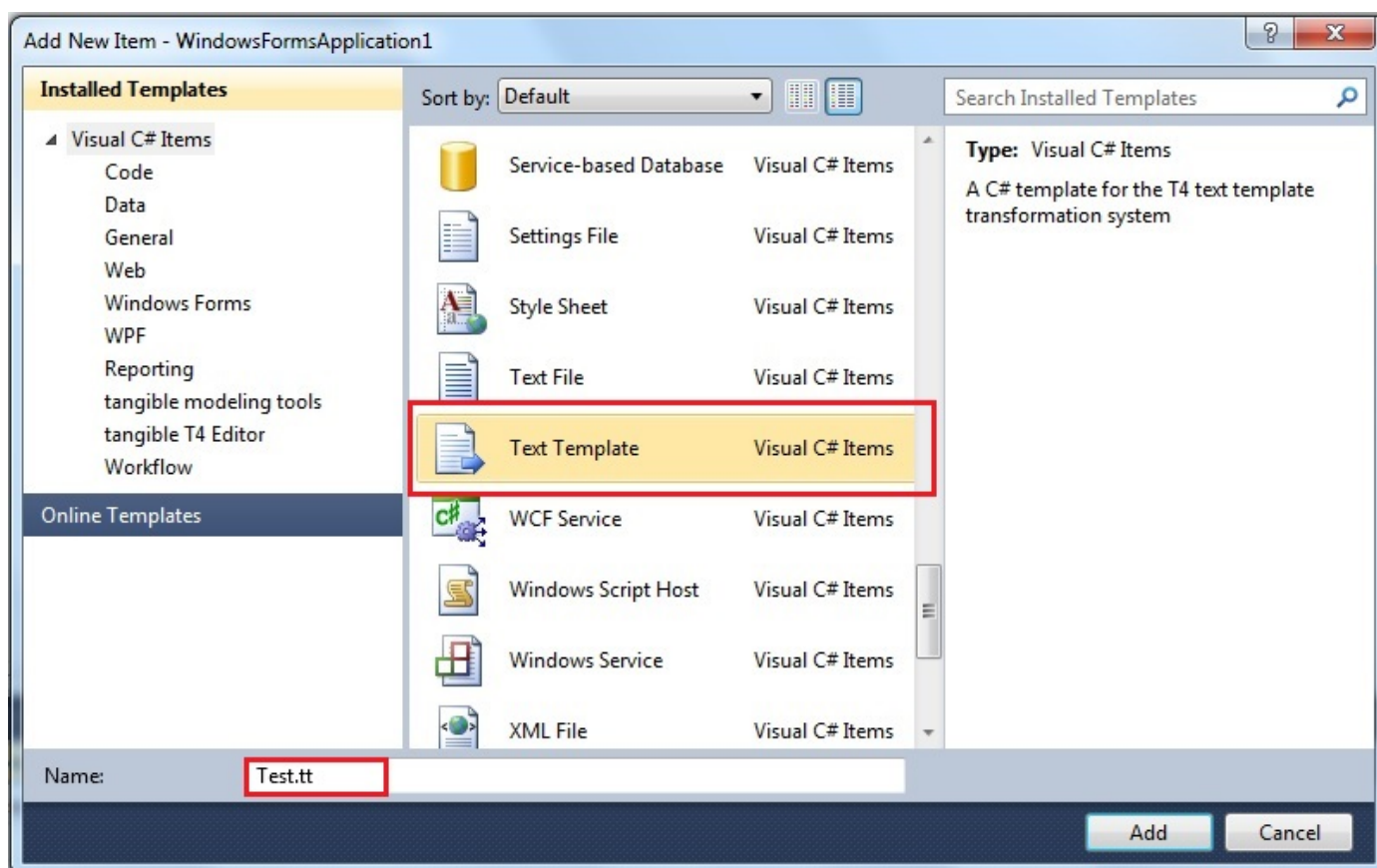


یکی از امکانات کمتر شناخته شده در دات نت، امکان تولید اتوماتیک کد (Code Generator)، توسط فایل هایی به عنوان Text template می باشد. اگر چه فایل های Text template با پسوند tt از Visual Studio 2008 بطور آشکار به IDE اضافه گردیده اند، این امکان پیش از این نیز بصورت توکار در Visual Studio جهت تولید کد دات نت برای ابزارهایی مانند Report ، Dataset و ... وجود داشته است. در حال حاضر Visual Studio بصورت توکار از این امکان برای تولید کلاس ها و کدهای EntityFramework (dbml ، Dataset و edmx ... استفاده می نماید. شما نیز با دانستن قواعد ساده کد نویسی برای Text template ها می توانید از این امکان برای تولید اتوماتیک کلاس ها، XML ، HTML و بطور کلی هر نوع فایل متنی استفاده نمایید. کاربردهای عملی Text template بیشتر برای تولید کد از روی دیتابیس و مثلاً بر اساس فیلدهای هر جدول و امثال آن می باشد. اما با کمی خلاقیت استفاده های بسیار زیاد و جالبی را می توانید از آن مشاهده نمایید.

کاربردهایی مانند : تولید خود کار فایل Config، تولید خودکار Unit Test، تولید خودکار کلاس های CRUD برای هر موجودیت در لایه Data، تولید خودکار SQL برای StoredProcedure ها، تولید خودکار Html و Js، تولید خودکار فرم های ASPX و ... برای فرم های ثابت و تکراری با فرآیند مشخص، تولید خودکار مستندات پروژه در قالب Word Document ، ... ، تولید خودکار فایل های Excel از اطلاعات خاص و تولید خودکار فرم های xaml و

در این آموزش با قواعد اصلی نوشتن کد برای Text template ها آشنا می شویم و چند نمونه از کاربردهای آن را به عنوان مثال کاربردی مورد بررسی قرار خواهیم داد. برای شروع قبل از توضیح در مورد قواعد کد نویسی Text template، یک فایل Text template ایجاد نمایید. برای ایجاد، از پنجره Add New Item یک فایل Text template به پروژه اضافه نمایید:



توجه داشته باشید که نام فایل خروجی دقیقا هم نام با فایل tt مشخص شده خواهد بود. اما پسوند آن بسته به تنظیمات داخل آن متفاوت است. با اضافه کردن Text template داخل فایل، باید کدهای زیر را مشاهده نمایید؛ در غیر این صورت آن را بنویسید:

```
<#@ template debug="false" hostspecific="false" language="C#" #>
<#@ output extension=".txt" #>
```

Language زبانی را مشخص می‌کند که می‌خواهید با آن داخل فایل Text template کد نویسی نمایید. تعیین آن برای کامپایل الزامی است.

Extension نیز پسوند فایل خروجی تولید شده را مشخص می‌نماید .

اگر در ادامه متن Hello dotnettips را بنویسید و فایل را Save کنید، کنار فایل tt مذکور، یک فایل با همان نام و پسوند txt ایجاد خواهد شد که داخل آن نوشته است:

Hello dotnettips

برای ایجاد فایل خروجی همچنین می‌توانید روی فایل tt کلیک راست نموده و از منوی باز شده Run Custom Tool را انتخاب نمایید. توجه داشته باشید که در تنظیمات Custom Tool باید مقدار TextTemplatingFileGenerator وجود داشته باشد:



خوب؛ برای ادامه، باید با قواعد کد نویسی در Text template آشنا شوید. جلسه بعدی قواعد کد نویسی T4 را بررسی خواهیم کرد.

نظرات خوانندگان

نویسنده: خلوت گزیده
تاریخ: ۱۹:۲۹ ۱۳۹۲/۱۲/۱۱

ممنون دوست عزیز. فقط اگه میشه در مورد گرفتن خروجی توضیح بیشتری بدی ممنون میشم. من موفق به گرفتن خروجی نشدم.

نویسنده: محسن خان
تاریخ: ۱۹:۴۷ ۱۳۹۲/۱۲/۱۱

این مطلب 3 قسمت دیگه هم داره. برای دنبال کردن آن [گروه T4](#) رو که ذیل مطلب لینکش هست بهتره پیگیری کنید.

بعد از ایجاد فایل Text template که [در جلسه قبل](#) با آن آشنا شدید، برای شروع قواعد زیر را در نظر بگیرید :

- تنظیمات مربوط به فایل Text template و نحوه تولید خروجی در ابتدای فایل و بین علامت <#@> و <#> قرار میگیرد.

- هر متنی که بصورت معمول در فایل tt نوشته شود، به همان صورت در فایل خروجی قرار می‌گیرد.

- هر دستوری که در بین علامت‌های <#> و <#> قرار گیرد هنگام کامپایل اجرا شده و معادل آن در همان مکان متن قرار میگیرد.

- هر دستوری که بین علامت‌های <#> و <#> قرار گیرد، هنگام کامپایل اجرا می‌شود. در این صورت دستورات نوشته شده در این

قسمت فقط اجرا می‌گردد و معمولاً برای استفاده در قسمت‌های دیگر، داخل بلوک <#=#> نوشته می‌شود .

- برای تعریف کلاس یا متد جدید جهت استفاده در فایل tt می‌توانیم کلاس را در بین علامت <#+> و <#> قرار دهیم. در این

صورت کلاس و متدهای نوشته شده در قسمت‌های دیگر، داخل بلوک <#=#> و یا <#> مورد استفاده قرار میگیرند.

اجازه دهید با یک مثال ساده قواعد اولیه را بررسی کنیم :

```
<#@ template debug="false" hostspecific="false" language="C#" #>
<#@ output extension=".txt" #>

<# var T = DateTime.Now; #>

The Time is : <#= T #>

The Time is : <#= DateTime.Now #>
```

در این مثال، T در واقع متغیری است که در بلوک <#> تعریف گردیده و در بلوک <#=#> مقدار آن استفاده میشود. خروجی فایل چیزی شبیه به دو خط زیر خواهد بود:

```
The Time is : 02/16/2014 14:17:39
```

```
The Time is : 02/16/2014 14:17:39
```

به عنوان یک مثال دیگر که قواعد توضیح داده شده را پوشش دهد به مثال زیر توجه کنید :

```
<#@ template debug="true" hostspecific="false" language="C#" #>
<#@ output extension=".cs" #>

using System;
using System.Text;

<# string ClassName = "DotnetTips"; #>
public class <#= ClassName + "_" + new MyTestClass().Str #>
{
}

<#+
public class MyTestClass
{
public string Str { get{return new DateTime().DayOfWeek.ToString(); } }
}
#>
```

خروجی Text template بالا فایل Cs با محتوی شبیه کد زیر خواهد بود: (روز نگارش مطلب البته دوشنبه است)

```
using System;
using System.Text;
```

```
public class DotnetTips_Monday
{
}
```

به عنوان یک مثال ساده دیگر برای فهم بیشتر به کد زیر جهت تولید Table در Html توجه کنید:

```
<#@ template debug="false" hostspecific="false" language="C#" #>
<#@ output extension=".html" #>

<html><body>
<table>
  <# for (int i = 1; i <= 10; i++)
  { #>
    <tr>
      <td>Test name <#= i #> </td>
      <td>Test value <#= i * i #> </td>
    </tr>
  <# } #>
</table>
</body></html>
```

فکر می‌کنم این 3 مثال ساده، تا حد زیادی قواعد اولیه T4 Text Template را برای شما روشن کرده باشد. در قسمت بعدی برخی قواعد تکمیلی را در این مورد خدمتتون ارائه میدم.

نظرات خوانندگان

نویسنده: افشین عباسپور
تاریخ: ۱۳۹۲/۱۱/۳۰ ۲۳:۲۳

لینک کدهای این قسمت در ورد رو میتونید از [اینجا](#) دانلود کنید . چون رنگ کدها رو میتوند [اونجا](#) ببینید .

خوب در دو قسمت قبلی ([۱](#) و [۲](#)) با T4 و قواعد کد نویسی Text Template آشنا شدید. در این قسمت برخی مفاهیم را با یک مثال کاربردی تر بررسی می‌کنیم.

در ادامه قواعد زیر را در نظر بگیرید :

- 1 - برای استفاده از یک کتابخانه خارجی (dll) در داخل کد Text Template از بلوک `<# assembly @#>` استفاده می‌شود .
مثلا برای استفاده از System.xml کد `<# "assembly name="System.xml" @#>` رو قرار بدید .
- 2- برای import کردن یک فضای نام نیز می‌توانید از بلوک `<# @#>` استفاده نمایید .
به عنوان مثال : `<# "import namespace="System.Data" @#>`
- 3- encoding خروجی قابل تنظیم می‌باشد. مثال: `<# output extension = ".html" encoding = "utf-8" @#>`
- 4- برای استفاده از یک فایل tt درون فایل دیگر میتوان از include استفاده کرد. مثال : `<# include file=" testpath " @#>`

`<# \basetest.tt`

حالا به مثال زیر توجه کنید:

در بسیاری از پروژه‌ها، معادل تمامی جداول موجود در یک دیتابیس Class هایی به نام DTO یا Data transfer object ساخته میشود که عموماً "کلاسهای سبکی هستند که فقط شامل خصوصیت‌های معادل فیلدهای جداول می‌باشند و از آن‌ها جهت مدل کردن داده‌ها و ... استفاده می‌گردد. تولید این کلاسهای ساده می‌تواند بصورت اتوماتیک صورت گیرد و از این جهت در زمان تولید پروژه صرفه جویی شود. همچنین با تغییر ساختار دیتابیس می‌توان همواره کلاسها را بروزرسانی کرد. نمونه ای از کد T4 برای تولید تمامی کلاسهای DTO به شکل زیر میباشد . فقط برای تست کد زیر دقت کنید که ConnectionString را در داخل کد متناسب با دیتابیس خود تغییر دهید:

```
<#@ template language="C#" debug="True" hostspecific="True" #>
<#@ output extension=".cs" #>
<#@ assembly name="System.Data" #>

<#@ assembly name="System.xml" #>
<#@ import namespace="System.Collections.Generic" #>
<#@ import namespace="System.Data.SqlClient" #>
<#@ import namespace="System.Data" #>

using System;
namespace MyProject.Entities
{
    <#
        string connectionString = "Password=22125110;Persist Security Info=True;User ID=sa;Initial
Catalog=DnnDB;Data Source=ABBASPOOR299";
        SqlConnection conn = new SqlConnection(connectionString);
        conn.Open();
        System.Data.DataTable schema = conn.GetSchema("TABLES");
        string selectQuery = "select * from @tableName";
        SqlCommand command = new SqlCommand(selectQuery,conn);
        SqlDataAdapter ad = new SqlDataAdapter(command);
        System.Data.DataSet ds = new DataSet();

        foreach(System.Data.DataRow row in schema.Rows)
        {
            <#
            public class <#= row["TABLE_NAME"].ToString().Trim('s') #>
            {
                <#
                command.CommandText =
selectQuery.Replace("@tableName",row["TABLE_NAME"].ToString());
                ad.FillSchema(ds, SchemaType.Mapped, row["TABLE_NAME"].ToString());

                foreach (DataColumn dc in ds.Tables[row["TABLE_NAME"].ToString()].Columns)
                {
                    <#
                }
            }
        }
    }
}
```

```
private <# dc.DataType.Name #> _<# dc.DataType.Name #>
dc.ColumnName.Replace(dc.ColumnName[0].ToString(), dc.ColumnName[0].ToString().ToLower()) #>;
public <# dc.DataType.Name #> <# dc.ColumnName #>
{
    get { return _<# dc.ColumnName.Replace(dc.ColumnName[0].ToString(),
dc.ColumnName[0].ToString().ToLower()) #>; }
    set { _<# dc.ColumnName.Replace(dc.ColumnName[0].ToString(),
dc.ColumnName[0].ToString().ToLower()) #> = value; }
}

<# } #>

}

<#
} #>

}
```

خوب! خودتون می‌توانید تست کنید ... این یکی از رایج‌ترین استفاده‌های T4 Text Template هست که برنامه نویسان از آن استفاده می‌کنند.

نظرات خوانندگان

نویسنده: افشین عباسپور
تاریخ: ۲۳:۲۵ ۱۳۹۲/۱۱/۳۰

متن رنگی کد این قسمت رو میتونید از [اینجا](#) دانلود کنید

نویسنده: علی صداقت
تاریخ: ۱۲:۱۴ ۱۳۹۲/۱۲/۰۳

با تشکر از مطلب مفید شما. ظاهرا پس از اتمام درج اطلاعات ستونهای یک جدول باید متد `ds.Tables.Clear` را فراخوانی کرد تا جداول موجود در دیتاست پاک شوند. در حال حاضر فیلدهای جدول اول به جای فیلدهای تمامی جداول قرار می گیرند.

نویسنده: افشین عباسپور
تاریخ: ۱۵:۲ ۱۳۹۲/۱۲/۰۳

ممنون دوست عزیز و پوزش از بی دقتی من .
مشکل `Clear` شدن دیتاست نیست . حلقه دو باید تغییر کنه `ds.Tables[0].Columns ...` باید تغییر کنه و تبدیل به `ds.Tables[row["TABLE_NAME"].ToString()].Columns` بشه ! باز هم ممنون از دقت شما

نویسنده: افشین عباسپور
تاریخ: ۱۵:۳۲ ۱۳۹۲/۱۲/۰۳

کد اصلاح شد .

نویسنده: ابوالفضل علیاری
تاریخ: ۲۳:۴۳ ۱۳۹۲/۱۲/۱۶

سلام
من کامل متوجه نشدم ، وقتی برنامه اجرا میشه این کدها به صورت اتوماتیک اجرا میشه؟
ممنون

نویسنده: افشین عباسپور
تاریخ: ۱۴:۱ ۱۳۹۲/۱۲/۱۷

نه دوست عزیز .. این به اجرای برنامه ربطی نداره ! وقتی فایل رو (فایل `tt`) در ویژوال استدیو تغییر میدید و `Save` میکنید بطور اتوماتیک اجرا میشه و خروجی رو در فایل تعیین شده در `output` (بالای کد) ایجاد میکنه ... قسمت چهارم رو مطالعه کنید بیشتر متوجه میشید
ضمن اینکه مثال ساده قسمت دوم رو اجرا کنید تا متوجه عملکرد فایل [Text template](#) بشی ...

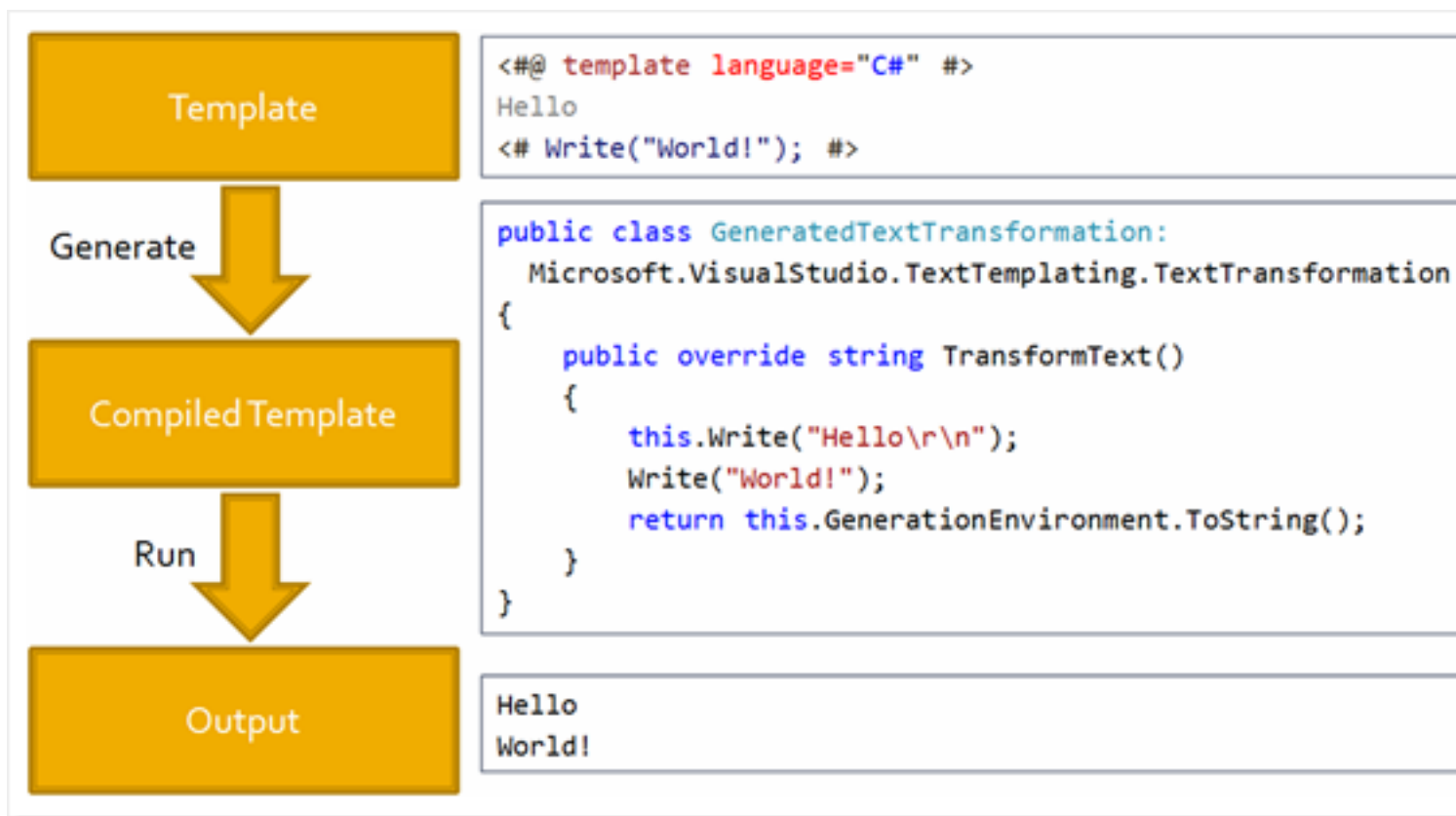
نویسنده: افشین عباسپور
تاریخ: ۱۴:۳ ۱۳۹۲/۱۲/۱۷

در مورد اجرای مثال قسمت سوم هم دقت کنید که `connectionString` رو باید متناسب دیتابیس مورد نظر خودتون در کامپیوتر تغییر بدید بعد نتیجه جادویی رو ببینید !

در قسمت‌های قبلی ([^](#) و [^](#) و [^](#)) با Text Template در Visual Studio آشنا شدید. این قسمت برای تکمیل بحث در مورد ابزاری که Microsoft از آن در برنامه‌های خود از جمله Visual Studio جهت تولید کدهای اتوماتیک استفاده می‌نماید، صحبت خواهیم کرد.

قبل از آن بد نیست که بدانید چرا این ابزار T4 نام گرفته !

T4 مخفف Text Template Transformation Toolkit می‌باشد (TTTT). شکل زیر مراحل اجرای یک کد Text Template را توسط T4 نشان می‌دهد:



پس این ابزار، یک ابزار کاربردی می‌باشد که بدون Visual Studio نیز میتوان از آن استفاده کرد. نام فایل این ابزار، TextTransform.exe است و در مسیر زیر وجود دارد :

Program Files (x86)\Common Files\microsoft shared\TextTemplating\10.0

برای اطلاع از نحوه کار با TextTransform.exe خارج از محیط Visual Studio بهتر است دستور زیر را در cmd.exe اجرا کنید تا راهنمای استفاده و پارامترهای اختیاری آن را مشاهده نمایید:

TextTransform.exe -h

برای آزمایش، یک فایل متنی کنار فایل TextTransform.exe با نام Text2.tt ایجاد نمایید و کد زیر را در داخل آن بنویسید:

```
<#@ template debug="true" hostspecific="false" language="C#" #>
<#@ output extension=".txt" #>

<#@ import namespace="System.Diagnostics" #>

Report In : <#= DateTime.Now #>

<#
Process[] Procs = Process.GetProcesses();
for (int i = 0; i < Procs.Length; i++)
{
    string Pstr = Procs[i].ProcessName + " -|- " + Procs[i].Id + Environment.NewLine ;
    #><#= Pstr #><#
}
#>
```

این مثال بعد از اجرا، لیست تمام Process های جاری سیستم را به همراه Id آنها، چاپ می‌نماید.
برای تولید فایل خروجی، دستور زیر را در cmd.exe اجرا کنید :

```
TextTransform.exe -out Report1.txt Text2.tt
```

توجه کنید که فایل Text2.tt را کنار فایل TextTransform.exe قرار دهید و بعد از اجرای دستور بالا، باید خروجی در فایل Text2.txt در همان مسیر ایجاد گردد.

نکته: اگر User شما به این پوشه دسترسی ندارد و کاربر Admin نیستید احتمالاً به مشکل بر می‌خورید. می‌توانید فایل TextTransform.exe را در مکان دیگری قرار دهید و دستور را از آن محل اجرا کنید و یا برای پوشه‌ی مذکور دسترسی ایجاد نمایید.

اگر می‌خواهید بیشتر در مورد معماری T4 بدانید بهتر است مقاله زیر را مطالعه کنید:

<http://www.olegrych.com/2008/05/t4-architecture>

نکته دیگر این که برای Visual Studio، ابزارهایی جهت بهبود کار با Text Template ها وجود دارند که با جستجوی T4 Editor، نمونه‌هایی از آنها را خواهید یافت. tangible T4 Editor نمونه ای از این Plugin ها می‌باشد که به Visual Studio افزوده می‌گردد و یا یک پروژه Open Source هم برای آشنایی بسیار بیشتر با T4 در t4toolbox.codeplex.com وجود دارد که می‌توانید مشاهده کنید.

نظرات خوانندگان

نویسنده: افشین عباسپور
تاریخ: ۱۳۹۲/۱۱/۳۰ ۲۳:۲۶

متن کد این قسمت هم میتونید از [اینجا](#) دانلود کنید

بدون شک دوستانی که با تکنولوژی محبوب ASP.NET MVC5 کار کرده اند این نکته را می‌دانند که اگر فایل‌های T4 که وظیفه Scaffolding را به عهده دارند به پروژه خود اضافه کنند می‌توانند نحوه تولید خودکار Controller ها و View های متناظر را سفارشی کنند. مثلاً می‌توان این فایل‌ها را طوری طراحی کرد که Controller و View های تولیدی به طور اتوماتیک چند زبانه و یا Responsive تولید شوند (این موضوعات بحث اصلی مقاله نیستند) و اما بحث اصلی را با یک مثال آغاز می‌کنیم:

فرض کنید در دیتابیس خود یک Table دارید که قرار است اطلاعات یک Slider را در خود نگه دارد. این Table دارای یک فیلد از نوع nvarchar برای ذخیره آدرس تصویر ارسالی توسط کاربر است.

در حالت عادی اگر از روی مدل این Table اقدام به تولید خودکار Controller و View متناظر کنید، یک editor (تکست باکس) برای دریافت آدرس تصویر تولید خواهد شد که برنامه نویسی یا طراح باید به طور دستی آن را (به طور مثال) با Kendo uploader جایگزین نماید. ما می‌خواهیم برای فیلدهایی که قرار است آدرس تصویر را در خود نگه دارد به طور اتوماتیک از Kendo uploader استفاده شود. راه حل چیست؟

بسیار ساده است. ابتدا باید در نظر داشت که هنگام طراحی Table در دیتابیس فیلد مورد نظر را به این شکل نامگذاری کنید:

ExampleIMGURL (نحوه نام گذاری دلخواه است) مقصود آن است که نام هر فیلدی که قرار است آدرس یک تصویر را در خود نگه دارد باید حاوی کلمه (IMGURL) باشد. مجدداً ذکر می‌شود که نحوه نامگذاری اختیاری است. سپس فایل Create.t4 را باز کنید و کد:

```
@Html.EditorFor(model => model.<#= property.PropertyName #>)
```

را با کد زیر جایگزین کنید:

```
<#
if (GetAssociationName(property).Contains ("IMGURL"))
{
#>
    @Html.Kendo().Upload().Name("<#= property.PropertyName #>")
<#
}
else
{
#>
    @Html.EditorFor(model => model.<#= property.PropertyName #>)
<#
}
#>
```

کد بالا چک می‌کند اگر نام فیلد مد نظر حاوی " IMGURL " باشد یک کندو آپلودر تولید کرده در غیر این صورت یک ادیتور ساده تولید می‌کند. البته این فقط یک مثال است و بدون شک دامنه استفاده از این تکنیک وسیع‌تر است.

اگر این مطلب مفید واقع شد با در نظر گرفتن نظرات ارسالی به تکنیک‌های آتی اشاره خواهد شد.

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۲/۱۰ ۱۱:۳۰

قابلیت سفارشی سازی EditorFor در ASP.NET MVC پیش بینی شده است و [با استفاده از UIHint](#) قابل انتساب به خواص مدل مورد نظر است. البته این مورد برای حالت Code first یا حالتیکه از [ViewModels](#) استفاده کنید بیشتر کاربرد دارد. یک مثال:

فایلی را به نام Upload.cshtml ، در مسیر Views/Shared/EditorTemplates با محتوای ذیل ایجاد کنید:

```
@model string
@Html.Kendo().Upload().Name("@ViewData.ModelMetadata.PropertyName")
```

سپس برای استفاده از آن فقط کافی است خاصیت مدنظر را با ویژگی UIHint مزین کنید:

```
[UIHint("Upload")]
public string ImageUrl {set;get;}
```

نویسنده: صادق نجاتی
تاریخ: ۱۳۹۳/۰۲/۲۹ ۱۱:۵

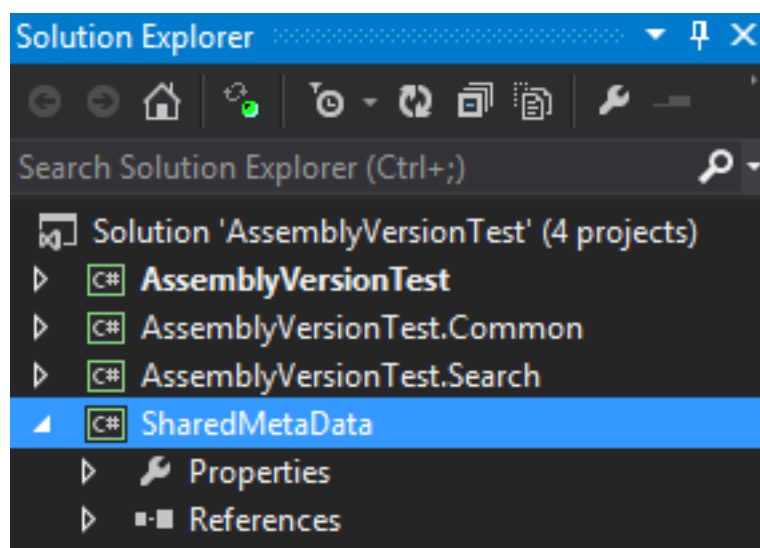
ضمن تشکر از آقای نصیری؛

بدون شک نقش UIHint در سفارشی سازی انکار ناپذیر است. ولی همانطور که گفته شد دامنه استفاده از این تکنیک وسیع تر است. مثلا حالتی را در نظر بگیرید که می خواهیم از طریق Scaffolding برای یک جدول بانک اطلاعاتی که یک فیلد آن آدرس یک تصویر را نگهداری می کند View ایجاد نماییم. خوب ما در صفحه Index می خواهیم تصویر مورد نظر با اندازه 100 * 100 پیکسل نمایش دهیم (چون قرار است لیستی از تصاویر نمایش داده شود باید در اندازه قابل نمایشی باشد) ولی در صفحه Details باید اندازه بزرگتری از تصویر را به نمایش بگذاریم. حال اگر از UIHint استفاده کنیم تنها یکی از موارد قبل (سفارشی سازی در لیست و جزئیات) محقق خواهد شد. اگر بخواهیم انجام این کارها را به صورت اتوماتیک به Scaffolding بسپاریم باید مطابق آنچه گفته شد ، فایل های T4 را (List.t4 و Details.t4) سفارشی سازی نماییم.

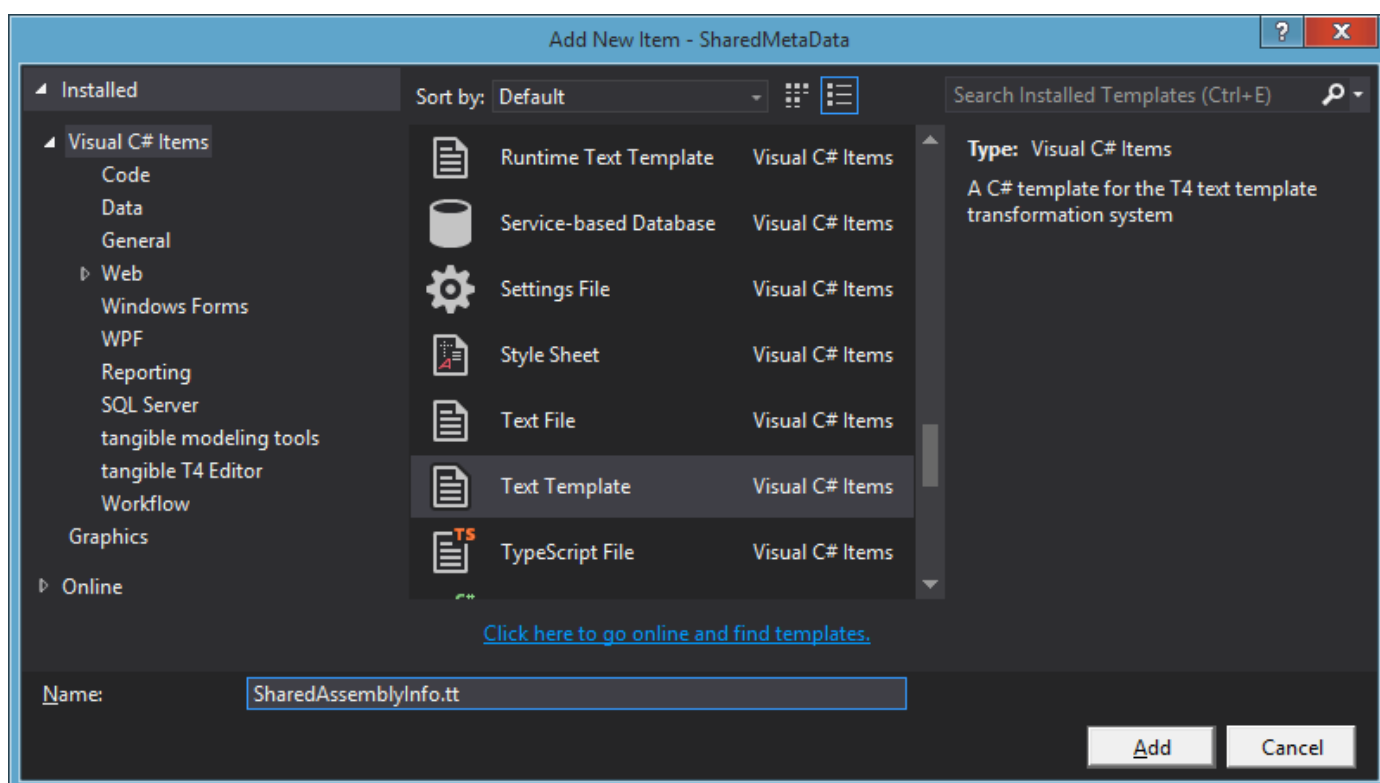
عموماً برای نگهداری ساده‌تر قسمت‌های مختلف یک پروژه، اجزای آن به اسمبلی‌های مختلفی تقسیم می‌شوند که هر کدام در یک پروژه‌ی مجزای ویژوال استودیو قرار خواهند گرفت. یکی از نیازهای مهم این نوع پروژه‌ها، داشتن شماره نگارش یکسانی بین اسمبلی‌های آن است. به این ترتیب توزیع نهایی ساده‌تر شده و همچنین پشتیبانی 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 این کار، 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

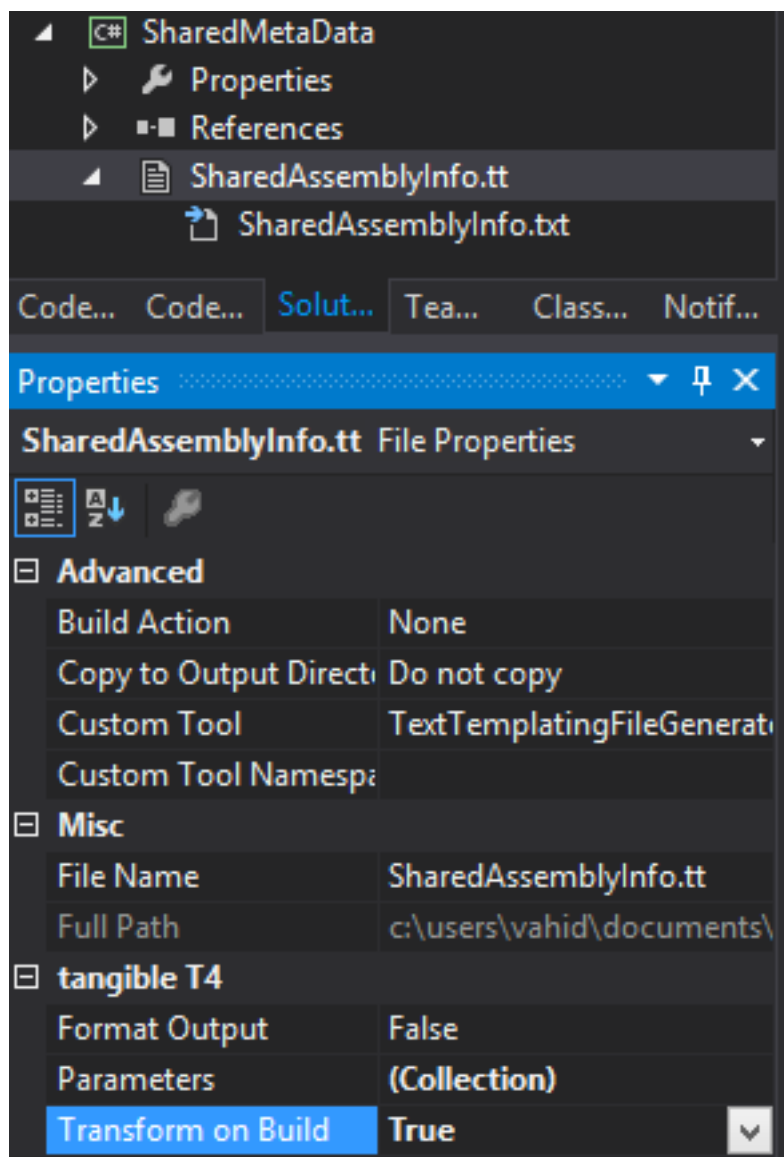
برای نگهداری فایل مشترک SharedAssemblyInfo.cs نهایی و همچنین اطمینان از تولید مجدد آن به ازای هر Build، یک پروژه‌ی class library جدید را به نام SharedMetadata به Solution جاری اضافه کنید.



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](#) آن را [true](#) کنید. به این ترتیب مطمئن خواهیم شد این فایل به ازای هر build جدید، مجدداً 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.

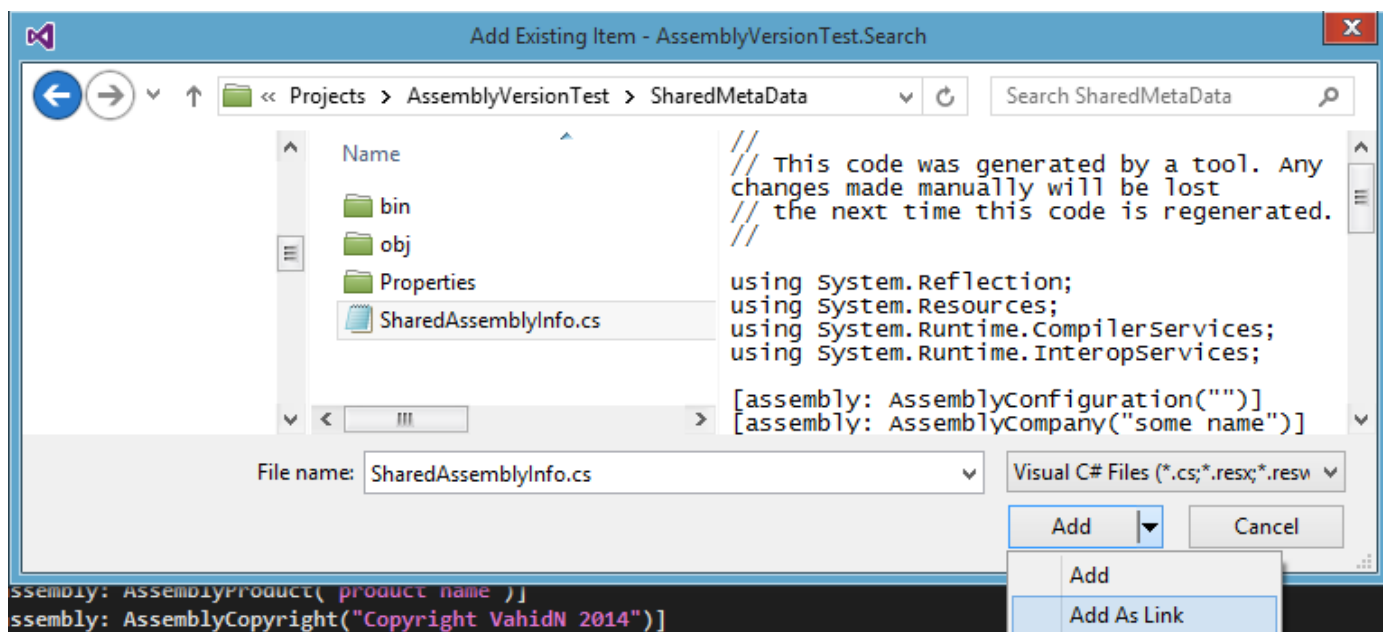
BuildNumber به صورت خودکار بر اساس هر Build انجام شده باید تغییر یابد. در اینجا این عدد به صورت خودکار به ازای هر روز، یک واحد افزایش پیدا می‌کند. ابتدای مبداء آن در این مثال، 2013 قرار گرفته‌است.

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 صرفاً لینک شده‌است، اگر محتوای آن در پروژه‌ی metadata تغییر کند، به صورت خودکار و یک دست، در تمام پروژه‌های دیگر نیز منعکس خواهد شد.

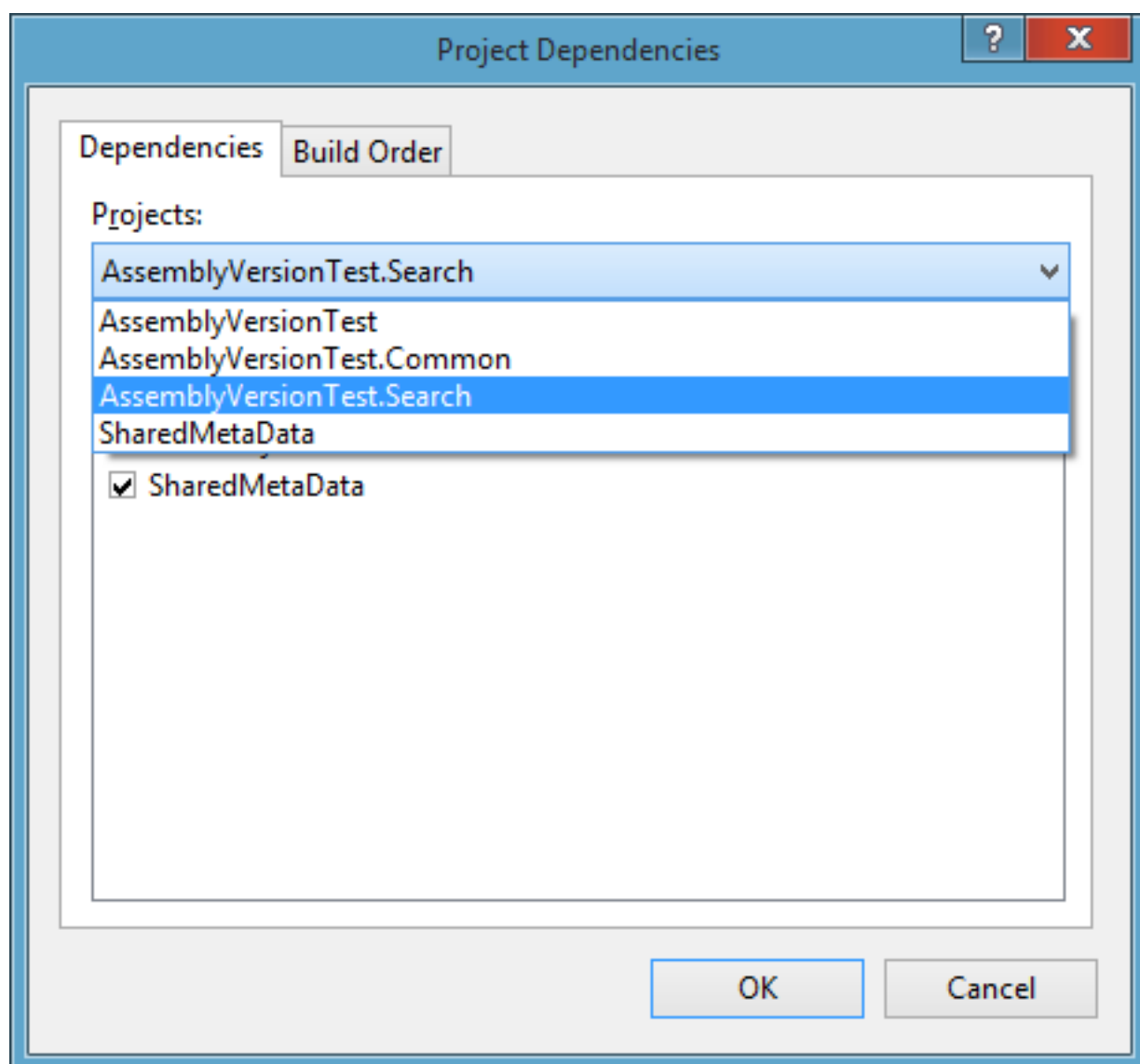
در ادامه اگر بخواهید Solution را Build کنید، پیام تکراری بودن یک سری از ویژگی‌ها را یافت خواهید کرد. این مورد از این جهت رخ می‌دهد که هنوز فایل‌های AssemblyInfo.cs اصلی، در پروژه‌های برنامه موجود هستند. این فایل‌ها را یافته و صرفاً چند سطر همیشه ثابت ذیل را در آن‌ها باقی بگذارید:

```
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 نمایید.



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:

Name	File version	Product version
 AssemblyVersionTest.Common.dll	1.0.645.0	1.0.645.0
 AssemblyVersionTest.Common.pdb		
 AssemblyVersionTest.exe	1.0.645.0	1.0.645.0
 AssemblyVersionTest.pdb		
 AssemblyVersionTest.Search.dll	1.0.645.0	1.0.645.0
 AssemblyVersionTest.Search.pdb		
 AssemblyVersionTest.vshost.exe	12.0.30723.0	12.0.30723.0
 AssemblyVersionTest.vshost.exe.manifest		

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 ۱۳:۰۰

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.