر از سمت کلاینت کلاس Book دریافت شود که مقدار خاصیت Author آن برابر Null باشد توسط متد OnDeserializing مقدار پیش فرض به آن اعمال می‌شود.مثل تصویر زیر:



روش بعدی استفاده از اینترفیس IExtensibleDataObject است. بعد از اینکه کلاس Book این اینترفیس را پیاده سازی کرد مشکل Versioning Round Trip حل می‌شود. به این صورت که سرویس یا کلاینتی که نسخه قدیمی را می‌شناسد اگر نسخه جدید را دریافت کند خصوصیاتی را که نمی‌شناسد مثل Author در خاصیت ExtensionData ذخیره می‌شود و هنگامی که کلاس Book برای سرویس یا کلاینتی که نسخه جدید را می‌شناسد DataContractSerializer اطلاعات مورد نظر را از خصوصیت ExtensionData بیرون می‌کشد و کلاس Book جدید را باز سازی می‌کند. بررسی کلاس ExtensionData توسط خود DataContractSreializer انجام می‌شود و نیاز به هیچ گونه ای کد نویسی ندارد.

```
[DataContract]
    public class Book : IExtensibleDataObject
    {
        [DataMember]
        public int Code { get; set; }
        [DataMember]
        public string Name { get; set; }

        [DataMember]
        public string Author { get; set; }

        public virtual ExtensionDataObject ExtensionData
        {
            get { return _extensionData; }
            set
            {
                _extensionData = value;
            }
        }
        private ExtensionDataObject _extensionData;
    }
```

اگر کد متد GetAll سمت سرور را به صورت زیر تغییر دهیم که خاصیت Author هم مقدار داشته باشد با استفاده از خاصیت ExtensionData کلاینت هم از این مقدار مطلع خواهد شد.

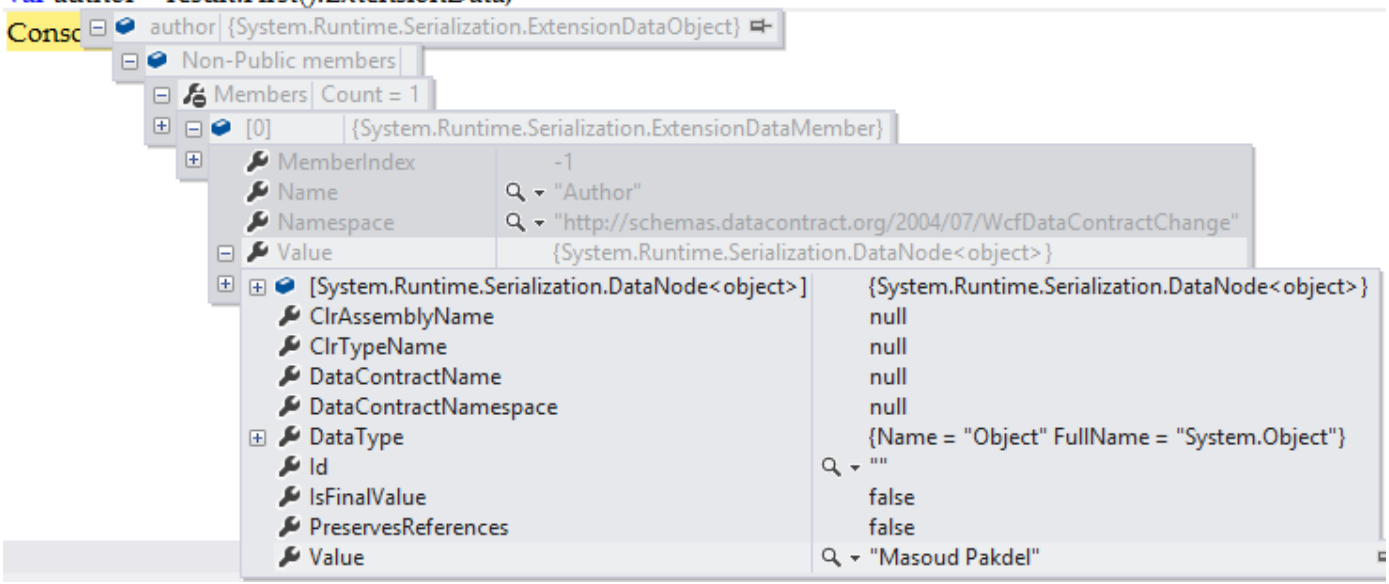
```
public IEnumerable<Book> GetAll()
        {
            ListOfBook.AddRange( new Book[]
            {
                new Book(){Code=1 , Name="Book1", Author="Masoud Pakdel"},
                new Book(){Code=2 , Name="Book2" },
```

```
            } );
            return ListOfBook;
        }
```

کلاینت هم به صورت زیر :

```
var result = sampleService.GetAll();
```

```
var author = result.First().ExtensionData;
```



همان طور که می‌بینید این نسخه از کلاینت هیچ گونه اطلاعی از وجود یک خاصیت به نام Author ندارد ولی از طریق ExtensionData متوجه می‌شود یک خاصیت به نام Author به مدل سمت سرور اضافه شده است.

اما در صورتی که قصد داشته باشیم که یک سرویس خاص از همان نسخه قدیمی کلاس Book استفاده کند و نیاز به نسخه جدید آن نداشته باشد می‌توانیم این کار را از طریق مقدار دهی True به خاصیت IgnoreExtensionDataObject در ServiceBehaviorAttribute انجام داد. بدین شکل

```
[ServiceBehavior( IgnoreExtensionDataObject = true )]
  public class SampleService : ISampleService
```

از این پس سرویس بالا از همان مدل Book بدون خاصیت Author استفاده می‌کند.

منابع :

http://msdn.microsoft.com/en-us/library/system.runtime.serialization.iextensibledataobject.aspx

http://msdn.microsoft.com/en-us/library/ms731083.aspx

http://msdn.microsoft.com/en-us/library/ms733832.aspx

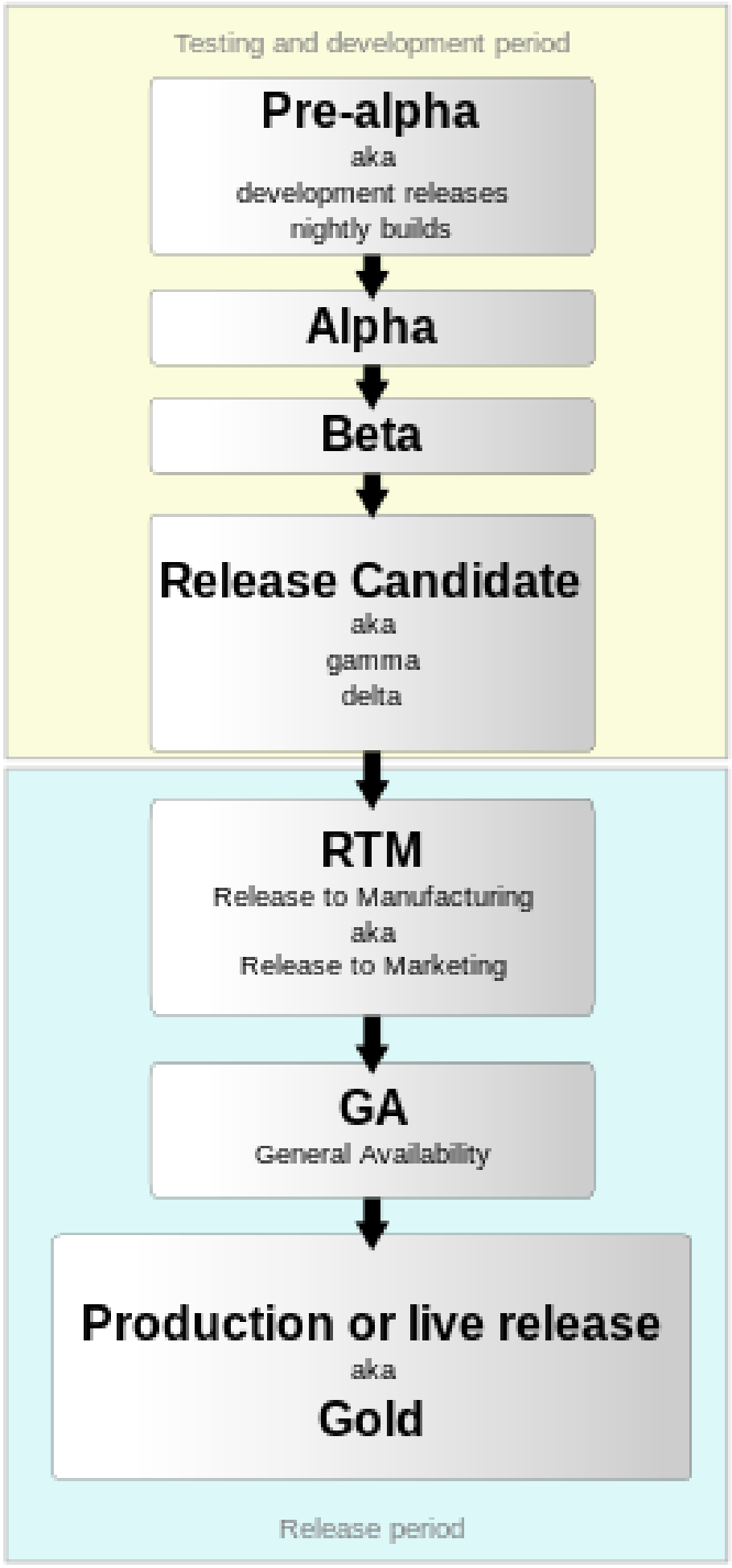
و...

عنوان: **آشنایی با نسخه بندی و چرخه انتشار نرم افزارها**
نویسنده: مجتبی کاویانی
تاریخ: ۱۳۹۲/۰۵/۰۵ ۲۳:۰
آدرس: www.dotnettips.info
برچسب‌ها: versioning, Software Development, version control, Assembly

نسخه بندی و چرخه انتشار یک نرم افزار، اهمیت زیادی در ارائه یک نرم افزار خوب دارد. هر چه نرم افزار شما بزرگ‌تر و از کتابخانه‌های بیشتری در تولید آن استفاده شده باشد، در بروز رسانی و نسخه بندی آن دقت بیشتری باید داشت و کار دشوارتری است. اما چگونه به بهترین روش، نسخه بندی نرم افرار خود را مدیریت نمایید.

مقدمه:

حتما نسخه بندی و نگارش‌های مختلف نرم افزارهایی را که استفاده می‌کنید، مشاهده نموده‌اید. نسخه‌های آلفا یا بتا یا نسخه بندی سالیانه یا با حروف و اعداد خاص. با این حال همه نرم افزارها علاوه بر عناوین متعارف، یک نسخه بندی داخلی عددی، شماره‌ای هم دارند. بسته به حجم و اندازه نرم افزارها، ممکن چرخه انتشار نرم افزارها متفاوت باشند. سیاست عرضه نرم افزار در هر شرکت هم متفاوت است. مثلا شرکت مایکروسافت برای عرضه ویندوز ابتدا نسخه بتا یا پیش نمایش آن را عرضه نموده تا با دریافت بازخوردهایی از استفاده کنندگان، نسخه نهایی نرم افزار خود را با حداقل ایراد و خطا عرضه نماید. البته این بخاطر بزرگی نرم افزار ویندوز نیز می‌باشد اما شرکت ادوبی اکثرا هر یکی دو سال بدون عرضه نسخه‌های قبل از نهایی یک دفعه نسخه جدیدی را رسما عرضه می‌نماید.


Testing and development period
Pre-alpha
aka
development releases
nightly builds
Alpha
Beta
Release Candidate
aka
gamma
delta
RTM
Release to Manufacturing
aka
Release to Marketing
GA
General Availability
Production or live release
aka
Gold
Release period

چرخه انتشار نرم افزار:

چرخه انتشار نرم افزار از زمان شروع کد نویسی تا عرضه نسخه نهایی می‌باشد که شامل چندین مرحله و عرضه نرم افزار می‌باشد.

Pre-alpha

این مرحله شامل تمام فعالیت‌های انجام شده قبل از مرحله تست می‌باشد. در این دوره آنالیز نیازمندیها، طراحی نرم افزار، توسعه نرم افزار و حتی تست واحد باشد. در نرم افزارهای سورس باز چندین نسخه قبل از آلفا ممکن است عرضه شوند.

Alpha

این مرحله شامل همه فعالیت‌ها از زمان شروع تست می‌باشد. البته منظور از تست، تست تیمی و تست خود نرم افزار می‌باشد. نرم افزارهای آلفا هنوز ممکن است خطا و اشکالاتی داشته باشند و ممکن است اطلاعات شما از بین رود. در این مرحله امکانات جدیدی مرتبا به نرم افزار اضافه می‌گردد.

Beta

نرم افزار بتا، همه قابلیت‌های آن تکمیل شده و خطاهای زیادی برای کامل شدن نرم افزار وجود دارد. در این مرحله بیشتر به تست کاهش تاثیرات به کاربران و تست کارایی دقت می‌شود. نسخه بتا، اولین نسخه‌ای خواهد بود که بیرون شرکت و یا سازمان در دسترس قرار می‌گیرد. برخی توسعه دهندگان به این مرحله preview، technical preview یا early access نیز می‌گویند.

Release candidate

در این مرحله نرم افزار، آماده عرضه به مصرف کنندگان است و نرم افزارهایی مثل سیستم عامل‌های ویندوز در دسترس تولید کنندگان قرار گرفته تا با جدیدترین سخت افزار خود یکپارچه شوند.

(General availability (GA

در این مرحله، عرضه عمومی نرم افزار و بازاریابی و فروش نرم افزار مد نظر است و علاوه بر این تست امنیتی و در نرم افزارهای خیلی بزرگ عرضه جهانی صورت می‌گیرد

مراحلی همچون عرضه در وب و پشتیبانی نیز وجود دارند.

نسخه بندی نرم افزار:

برنامه‌های ویندوزی یا وب در ویژوال استادیو یک فایل AssemblyInfo دارند که در قسمت آخر آن، اطلاعات مربوط به نسخه نرم افزار ذخیره می‌شود. هر نسخه نرم افزار شامل چهار عدد می‌باشد که با نقطه از هم جدا شده است.

Major Version

وقتی افزایش می‌یابد که تغییرات قابل توجهی در نرم افزار ایجاد شود

Minor Version

وقتی افزایش یابد که ویژگی جزئی یا اصلاحات قابل توجهی به نرم افزار ایجاد شود.

Build Number

به ازای هر بار ساخته شدن پروژه افزایش می‌یابد.

Revision

وقتی افزایش می‌یابد که نواقص و باگ‌های کوچکی رفع شوند.

وقتی که major یا minor افزایش یابد می‌تواند با کلماتی همچون alpha، beta یا release candidate همراه شود.در اکثر برنامه‌های تجاری اولین شمارهٔ انتشار یک محصول از نسخهٔ شمارهٔ یک شروع می‌شود. ترتیب نسخه بندی هم ممکن است تغییر یابد

```
major.minor[.build[.reversion]]
```

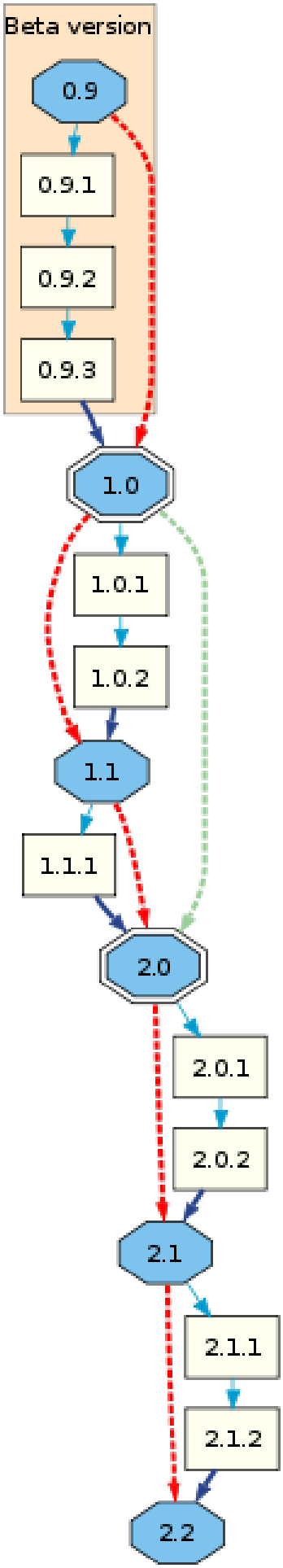
یا

```
major.minor[.maintenance[.build]]
```

نسخه بندی مایکروسافت:

اگر به نسخه برنامه Office توجه کرده باشید مثلا Office 2013 نسخه 15.0.4481.1508 می‌باشد که در این روش از تاریخ شروع پروژه و تعداد ماه‌ها یا روزها و یا ثانیه‌ها با یک الگوریتم خاص برای تولید نسخه نرم افزار استفاده می‌شود.

نسخه بندی معنایی:


Beta version
0.9
0.9.1
0.9.2
0.9.3
1.0
1.0.1
1.0.2
1.1
1.1.1
2.0
2.0.1
2.0.2
2.1
2.1.1
2.1.2
2.2

به عنوان یک راه حل، مجموعه‌ی ساده‌ای از قوانین و الزامات که چگونگی طراحی شماره‌های نسخه و افزایش آن را مشخص می‌کند، وجود دارد. برای کار کردن با این سیستم، شما ابتدا نیاز به اعلام API عمومی دارید. این خود ممکن است شامل مستندات و یا اجرای کد باشد.

علیرغم آن، مهم است که این API، روشن و دقیق باشد. هنگامیکه API عمومی خود را تعیین کردید، تغییرات برنامه شما بر روی نسخه API عمومی تاثیر خواهد داشت و آنرا افزایش خواهد داد. بر این اساس، این مدل نسخه‌بندی را در نظر بگیرید: X.Y.Z یعنی (Major.Minor.Patch).

رفع حفره‌هایی که بر روی API عمومی تاثیر نمی‌گذارند، مقدار Patch را افزایش می‌دهند، تغییرات جدیدی که سازگار با نسخه قبلی است، مقدار Minor را افزایش می‌دهند و تغییرات جدیدی که کاملا بدیع هستند و به نحوی با تغییرات قبلی سازگار نیستند مقدار Major را افزایش می‌دهند.

نرم‌افزارهایی که از نسخه بندی معنایی استفاده می‌کنند، باید یک API عمومی داشته باشند. این API می‌تواند در خود کد یا و یا به طور صریح در مستندات باشد که باید دقیق و جامع باشد.
یک شماره نسخه صحیح باید به شکل X.Y.Z باشد که در آن X،Y و Z اعداد صحیح غیر منفی هستند. X نسخه‌ی Major می‌باشد، Y نسخه‌ی Minor و Z نسخه‌ی Patch می‌باشد. هر عنصر باید یک به یک و بصورت عددی افزایش پیدا کند. به عنوان مثال: 1.9.0 -> 1.10.0 <- 1.11.0
هنگامی که به یک نسخه‌ی Major یک واحد اضافه می‌شود، نسخه‌ی Minor و Patch باید به حالت 0 (صفر) تنظیم مجدد گردد.
هنگامی که به شماره نسخه‌ی Minor یک واحد اضافه می‌شود، نسخه‌ی Patch باید به حالت 0 (صفر) تنظیم مجدد شود. به عنوان مثال: 1.1.3 -> 2.0.0 و 2.1.7 -> 2.2.0
هنگامیکه یک نسخه از یک کتابخانه منتشر می‌شود، محتوای کتابخانه مورد نظر نباید به هیچ وجه تغییری داشته باشد. هر گونه تغییر جدیدی باید در قالب یک نسخه جدید انتشار پیدا کند.
نسخه‌ی Major صفر (Y.Z.0) برای توسعه‌ی اولیه است. هر چیزی ممکن است در هر زمان تغییر یابد. API عمومی را نباید پایدار در نظر گرفت.
نسخه 1.0.0 در حقیقت API عمومی را تعریف می‌کند. چگونگی تغییر و افزایش هر یک از نسخه‌ها بعد از انتشار این نسخه، وابسته به API عمومی و تغییرات آن می‌باشد.
نسخه Patch یا (x.y.Z | x > 0) فقط در صورتی باید افزایش پیدا کند که تغییرات ایجاد شده در حد برطرف کردن حفره‌های نرم‌افزار باشد. برطرف کردن حفره‌های نرم‌افزار شامل اصلاح رفتارهای اشتباه در نرم‌افزار می‌باشد.
نسخه Minor یا (x.Y.z | x > 0) فقط در صورتی افزایش پیدا خواهد کرد که تغییرات جدید و سازگار با نسخه قبلی ایجاد شود. همچنین این نسخه باید افزایش پیدا کند اگر بخشی از فعالیت‌ها و یا رفتارهای قبلی نرم‌افزار به عنوان فعالیت منقرض شده اعلام شود. همچنین این نسخه می‌تواند افزایش پیدا کند اگر تغییرات مهم و حیاتی از طریق کد خصوصی ایجاد و اعمال گردد. تغییرات این نسخه می‌تواند شامل تغییرات نسخه Patch هم باشد. توجه به این نکته ضروری است که در صورت افزایش نسخه Minor، نسخه Patch باید به 0 (صفر) تغییر پیدا کند.
نسخه Major یا (X.y.z | X > 0) در صورتی افزایش پیدا خواهد کرد که تغییرات جدید و ناهمخوان با نسخه فعلی در نرم‌افزار اعمال شود. تغییرات در این نسخه می‌تواند شامل تغییراتی در سطح نسخه Minor و Patch نیز باشد. باید به این نکته توجه شود که در صورت افزایش نسخه Major، نسخه‌های Minor و Patch باید به 0 (صفر) تغییر پیدا کنند.
یک نسخه قبل از انتشار می‌تواند توسط یک خط تیره (dash)، بعد از نسخه Patch (یعنی در انتهای نسخه) که انواع با نقطه (dot) از هم جدا می‌شوند، نشان داده شود. نشان‌گر نسخه قبل از انتشار باید شامل حروف، اعداد و خط تیره باشد [0-9A-Za-z-]. باید به این نکته دقت داشت که نسخه‌های قبل از انتشار خود به تنهایی یک انتشار به حساب می‌آیند اما اولویت و اهمیت نسخه‌های عادی را ندارد. برای مثال: 1.0.0-x.7.z.92 ، 1.0.0-0.3.7 ، 1.0.0-alpha.1 ، alpha
یک نسخه Build می‌تواند توسط یک علامت مثبت (+)، بعد از نسخه Patch یا نسخه قبل از انتشار (یعنی در انتهای نسخه) که انواع آن با نقطه (dot) از هم جدا می‌شوند، نشان داده شود. نشان‌گر نسخه Build باید شامل حروف، اعداد و خط تیره باشد [0-

-9A-Za-z]. باید به این نکته دقت داشت که نسخه‌های Build خود به تنهایی یک انتشار به حساب می‌آیند و اولویت و اهمیت بیشتری نسبت به نسخه‌های عادی دارند. برای مثال: 1.0.0+build.11.e0f985a ، 1.3.7+build.1

اولویت‌بندی نسخه‌ها باید توسط جداسازی بخش‌های مختلف یک نسخه به اجزای تشکیل دهنده آن یعنی Minor، Major، Patch، نسخه قبل از انتشار و نسخه Build و ترتیب اولویت بندی آن‌ها صورت گیرد. نسخه‌های Minor، Major و Patch باید بصورت عددی مقایسه شوند. مقایسه نسخه‌های قبل از انتشار و نسخه Build باید توسط بخش‌های مختلف که توسط جداکننده‌ها (نقطه‌های جداکننده) تفکیک شده است، به این شکل سنجیده شود:

بخش‌هایی که فقط حاوی عدد هستند، بصورت عددی مقایسه می‌شوند و بخش‌هایی که حاری حروف و یا خط تیره هستند بصورت الفبایی مقایسه خواهند شد.

بخش‌های عددی همواره اولویت پایین‌تری نسبت به بخش‌های غیر عددی دارند. برای مثال:

1.0.0-alpha < 1.0.0-alpha.1 < 1.0.0-beta.2 < 1.0.0-beta.11 < 1.0.0-rc.1 < 1.0.0-rc.1+build.1 < 1.0.0 < 1.0.0+0.3.7 < 1.3.7+build < 1.3.7+build.2.b8f12d7 < 1.3.7+build.11.e0f985a

منبع نسخه بندی معنایی : semver.org

نظرات خوانندگان

نویسنده: میثم هوشمند
تاریخ: ۱۳۹۲/۰۷/۲۸ ۱۰:۲۱

در خصوص RTM توضیح خاصی ارائه نشده!

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۷/۲۹ ۱۸:۵۱

rtm هست در تصویر اول. به معنای release to manufacturing است. مثلا مایکروسافت اول ویندوز 8 رو در اختیار لپ تاپ سازها قرار می‌ده تا نصب کنند. بعد همون نگارش چند وقت بعد برای عموم توزیع میشه. اگر این RTM برای برنامه نویس‌ها باشه، یعنی نگارش نهایی که مثلا به دارندگان اکانت‌های MSDN اول ارائه شده. بعد از چند وقت همون توزیع‌ها با سریال آزمایشی در اختیار عموم قرار می‌گیرند. rtm به معنای کیفیتی از کار است که قابل ارائه است به عموم در سطح وسیع.

نویسنده: ناظم
تاریخ: ۱۳۹۲/۰۸/۲۵ ۱۳:۵۴

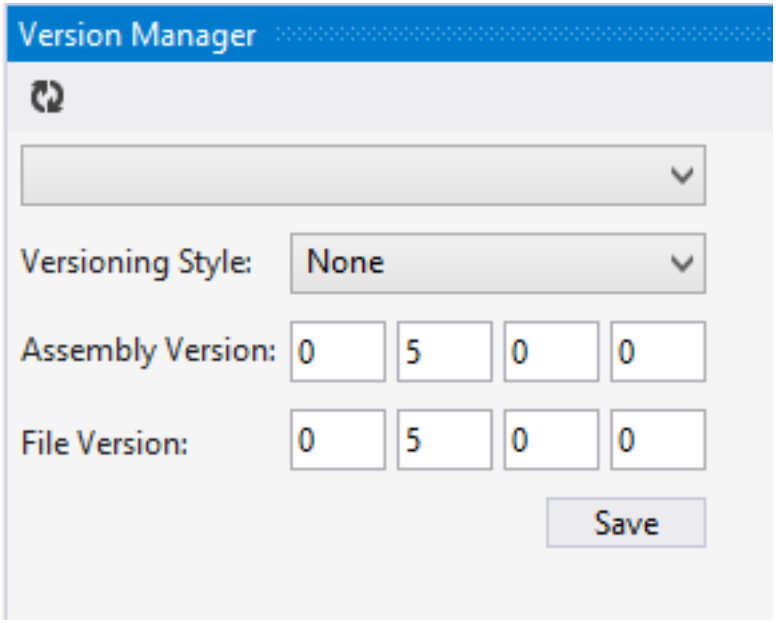
سلام
بنا بر این میتوان نتیجه گرفت که نسخه rtm همیشه همان نسخه ای هست که انتشار نهایی و عمومی میشه؟

عنوان: **آشنایی و بررسی ابزار Version Manager**
نویسنده: مجتبی کاویانی
تاریخ: ۱۳۹۲/۰۶/۱۲ ۰:۳۵
آدرس: www.dotnettips.info
برچسب‌ها: versioning, version control

در مطلبی با نسخه بندی و چرخه انتشار نرم افزار آشنا شدید. اما کمبود ابزاری کارآمد و حرفه ای در ویژوال استادیو من را بر آن داشت تا افزونه‌ای را تهیه و در دسترس تمامی توسعه دهندگان قرار دهم. پس از مطالعه و بررسی روش‌های نگارش بندی نرم افزار و ابزارهای موجود، دو روش عمده نسخه بندی نرم افزار وجود دارد که در زیر آورده شده است.
نسخه بندی معنایی
نسخه بندی استاندارد مایکروسافت

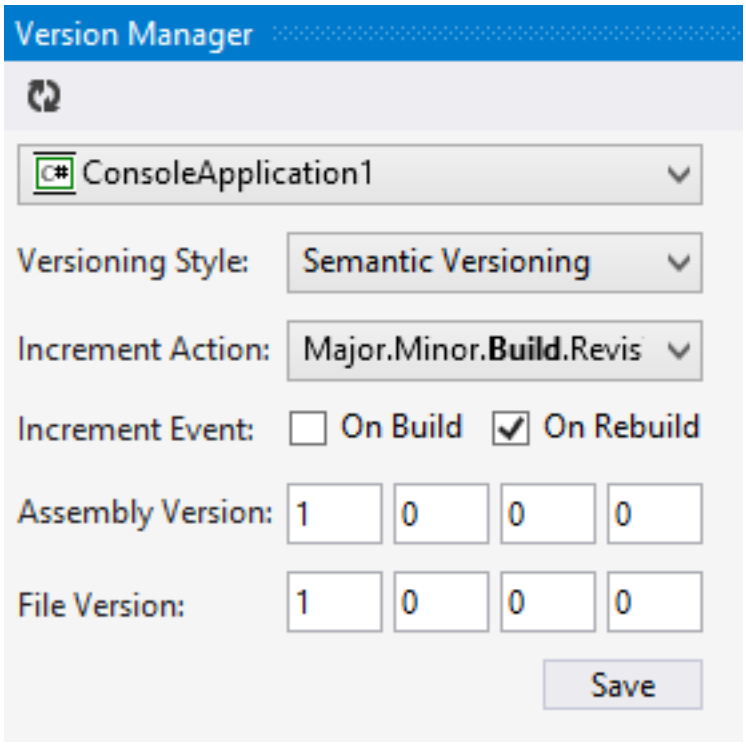
در روش معنایی چهار قسمت نسخه بندی Major.Minor.Build.Revision به صورت زیر افزایش و تغییر می‌یابد:
Revision تا 1000 افزایش می‌یابد
در اجرای بعدی به 0 تنظیم شده و قسمت Build بعلاوه 1 می‌شود.
Build تا 100 افزایش می‌یابد
در اجرای بعدی Build به 0 تنظیم شده و قسمت Minor بعلاوه 1 می‌شود و Revision هم 0 می‌شود.
Minor تا 10 افزایش می‌یابد.
در اجرای بعدی Minor به 0 تنظیم شده، قسمت Major بعلاوه 1 می‌شود و قسمت‌های قبل همه 0 می‌شوند.

در روش استاندارد ماکروسافت مقادیر قسمت‌های Build و Revision از الگوریتم زیر محاسبه می‌شود.
مقدار Build از مجموع روزهای که از تاریخ پروژه گذشته است بدست می‌آید.
مقدار Revision از مجموع ثانیه‌های که از نیمه شب محلی روز جاری تا زمان اجرای پروژه تقسم بر دو بدست می‌آید.
مقادیر Major و Minor هم با توجه تولید نسخه جدید و تغییرات عمده در نرم افزار بصورت دستی تغییر می‌یابد.
راهنمای نصب و اجرا:
ابتدا افزونه Version Manager را از سایت ویژوال استادیو گالری دریافت و نصب کنید.
سپس از منوی Version Manager > Other Windows > View را انتخاب کنید.



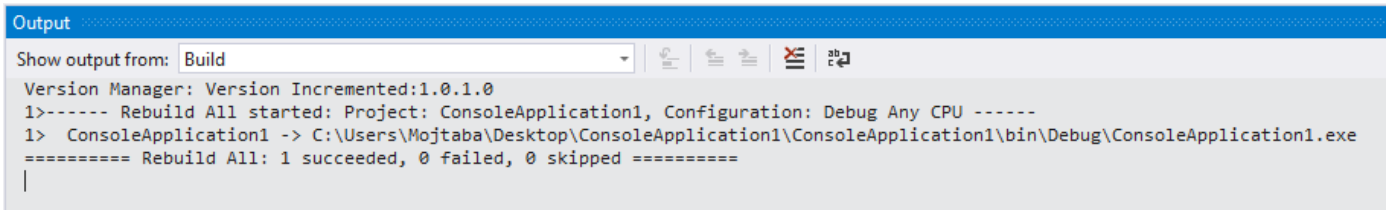
پس از باز شدن پنجره Version Manager پروژه‌های Solution جاری بارگذاری و Assembly Version و File Version هر پروژه را می‌توانید به راحتی مشاهده و یا تغییر دهید.

اگر بخواهید بصورت خودکار بر اساس شمای انتخاب شده، نسخه نرم افزار را افزایش دهید، شمای نسخه بندی را انتخاب کنید. در دو زمان Build و یا Rebuild پروژه می توان نسخه را افزایش داد.



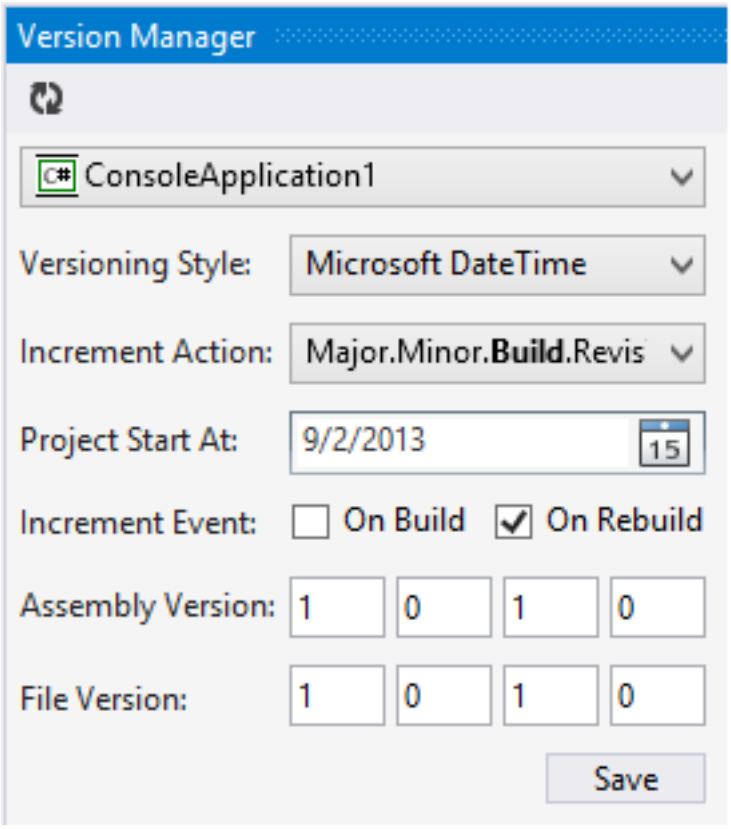
Semantic Versioning:

گزینه Semantic Versioning را از قسمت Version Style انتخاب کنید.
گزینه Increment Action قسمتی از نگارش که می‌خواهید شمای نسخه بندی بر روی آن اعمال شود را مشخص می‌کند. که دو گزینه Build یا Revision وجود دارد
گزینه Increment Event زمان رویداد اعمال شمای انتخاب شده را مشخص می‌سازد.
تنظیمات را ذخیره و پروژه خود را Rebuild نمایید.
در پنجره Output نسخه جدید نشان داده می‌شود.

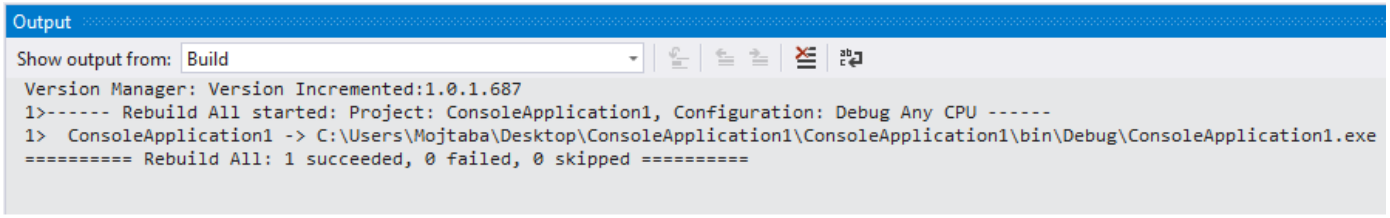


Microsoft DateTime:

گزینه Microsoft DateTime را از قسمت Version Style انتخاب کنید.



تاریخ شروع پروژه بصورت خودکار خوانده و نمایش داده می‌شود یا تاریخ شروع پروژه را انتخاب کنید
با توجه به انتخاب Increment Action نسخه جدید محاسبه می‌شود.
پروژه را Rebuild و نسخه جدید را مشاهده نمایید.



همچنین از پروژه‌های C# و VB.NET و C++ Managed پشتیبانی می‌کند

این ابزار هنوز در نگارش بتا است و ممکن است باگ‌هایی داشته باشد.

نظرات و پیشنهادات شما توسعه دهندگان عزیز می‌تواند موجب هر چه بهتر و کامل‌تر شدن این ابزار باشد.

نظرات خوانندگان

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۶/۱۸ ۱۷:۲۳

با تشکر از کار شما
من این ماژول را نصب کردم اما زمان فعال سازی خطا صادر می‌شود، فایل ActivityLog را بررسی کردم خطای ذیل ثبت شده بود.

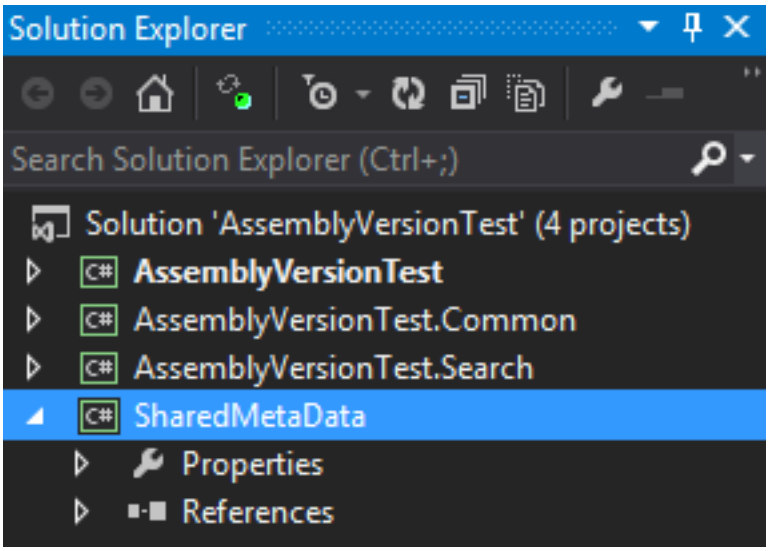
```
<entry>
    <record>5390</record>
    <time>2013/09/09 08:41:39.525</time>
    <type>Error</type>
    <source>VisualStudio</source>
    <description>End package load [VersionManagerPackage]</description>
    <guid>{775E4DAB-A8DC-46E5-A64B-4072C0DD3A42}</guid>
    <hr>80004005 - E_FAIL</hr>
    <errorinfo>Could not load file or assembly 'Microsoft.VisualStudio.Shell.12.0, Version=12.0.0.0,
Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a' or one of its dependencies. The system cannot find
the file specified.</errorinfo>
  </entry>
```

عنوان: **مدیریت هماهنگ شماره نگارش اسمبلی در چندین پروژه‌ی ویژوال استودیو**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۷/۱۶ ۹:۵
آدرس: www.dotnettips.info
گروه‌ها: Visual Studio, versioning, T4, Text Template

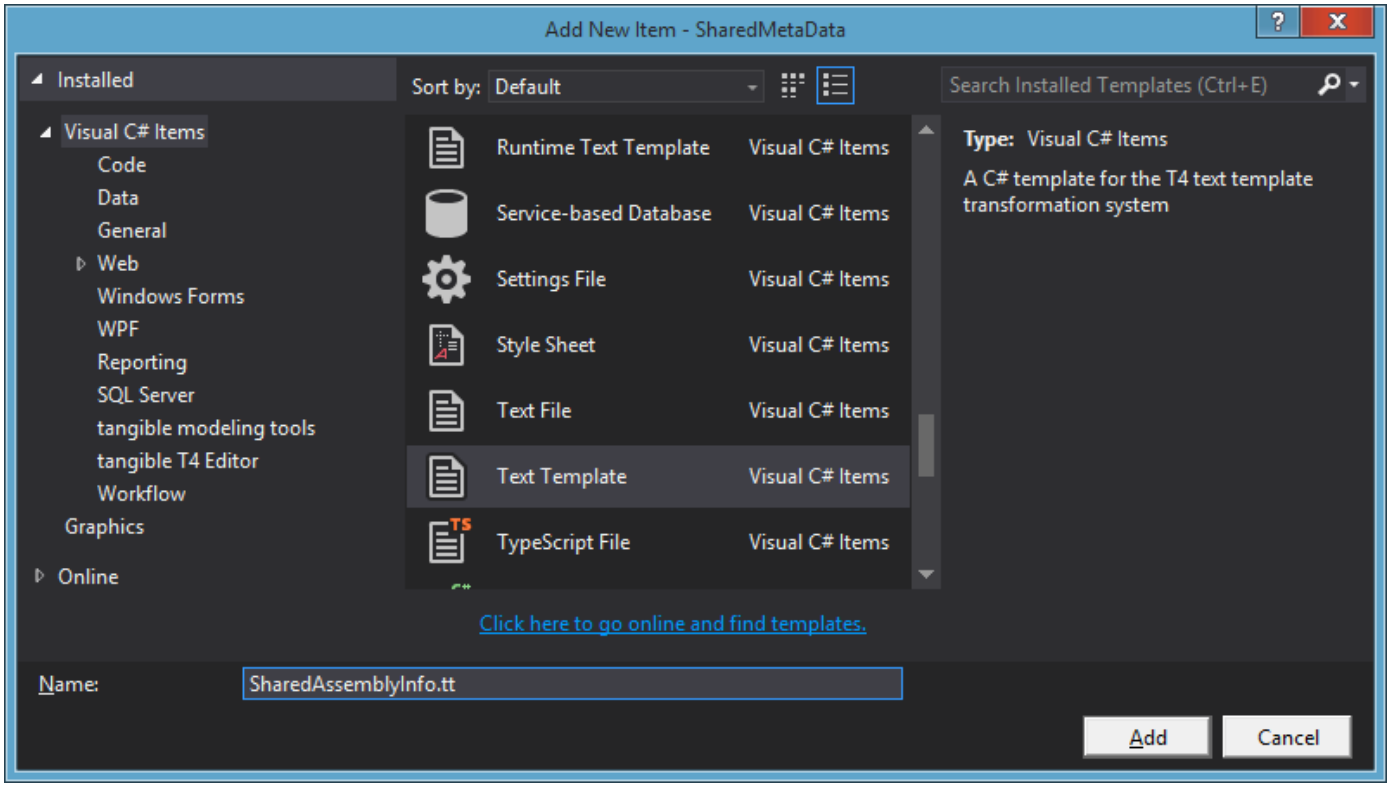
عموما برای نگهداری ساده‌تر قسمت‌های مختلف یک پروژه، اجزای آن به اسمبلی‌های مختلفی تقسیم می‌شوند که هر کدام در یک پروژه‌ی مجزای ویژوال استودیو قرار خواهند گرفت. یکی از نیازهای مهم این نوع پروژه‌ها، داشتن شماره نگارش یکسانی بین اسمبلی‌های آن است. به این ترتیب توزیع نهایی ساده‌تر شده و همچنین پشتیبانی از آن‌ها در دراز مدت، بر اساس این شماره نگارش بهتر صورت خواهد گرفت. برای مثال در لاگ‌های خطای برنامه با بررسی شماره نگارش اسمبلی مرتبط، حداقل می‌توان متوجه شد که آیا کاربر از آخرین نسخه‌ی برنامه استفاده می‌کند یا خیر.
روش معمول انجام این‌کار، به روز رسانی دستی تمام فایل‌های AssemblyInfo.cs یک Solution است و همچنین اطمینان حاصل کردن از همگام بودن آن‌ها. در ادامه قصد داریم با استفاده از فایل‌های T4، یک فایل SharedAssemblyInfo.tt را جهت تولید اطلاعات مشترک Build بین اسمبلی‌های مختلف یک پروژه، تولید کنیم.

ایجاد پروژه‌ی SharedMetaData

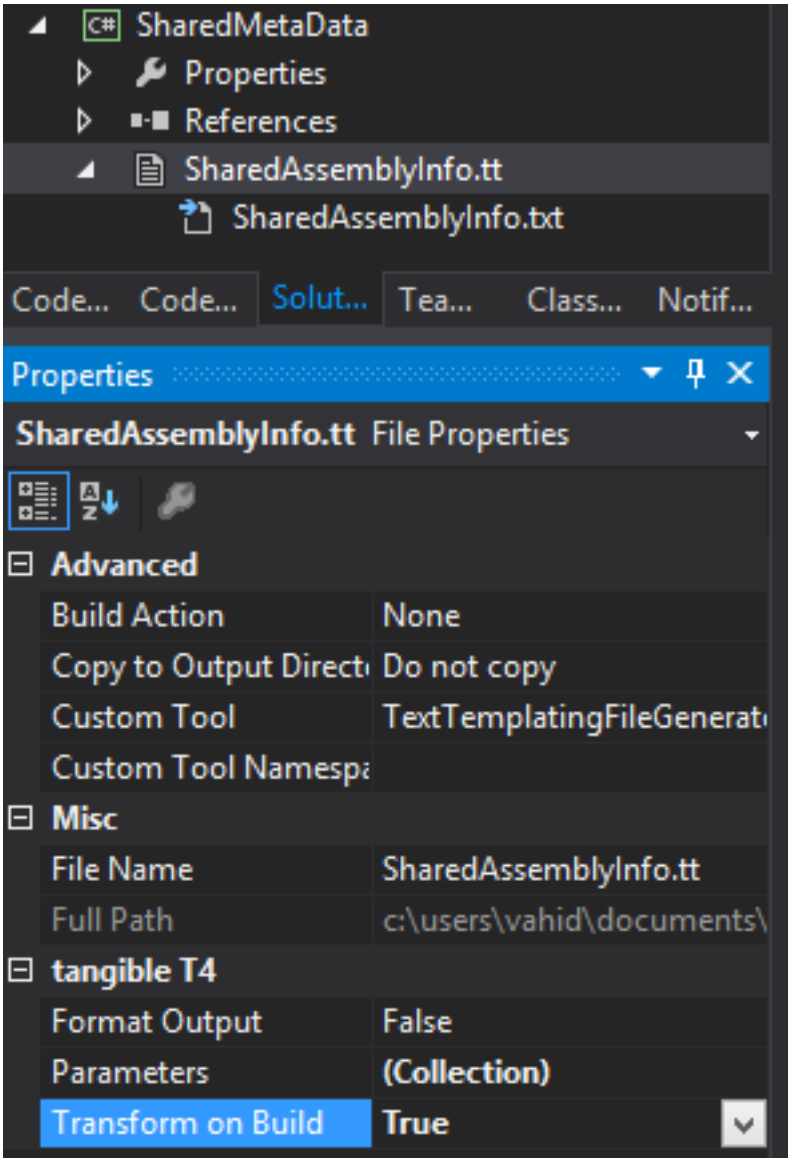
برای نگهداری فایل مشترک SharedAssemblyInfo.cs نهایی و همچنین اطمینان از تولید مجدد آن به ازای هر Build، یک پروژه‌ی class library جدید را به نام SharedMetaData به Solution جاری اضافه کنید.



سپس نیاز است یک فایل text template جدید را به نام SharedAssemblyInfo.tt، به این پروژه اضافه کنید.



به خواص فایل SharedAssemblyInfo.tt مراجعه کرده و Transform on build آن‌را true کنید . به این ترتیب مطمئن خواهیم شد این فایل به ازای هر build جدید، مجددا تولید می‌گردد.



اکنون محتوای این فایل را به نحو ذیل تغییر دهید:

```
<#@ template debug="false" hostspecific="false" language="C#" #>
//
// This code was generated by a tool. Any changes made manually will be lost
// the next time this code is regenerated.
//
using System.Reflection;
using System.Resources;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;

[assembly: AssemblyCompany("some name")]
[assembly: AssemblyCulture("")]
[assembly: NeutralResourcesLanguageAttribute("en")]

[assembly: AssemblyProduct("product name")]
[assembly: AssemblyCopyright("Copyright VahidN 2014")]
[assembly: AssemblyTrademark("some name")]

#if DEBUG
[assembly: AssemblyConfiguration("Debug")]
#else
[assembly: AssemblyConfiguration("Release")]
#endif

// Assembly Versions are incremented manually when branching the code for a release.
```

```
[assembly: AssemblyVersion("<#= this.MajorVersion #>.<#= this.MinorVersion #>.<#= this.BuildNumber
#>.<#= this.RevisionNumber #>")]
// Assembly File Version should be incremented automatically as part of the build process.
[assembly: AssemblyFileVersion("<#= this.MajorVersion #>.<#= this.MinorVersion #>.<#= this.BuildNumber
#>.<#= this.RevisionNumber #>")]

<#+
// Manually incremented for major releases, such as adding many new features to the solution or
introducing breaking changes.
int MajorVersion = 1;
// Manually incremented for minor releases, such as introducing small changes to existing features or
adding new features.
int MinorVersion = 0;
// Typically incremented automatically as part of every build performed on the Build Server.
int BuildNumber = (int)(DateTime.UtcNow - new DateTime(2013,1,1)).TotalDays;
// Incremented for QFEs (a.k.a. “hotfixes” or patches) to builds released into the Production
environment.
// This is set to zero for the initial release of any major/minor version of the solution.
int RevisionNumber = 0;
#>
```

در این فایل اجزای شماره نگارش برنامه به صورت متغیر تعریف شده‌اند. هر بار که نیاز است یک نگارش جدید ارائه شود، می‌توان این اعداد را تغییر داد.

MajorVersion با افزودن تعداد زیادی قابلیت به برنامه، به صورت دستی تغییر می‌کند. همچنین اگر یک breaking change در برنامه یا کتابخانه وجود داشته باشد نیز این شماره باید تغییر نماید.

MinorVersion با افزودن ویژگی‌های کوچکی به نگارش فعلی برنامه تغییر می‌کند.

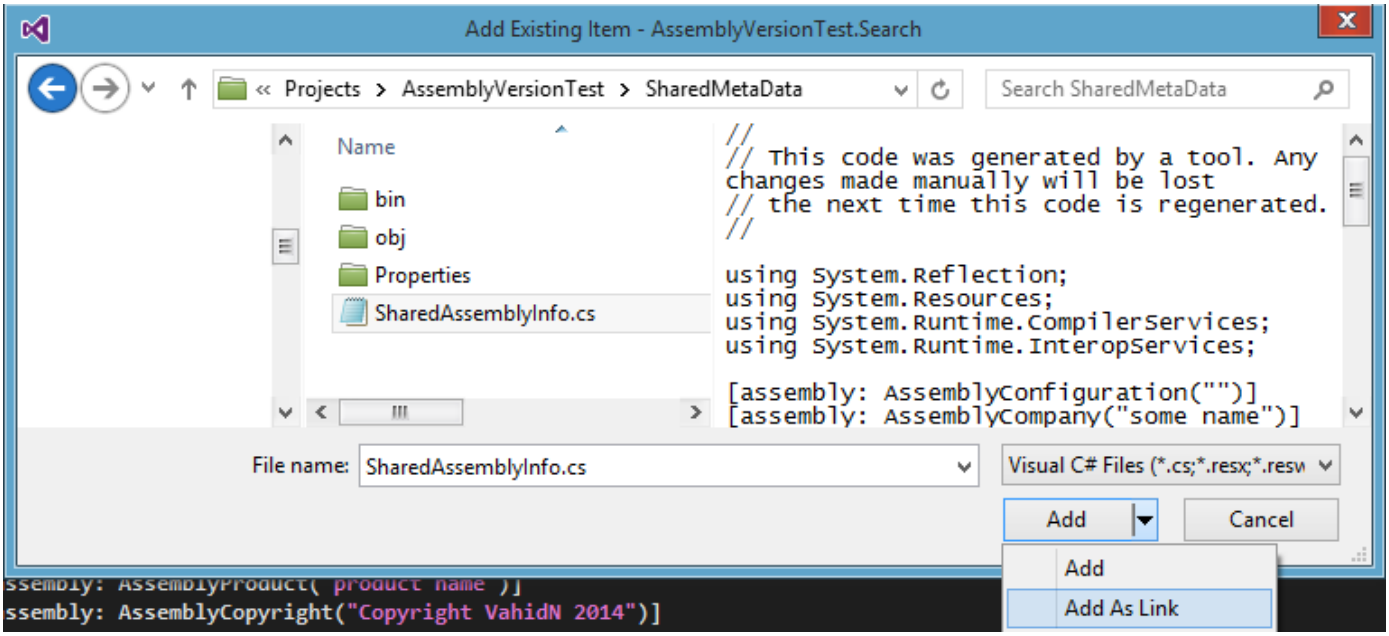
BuildNumber به صورت خودکار بر اساس هر Build انجام شده باید تغییر یابد. در اینجا این عدد به صورت خودکار به ازای هر روز، یک واحد افزایش پیدا می‌کند. ابتدای مبداء آن در این مثال، 2013 قرار گرفته‌است.

RevisionNumber با ارائه یک وصله جدید برای نگارش فعلی برنامه، به صورت دستی باید تغییر کند. اگر اعداد شماره نگارش major یا minor تغییر کنند، این عدد باید به صفر تنظیم شود.

اکنون اگر این محتوای جدید را ذخیره کنید، فایل SharedAssemblyInfo.cs به صورت خودکار تولید خواهد شد.

افزودن فایل SharedAssemblyInfo.cs به صورت لینک به تمام پروژه‌ها

نحوه‌ی افزودن فایل جدید SharedAssemblyInfo.cs به پروژه‌های موجود، اندکی متفاوت است با روش معمول افزودن فایل‌های cs هر پروژه. ابتدا از منوی پروژه گزینه‌ی add existing item را انتخاب کنید. سپس فایل SharedAssemblyInfo.cs را یافته و به صورت add as link، به تمام پروژه‌های موجود اضافه کنید.



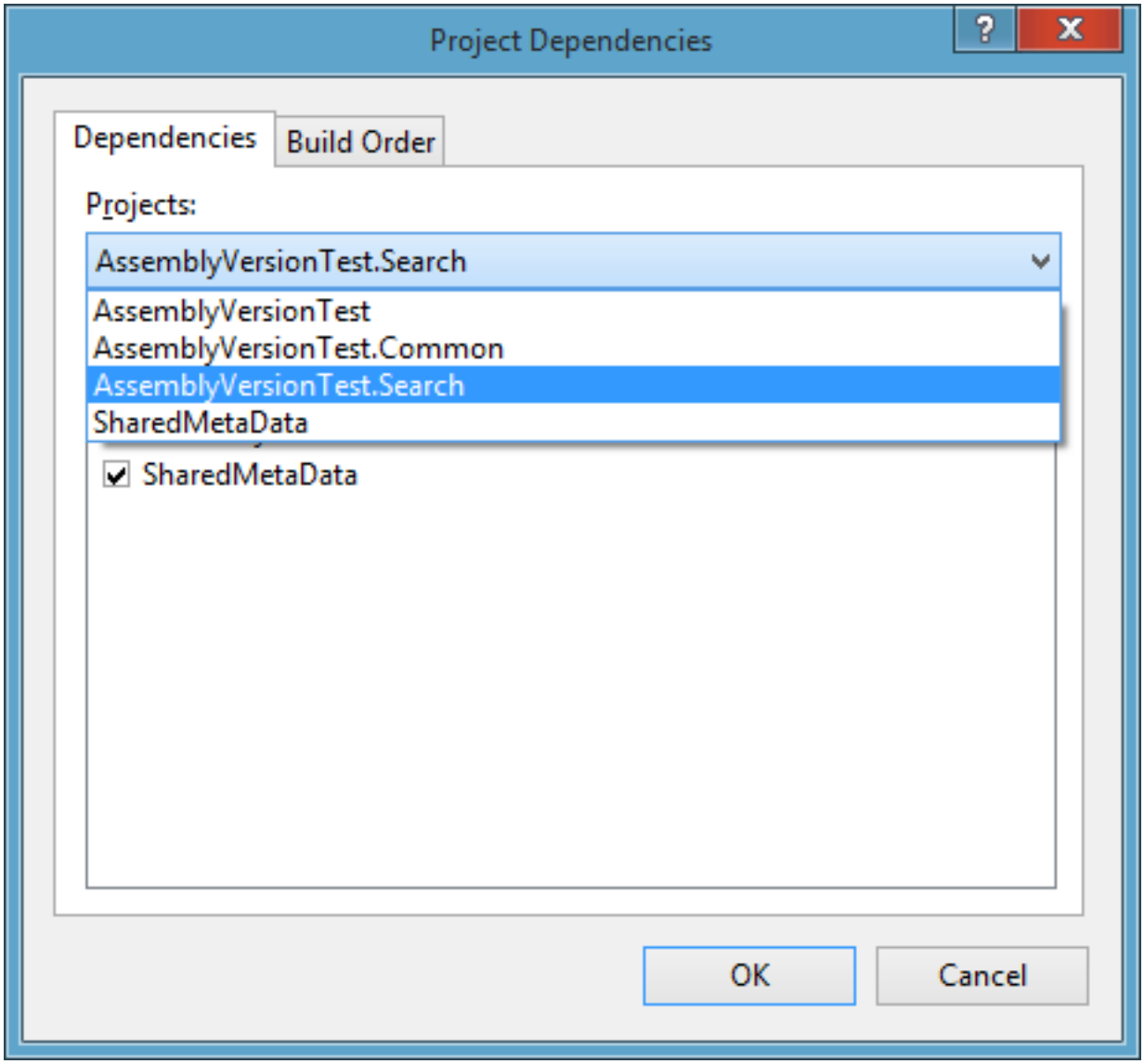
اینکار باید در مورد تمام پروژه‌ها صورت گیرد. به این ترتیب چون فایل SharedAssemblyInfo.cs به این پروژه‌ها صرفا لینک شده‌است، اگر محتوای آن در پروژه‌ی metadata تغییر کند، به صورت خودکار و یک دست، در تمام پروژه‌های دیگر نیز منعکس خواهد شد.

در ادامه اگر بخواهید Solution را Build کنید، پیام تکراری بودن یک سری از ویژگی‌ها را یافت خواهید کرد. این مورد از این جهت رخ می‌دهد که هنوز فایل‌های AssemblyInfo.cs اصلی، در پروژه‌های برنامه موجود هستند.
این فایل‌ها را یافته و صرفا چند سطر همیشه ثابت ذیل را در آن‌ها باقی بگذارید:

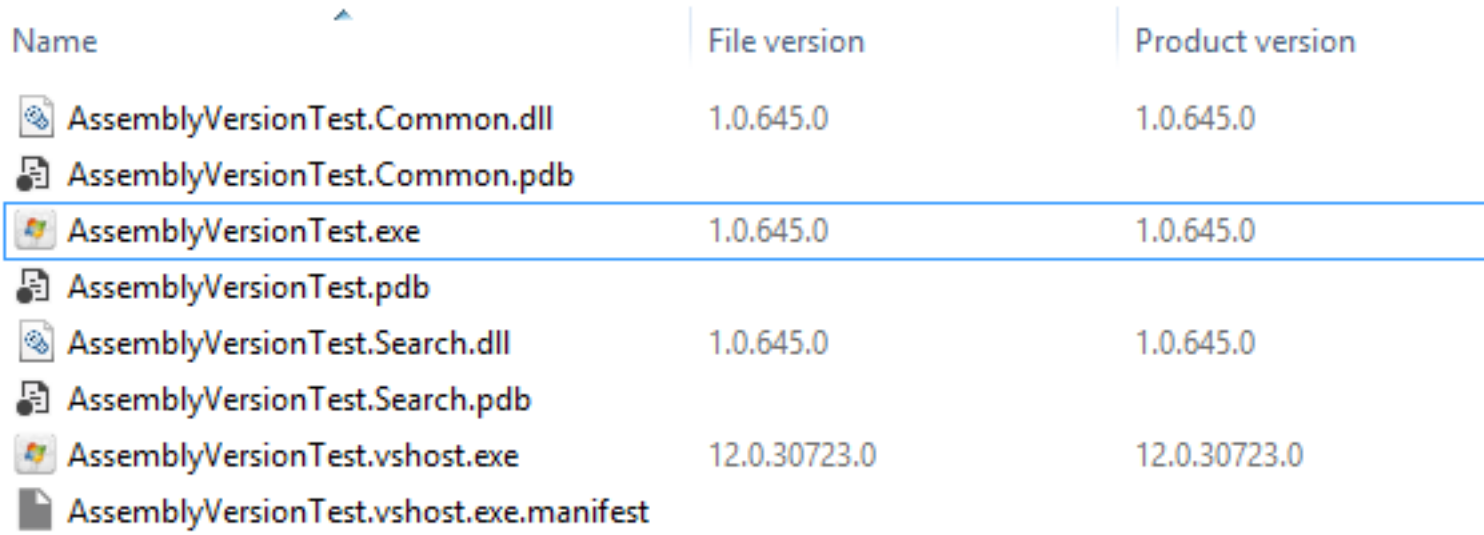
```
using System.Reflection;
using System.Runtime.InteropServices;

[assembly: AssemblyTitle("title")]
[assembly: AssemblyDescription("")]
[assembly: ComVisible(false)]
[assembly: Guid("9cde6054-dd73-42d5-a859-7d4b6dc9b596")]
```

اضافه کردن build dependency به پروژه MetaData

در پایان کار نیاز است اطمینان حاصل کنیم، فایل SharedAssemblyInfo.cs به صورت خودکار پیش از Build هر پروژه، تولید می‌شود. برای این منظور، از منوی Project، گزینه‌ی Project dependencies را انتخاب کنید. سپس در برگه‌ی dependencies آن، به ازای تمام پروژه‌های موجود، گزینه‌ی SharedMetadata را انتخاب نمائید.



این مساله سبب اجرای خودکار فایل SharedAssemblyInfo.tt پیش از هر Build می‌شود و به این ترتیب می‌توان از تازه بودن اطلاعات SharedAssemblyInfo.cs اطمینان حاصل کرد.
اکنون اگر پروژه را Build کنید، تمام اجزای آن شماره نگارش یکسانی را خواهند داشت:



و در دفعات آتی، تنها نیاز است تک فایل `SharedAssemblyInfo.tt` را برای تغییر شماره نگارش‌های اصلی، ویرایش کرد.

نظرات خوانندگان

نویسنده: لیبرتاد
تاریخ: ۱۳۹۳/۰۷/۱۹ ۱۱:۴۱

آموزش خوبی بود البته در اکثر اوقات بهتر است که شماره نگارش اسمبلی‌های پروژه‌ها یکی نباشد. ممکن است از چندین پروژه یک یا چندتای آنها در آپدیتهای مختلف هیچ تغییری نداشته باشند

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۷/۱۹ ۱۳:۰

یک اسمبلی در پروژه، به خودی خود فاقد مفهوم است و در قالب نگارش کلی برنامه مفهوم پیدا می‌کند.
فرض کنید برنامه شما از یک فایل exe به همراه دو اسمبلی A و B، تشکیل شده‌است. اسمبلی A، نگارش یک دارد. اسمبلی B نگارش 2 و کل برنامه در نگارش 2.5 است. خطایی به شما گزارش شده‌است که در آن استثنای حاصل، از نگارش یک اسمبلی A صادر شده‌است. این مشکل که در نتیجه‌ی در یافت پردازش اشتباهی از اسمبلی B بوده و در نگارش 2 آن برطرف شده، به صورت خودکار با ارتقاء به آخرین نگارش برنامه، برطرف می‌شود.
سؤال: آیا اکنون می‌توانید تشخیص دهید کاربر از آخرین نگارش محصول شما استفاده می‌کند؟ نگارش یک A، آخرین نگارش آن است و اما برنامه در نگارش 2.5 قرار دارد. کاربر هم مدتی است که برنامه را به روز نکرده‌است.
یک سیستم از همکاری اجزای مختلف آن مفهوم پیدا می‌کند.
برای مطالعه بیشتر: « Best Practices for .NET Assembly Versioning ». عبارت «ensuring all of the various assemblies in the solution share the same version» حداقل دوبار در آن تکرار شده‌است.