عنوان: **MSBuild**
نویسنده: یوسف نژاد
تاریخ: ۱۳۹۱/۱۱/۰۸ ۲۰:۵۰
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 کاملا 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:

- ویزوال 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 فرایندِ تولیدِ 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 نویس 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 buidlRequest = new BuildRequestData(projectFullPath, globalProperties, null, new string[] {
"Build" }, null);
  var buildResult = BuildManager.DefaultBuildManager.Build(new BuildParameters(pc), buidlRequest);
}
```

با اینکه ارائه مقداری غیرنال برای آرگومان globalProperties اجباری است اما پرکردن آن کاملا 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 csprj. یا vbprj. 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:

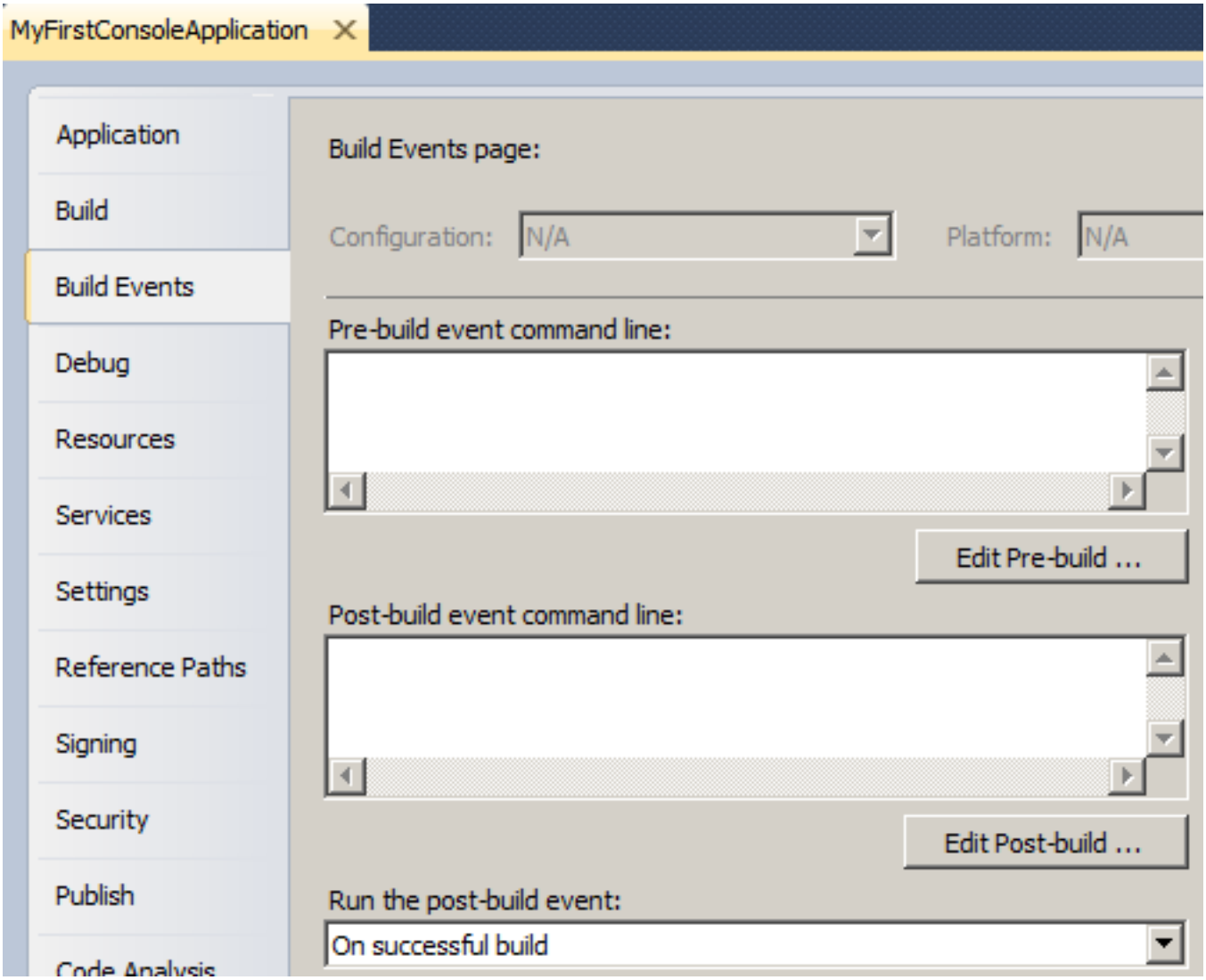
```
private static void TestMSBuild1(string projectPath)
{
  var regKey = Registry.LocalMachine.OpenSubKey(@"SOFTWARE\Microsoft\MSBuild\ToolsVersions\4.0");
  if (regKey == null) return;
  var msBuildExeFilePath = Path.Combine(regKey.GetValue("MSBuildToolsPath").ToString(), "MSBuild.exe");
  var startInfo = 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.

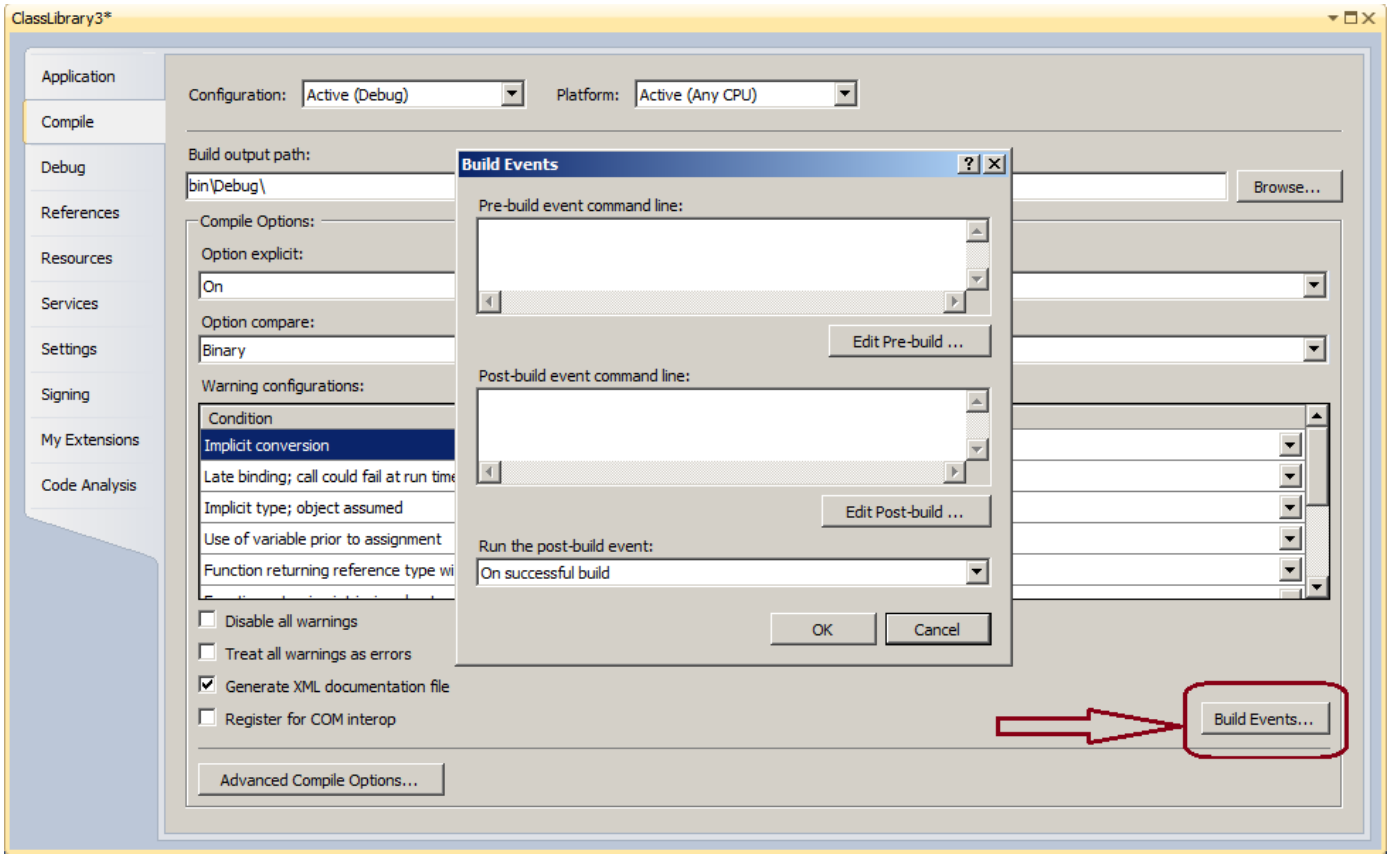
عنوان: **Build Events**
نویسنده: یوسف نژاد
تاریخ: ۱۳۹۱/۱۱/۱۱ ۲۳:۴۵
آدرس: www.dotnettips.info
برچسب‌ها: Visual Studio, MSBuild, Build Events

در ویژوال استودیو یک ویژگی جالب با عنوان **Pre/Post-Build Event** وجود دارد. این ویژگی به رویدادهای «قبل از بیلد» و «بعد از بیلد» اشاره دارد. از این ویژگی برای اجرای یکسری دستورات، قبل (Pre-build) یا بعد (Post-build) از عملیات بیلد استفاده میشود. دستوراتی که در این قسمت قابل اجرا هستند دقیقا همانند دستورات موجود در یک batch فایل میباشند. حتی میتوان یک فایل bat. را در این قسمت فراخوانی کرد. بطور خلاصه هرگونه دستوری که درون Command Prompt ویندوز یا در یک bat. فایل قابل اجرا باشد در این قسمت نیز قابل استفاده است. درنهایت تمام این دستورات توسط برنامه Cmd.exe اجرا میشوند.
نکته: قبل از ادامه بهتر است به این نکته اشاره کنم که مجموعه این دستورات چیزی فراتر از فراخوانی ساده یکسری فایل exe. هستند. درواقع کدی که در این قسمت به آن اشاره میشود، دارای ساختاری به صورت یک زبان برنامه نویسی ساده است. یعنی متن نهایی‌ای که برای اجرا به cmd.exe ارسال میشود میتواند شامل دستورات ساده و اولیه برنامه نویسی چون if .. then .. else و حلقه for و از این قبیل نیز باشد. برای آشنایی بیشتر با زبان این نوع دستورات به منابع زیر مراجعه کنید: Batch file
Using batch files

تنظیم رویدادهای بیلد (Build Events)
برای تنظیم این رویدادها باید به تب Build Events در صفحه پراپرتی‌های پروژه موردنظر مراجعه کنید. همانند تصویر زیر در یک پروژه کنسول #C:



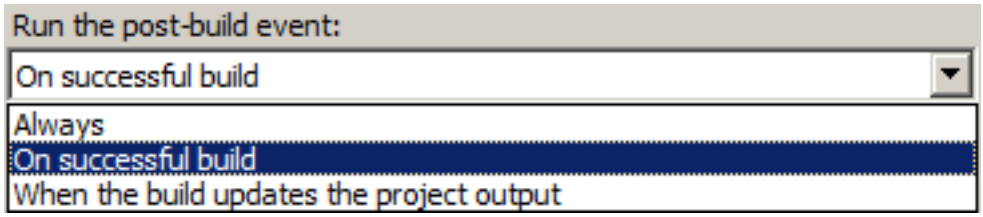
البته در پروژه‌های VB.NET مسیر منتهی به این قسمت کمی فرق میکند که در تصویر زیر نشان داده شده است:



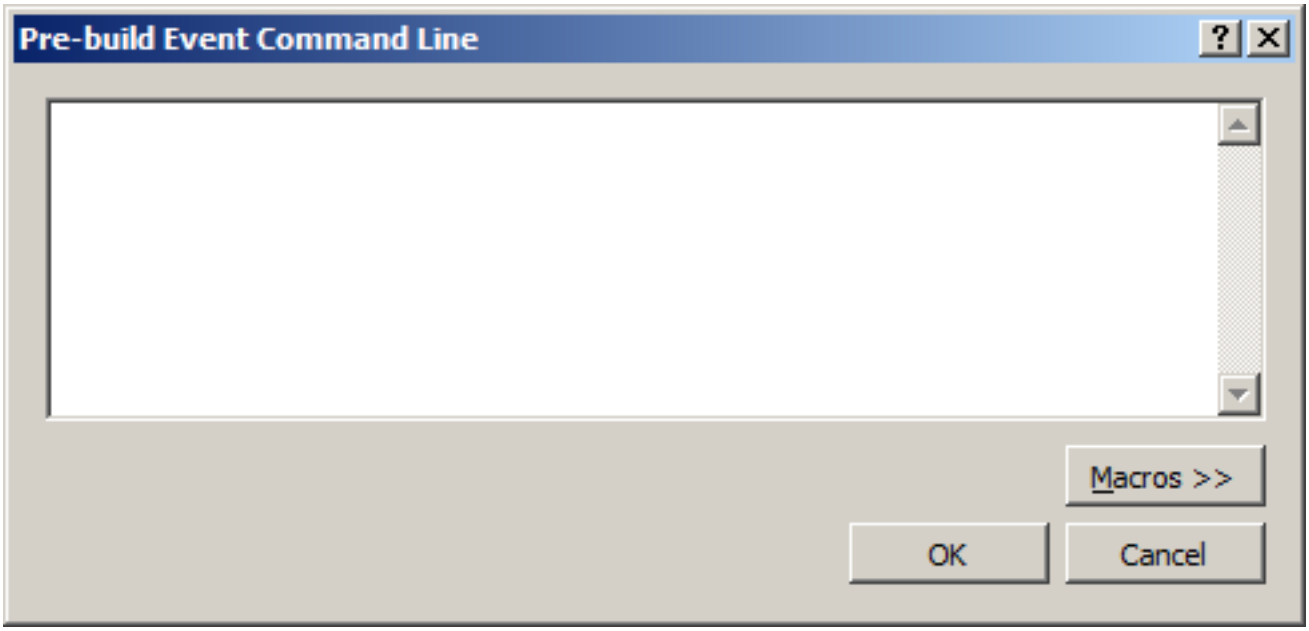
در پروژه‌های مربوط به زبانهای دیگر هم مسیر رسیدن به این رویدادها کمی متفاوت است. برای کسب اطلاعات بیشتر به اینجا مراجعه کنید.

در این قسمت میتوان همانند یک فایل batch دستورات موردنظر را در خطوط مجزا برای اجرا اضافه کرد. از این دستورات معمولا برای مدیریت عملیات بیلد، کپی فایلهای موردنیاز قبل یا بعد از بیلد، پاک کردن فولدرها، تغییر برخی تنظیمات با توجه به نوع کانفیگ بیلد (Debug یا Release)، ثبت یک اسمبلی در GAC و یا حتی اجرای برخی آزمونهای واحد و ... استفاده میشود.

نکته: درصورتیکه پروژه به روز باشد (یعنی ویژوال استودیو نیازی به تولید فایل اسمبلی نهایی پروژه به دلیل عدم وجود تغییری در کد برنامه نبیند) بدلیل عدم اجرای عملیات بید، دستورات قسمت Pre-build اجرا نمیشوند. اجرای دستورات قسمت Post-build نیز بستگی به تنظیمات قسمت Run the post-build events: همانند تصویر زیر دارد:



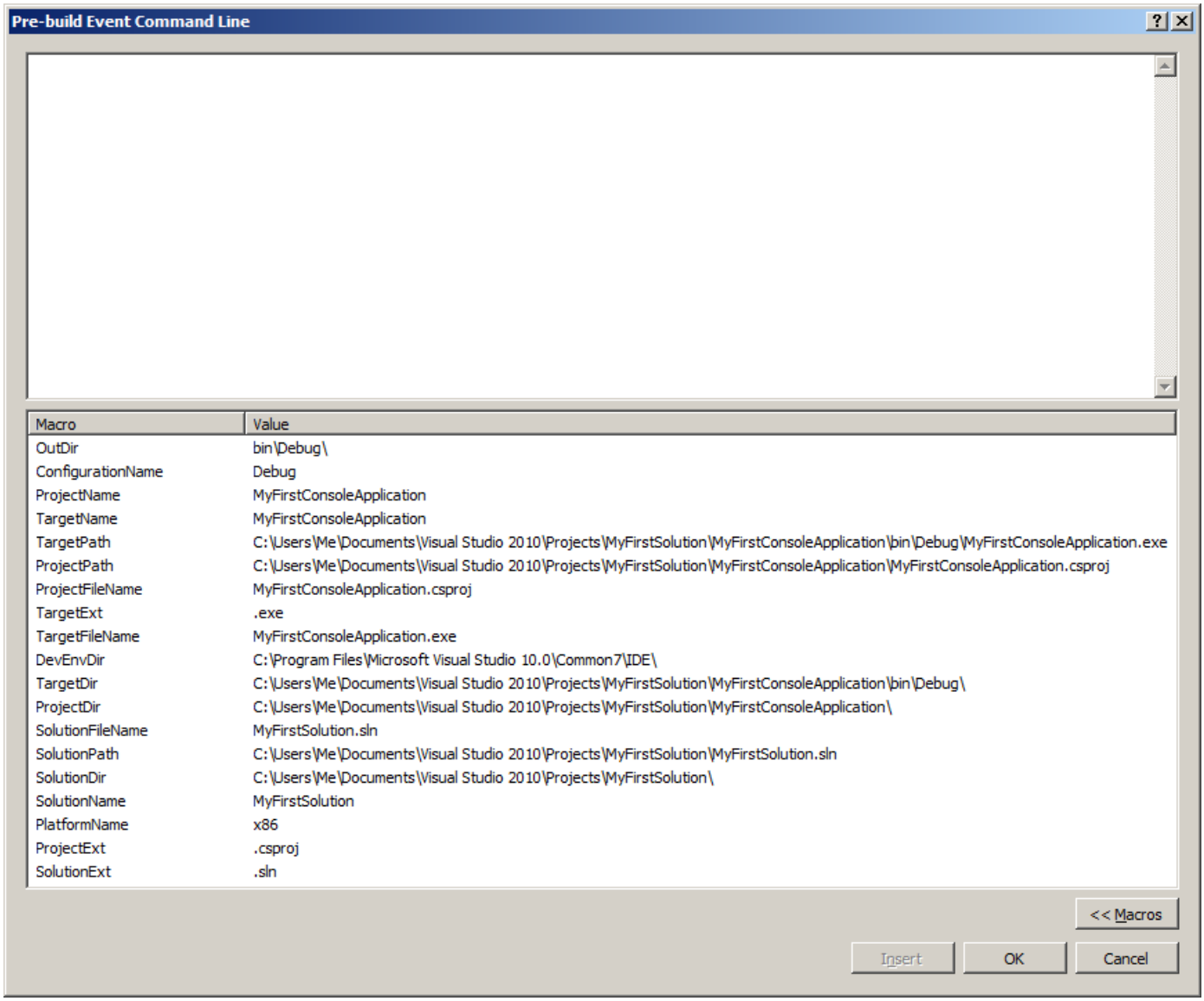
برای استفاده راحتتر از این ویژگی فرمی مخصوص وارد کردن این دستورات در ویژوال استودیو وجود دارد. برای دیدن این فرم بر روی دکمه Edit Pre-build... یا Edit Post-build... کلیک کنید. پنجره زیر نمایش داده میشود:



در این پنجره میتوان دستورات مورد نظر را وارد کرد. با اینکه هیچ امکان خاصی برای کمک به اضافه و ویرایش دستورات در این پنجره وجود ندارد! اما تنها ویژگی موجود در این فرم کمک بسیاری برای تکمیل دستورات موردنظر میکند. قبل از توضیح این ویژگی بهتر است با مفهوم Macro در این قسمت آشنا شویم.

Macro

در Build Events ویژوال استودیو یکسری متغیرهای ازقبل تعریف شده وجود دارد که به آنها Macro گفته میشود. برای مشاهده لیست این ماکروها روی دکمه Macro >> کلیک کنید. پنجره مربوطه به صورت زیر گسترش می‌یابد تا جدولی به نام Macro Table را نمایش دهد:



همانطور که مشاهده میکنید تعداد 19 ماکرو به همراه مقادیرشان در این قسمت به نمایش گذاشته شده است. برای استفاده از این ماکروها کافی است تا روی یکی از آنها دابل کلیک کنید یا پس از انتخاب ماکروی موردنظر روی دکمه Insert کلیک کنید. دقت کنید که نحوه نمایش این ماکروها در متن دستورات به صورت زیر است:

```
$(<Macro_Name>)
```

که به جای عبارت <Macro_Name> عنوان ماکرو قرار میگیرد. مثلا:

```
$(OutDir)    یا    $(ProjectName)
```

نکته: نام این ماکروها case-sensitive **نیست** .

نحوه اجرای دستورات توسط ویژوال استودیو

ویژوال استودیو برای اجرای دستورات کار خاصی به صورت مستقیم انجام نمیدهد! وظیفه اصلی برعهده MSBuild (^) است. این ابزار پس از جایگزین کردن مقادیر ماکروها، محتوای کل دستورات موجود در هر یک از رویدادها را در یک فایل batch ذخیره میکند و فایل مربوط به هر رویداد را در زمان خودش به اجرا میگذارد. مثلا دستور زیر را درنظر بگیرید:

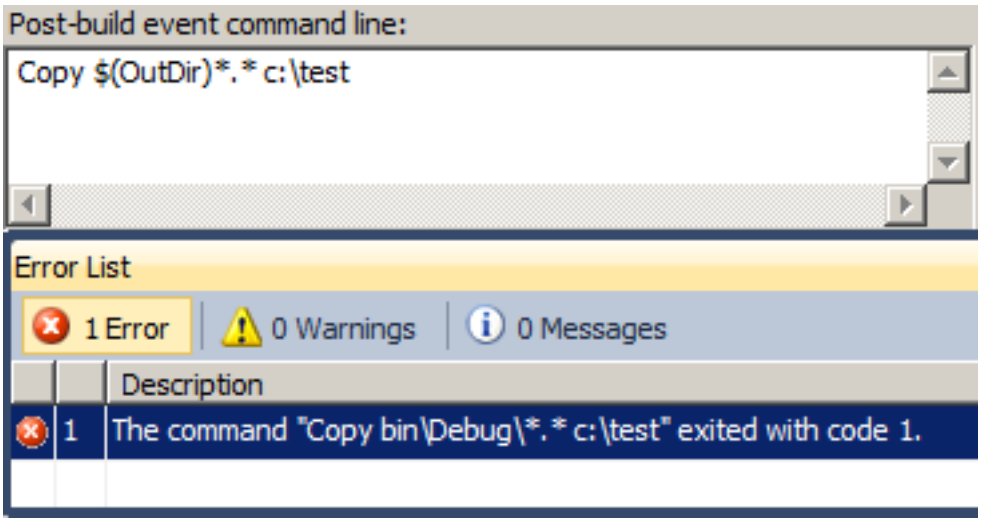
```
Copy $(OutDir)*.* %WinDir%
```

پس از ذخیره در فایل batch نهایی به صورت زیر در خواهد آمد:

```
Copy bin\Debug\*.* %WinDir%
```

نکته: در این زبان برنامه نویسی، عبارتی چون %WinDir% معرف یک متغیر است. در این مورد خاص این عبارت یک متغیر محیطی (Environment Variable) است. اطلاعات بیشتر در اینجا .

MSBuild عملیات اجرای این batch فایلهای تولیدی را زیر نظر دارد و هرگونه خطای موجود در این دستورات را به عنوان خطای زمان بیلد گزارش میدهد. اما از آنجاکه کل دستورات مربوط به هر رویداد درون یک فایل batch اجرا میشود، امکان گزارش محل دقیق خطای رخداده وجود ندارد. یعنی درصورتیکه مثلا تنها یکی از صدها خط دستور نوشته شده در این قسمت خطا بدهد تنها یک خطا و برای تمام دستورات نمایش داده میشود. البته همانطور که حدس میتوان حدس زد اجرای این دستورات ترنزکشنال نیست و اجرای تمامی دستورات تا قبل از وقوع خطا برگشت ناپذیر خواهند بود. برای نمونه به تصویر زیر و خطای نمایش داده شده دقت کنید:



نمونه اصلاح شده دستور فوق به صورت زیر است:

```
Copy "$(ProjectDir)$(OutDir)*.*" c:\test
```

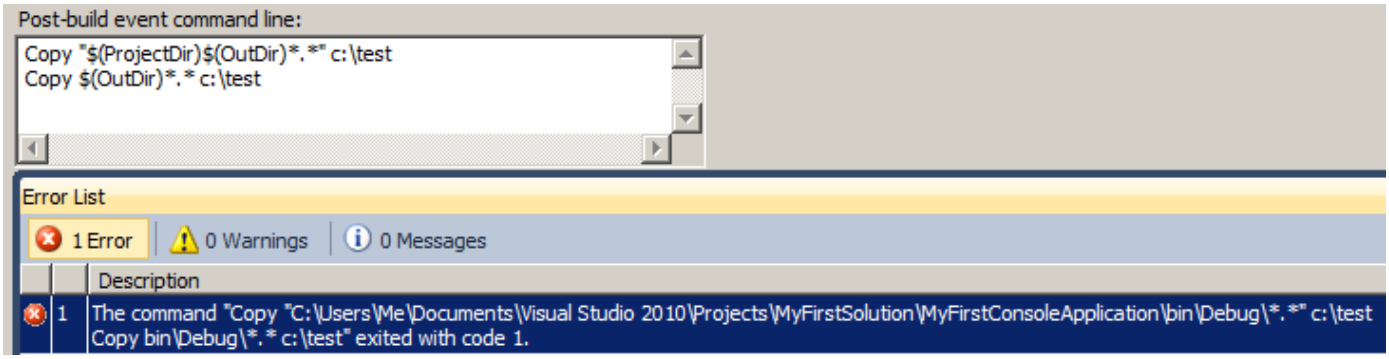
نکته: به دلیل استفاده از کاراکتر فاصله به عنوان جداکننده آرگومانها در دستورات DOS، وجود فاصله در مسیرهای مورد استفاده در این دستورات عملیات را دچار خطا خواهد کرد. راه‌حل استفاده از کاراکتر " در ابتدا و انتهای رشته‌های مربوط به مسیرها همانند دستور بالاست.

نکته: درصورت استفاده از یک فایل bat. برای ذخیره دستورات، امکان استفاده مستقیم از ماکروهای ویژوال استودیو درون آن وجود نخواهد داشت! یکی از راه‌حلها پاس کردن این متغیرها به صورت پارامتر در زمان فراخوانی فایل bat. است. مثلا:

```
"$(ProjectDir)postBuild.bat"   "$(SolutionPath)"
```

برای دریافت این پارامترهای پاس شده درون batch فایل باید از عبارات %1 برای پارامتر اول و %2 برای پارامتر دوم و ... تا %9

برای پارامتر نهم است. برای کسب اطلاعات بیشتر به منابع معرفی شده در ابتدای مطلب مخصوصا قسمت Using batch parameters مراجعه کنید.
حال مجموعه دستورات زیر و خطای رخ داده را درنظر بگیرید:



با بررسی مطلب متوجه میشویم با اینکه خط اول مجموعه دستورات فوق درست بوده و کاملا صحیح اجرا میشود اما خطای رخ داده به کل دستورات اشاره دارد و مشخص نشده است که کدام دستور مشکل دارد. دقت کنید که دستور اول کاملا اجرا میشود!
راه حل ساده ای در اینجا برای حل این مشکل ارائه شده است. در این راه حل با استفاده از قابلیتهای این زبان، کل عملیات و مخصوصا خطاهای رخ داده در این مجموعه دستورات هندل میشود تا کنترل بهتری در این مورد بر روی فرایند وجود داشته باشد.
نمونه این راه حل به صورت زیر است:

```
echo ------------------------------------------------------------------------
echo Copy "$(ProjectDir)$(OutDir)*.*" c:\test --Starting...
Copy "$(ProjectDir)$(OutDir)*.*" c:\test
if errorlevel 1 goto error
echo Copy "$(ProjectDir)$(OutDir)*.*" c:\test --DONE!
echo ------------------------------------------------------------------------
echo ------------------------------------------------------------------------
echo Copy $(OutDir)*.* c:\test --Starting...
Copy $(OutDir)*.* c:\test
if errorlevel 1 goto error
echo Copy $(OutDir)*.* c:\test --DONE!
echo ------------------------------------------------------------------------
goto ok
:error
echo POSTBUILDSTEP for $(ProjectName) FAILED
notepad.exe
exit 1
:ok
echo POSTBUILDSTEP for $(ProjectName) COMPLETED OK
```

با استفاده از مجموعه دستوراتی شبیه دستورات بالا میتوان لحظه به لحظه اجرای عملیات را بررسی کرد.
نکته: خروجی تمام این دستورات و نیز خروجی دستورات echo در پنجره Output ویژوال استودیو به همراه سایر پیغامهای بیلد نمایش داده میشود.
نکته: در اسکریپت فوق برای درک بیشتر مسئله با استفاده از دستور notepad.exe در قسمت error: از وقوع خطا اطمینان حاصل میشود. دقت کنید تا زمانیکه برنامه اجرا شده Notepad بسته نشود فوکس به ویژوال استودیو برنمیگردد و عملیات بیلد تمام نمیشود.
نکته: درصورت استفاده از دستور exit 0 در انتهای قسمت error: (به جای دستور exit 1 موجود) به دلیل اعلام خروج موفق از عملیات، ویژوال استودیو خطایی نمایش نخواهد داد و عملیات بیلد بدون نمایش خطا و با موفقیت به پایان خواهد رسید. درواقع استفاده از هر عددی غیر از صفر به معنی خروج با خطا است که این عدد غیرصفر کد خطا یا error level را مشخص میکند (^ و ^).

یکی از دستورات جالبی که میتوان در این رویدادها از آن استفاده کرد، دستور نصب نسخه ریلیز برنامه در GAC است. نحوه

استفاده از آن میتواند به صورت زیر باشد:

```
if $(ConfigurationName) == Release (
gacutil.exe /i "$(SolutionDir)$(OutDir)$(TargetFileName)"
)
```

نکته: درصورتیکه در دستورات مربوط به رویداد قبل از بیلد یعنی Pre-build خطایی رخ بدهد عملیات بیلد متوقف خواهد شد و برای پروژه فایلی تولید نمیشود. اما اگر این خطا در رویداد بعد از بیلد یعنی Post-build رخ دهد با اینکه ویژوال استودیو وقوع یک خطا را گزارش میدهد اما فایلهای خروجی پروژه حاصله از عملیات بیلد تولید خواهند شد.

نکته: توجه داشته باشید که در استفاه از این ویژگی زیاده‌روی نباید کرد. استفاده زیاد و بیش از حد (و با تعداد زیاد دستورات) از این رویدادها ممکن است عملیات بیلد را دچار مشکلاتی پیچیده کند. دیباگ این رویدادها و دستورات موجود در آنها بسیار مشکل خواهد بود. اگر تعداد خطوط دستورات موردنظر زیاد باشد بهتر است کل دستورات را درون یک فایل bat. ذخیره کنید و این فایل را بطور جداگانه مدیریت کنید که کار راحتتری است.

نکته: بهتر است قبل از وارد کردن دستورات درون این رویدادها، ابتدا تمام دستورات را در یک پنجره cmd آزمایش کنید تا از درستی ساختار و نتیجه آن‌ها مطمئن شوید.

رویدادهای بیلد و MSBuild

همانطور که در اینجا توضیح داده شده است، ویژوال استودیو از ابزار MSBuild برای تولید اپلیکیشنها استفاده میکند. عملیات مدیریت رویدادهای بیلد نیز توسط این ابزار انجام میشود. اگر به فایل پروژه مربوط به مثال قبل مراجعه کنید به محتوایی شبیه خطوط زیر میرسید:

```
...
<PropertyGroup>
<PostBuildEvent>echo ------------------------------------------------------------------------------
echo Copy "$(ProjectDir)$(OutDir)*.*" c:\test --Starting...
Copy "$(ProjectDir)$(OutDir)*.*" c:\test
if errorlevel 1 goto error
echo Copy "$(ProjectDir)$(OutDir)*.*" c:\test --DONE!
echo ------------------------------------------------------------------------------
echo ------------------------------------------------------------------------------
echo Copy $(OutDir)*.* c:\test --Starting...
Copy $(OutDir)*.* c:\test
if errorlevel 1 goto error
echo Copy $(OutDir)*.* c:\test --DONE!
echo ------------------------------------------------------------------------------
goto ok
:error
echo POSTBUILDSTEP for $(ProjectName) FAILED
notepad.exe
exit 1
:ok
echo POSTBUILDSTEP for $(ProjectName) COMPLETED OK</PostBuildEvent>
</PropertyGroup>
...
```

همانطور که میبینید در ویژوال استودیو تنها ذخیره این تنظیمات در فایل پروژه انجام میشود و کلیه عملیات توسط ابزار MSBuild مدیریت میگردد. امکان بهر برداری از این رویدادها با استفاده مستقیم از ابزار MSBuild نیز وجود دارد اما به دلیل مفصل بودن بحث، جستجوی بیشتر به خوانندگان واگذار میشود.

منابع برای مطالعه بیشتر: (#How to: Specify Build Events (C

Specifying Custom Build Events in Visual Studio

Pre-build Event/Post-build Event Command Line Dialog Box

Customize Your Project Build Process

نظرات خوانندگان

نویسنده: سعید
تاریخ: ۱۳۹۱/۱۱/۱۵ ۱۸:۹

با تشکر از مطلب مفیدتان. برای کاربردهای معمولی تاجایی که دیدم بیشتر مثلا برای obfuscating خودکار اسمبلی پس از بیلد ازش استفاده میشه. اما در کارهای تیمی در continuous integration به نظر می‌رسه خیلی کاربرد داره. بررسی کیفیت کد، اجرای آزمون‌های واحد، اجرای آنالیزهای خودکار و مثل این‌ها

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۳/۰۷ ۱:۵۹

سلام با تشکر از پست شما
من می‌خوام اندازه پشته توی ویزوال استودیو تغییر بدم در Post Build Event کد زیر نوشتم

```
editbin.exe /STACK:1000000 $(TargetFileName)
```

اما در زمان کامپایل پروژه با این خطا مواجه می‌شم

```
Error 7 The command "editbin.exe /STACK:10000 MS-AUV.exe" exited with code 9009.MS-AUV
```

ممنون میشم راهنماییم کنین

نویسنده: یوسف نژاد
تاریخ: ۱۳۹۲/۰۳/۰۷ ۹:۴۹

با سلام در پاسخ به مشکل شما چند نکته باید اشاره بشه.

نکته اول: ماکروی TargetFileName فقط اسم فایل خروجی پروژه رو برمی‌گردونه، درصورتیکه برای کارکردن دستور فوق مسیر کامل فایل نیازه. چون برنامه editbin.exe درون مسیر خروجی پروژه شما اجرا نمی‌شه. شما می‌تونین از ماکروی TargetPath استفاده کنید که مسیر کامل فایل خروجی پروژه رو برمی‌گردونه.

نکته دوم: کد خطای 9009 مربوط به پیدا نکردن فایل هست. البته فایلی که در اینجا پیدا نشده خروجی پروژه شما نیست بلکه خود ابزار editbin هستش. مسیر درستش در سیستم 32 بیتی برای ویژوال استودیو 2010 اینه:

```
C:\Program Files\Microsoft Visual Studio 10.0\VC\bin\editbin.exe
```

اما چون این مسیرها معمولا حاوی **فاصله** هستند نیاز به استفاده از **دابل کوتیشن** در ابتدا و انتها وجود داره. بنابراین دستور کامل باید به صورت زیر باشه:

```
"C:\Program Files\Microsoft Visual Studio 10.0\VC\bin\editbin.exe" /STACK:1000000 "$(TargetPath)"
```

اما با اجرای دستور فوق باز هم خطایی صادر میشه که کمی خطرناکتر از قبلیه. و اما دلیلش:

نکته سوم: متن زیر از msdn گرفته شده:

```
You can start this tool only from the Visual Studio command prompt. You cannot start it from a system
```

```
command prompt or from Windows Explorer.
```

البته منظور دقیق‌تر این جمله اینه که ابزار editbin نیاز به یکسری تنظیمات و متغیرهای ازپیش تعیین شده داره که در Visual Studio command prompt انجام شده. اما نگران نباشید برای تنظیم این تنظیمات و تبدیل خط فرمان Build Events در ویژوال استودیو به یک Visual Studio command prompt کافیه که خط زیر رو در ابتدای مجموعه دستورات build events خودتون قرار بدین:

```
call "$(DevEnvDir)..\Tools\vsvars32.bat"
```

این بچ فایل حاوی دستوراتی نسبتا مفصل برای تنظیم تنظیمات موردنیاز است. درواقع با اجرای این بچ فایل هر خط فرمانی تقریبا تبدیل به Visual Studio command prompt خواهد شد. با توجه به ماکروی $(DevEnvDir) مسیر کامل این فایل در سیستم 32 بیتی و برای ویژوال استودیوی 2010 به صورت زیر است:

```
C:\Program Files\Microsoft Visual Studio 10.0\Common7\Tools\vsvars32.bat
```

بنابراین برای کار کردن دستور موردنظر شما کافیه که این دو دستور به صورت زیر در Post Build Event اضافه بشه:

```
call "$(DevEnvDir)..\Tools\vsvars32.bat"
"C:\Program Files\Microsoft Visual Studio 10.0\VC\bin\editbin.exe" /STACK:1000000 "$(TargetPath)"
```

نکته چهارم: با توجه به اشاره‌ای که در نکته قبلی شد ("با اجرای این فایل هر خط فرمانی تقریبا تبدیل به Visual Studio command prompt خواهد شد.") بنابراین دستور فوق را میتوان به صورت زیر خلاصه کرد:

```
call "$(DevEnvDir)..\Tools\vsvars32.bat"
"editbin.exe" /STACK:1000000 "$(TargetPath)"
```

موفق باشید.

نویسنده: یوسف نژاد
تاریخ: ۱۳۹۲/۰۳/۰۷ ۹:۵۳

نکته پنجم: پس از بررسی معلوم شد که اگر دستورات فوق ازطریق خط فرمان Build Events اجرا شوند استفاده از همان ماکروی $(TargetFileName) نیز کفایت می‌کند.

عنوان: **کامپایل خودکار یک پروژه برای دو فریم ورک**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۶/۲۵ ۱۸:۵
آدرس: www.dotnettips.info
گروه‌ها: MSBuild, .NET 4.5, Compile

فرض کنید می‌خواهید زمانیکه دکمه‌ی build در VS.NET فشرده شد، دو نسخه‌ی دات نت 4 و دات نت 4.5، از پروژه‌ی شما در پوشه‌های مجزایی کامپایل شده و قرار گیرند. در ادامه نحوه‌ی انجام این‌کار را بررسی خواهیم کرد.

پروژه نمونه

تنظیمات ذیل را بر روی یک پروژه از نوع class library دات نت 4 در VS 2013 اعمال خواهیم کرد.

ویرایش فایل پروژه برنامه

برای اینکه تنظیمات کامپایل خودکار مخصوص دات نت 4.5 را نیز به این پروژه دات نت 4 اضافه کنیم، نیاز است فایل csproj آن‌را مستقیما ویرایش نمائیم. این تغییرات شامل مراحل ذیل هستند:

الف) تعریف متغیر Framework

```
<PropertyGroup>
    <!-- ...-->
    <Framework Condition=" '$(Framework)' == '' ">NET40</Framework>
  </PropertyGroup>
```

به ابتدای فایل csproj در قسمت PropertyGroup آن یک متغیر جدید را به نام Framework اضافه کنید. از این متغیر در شرط‌های کامپایل استفاده خواهد شد.

ب) ویرایش مسیر خروجی تنظیمات کامپایل فعلی

```
<PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">
    <!-- ...-->
    <OutputPath>bin\$(Configuration)\$(Framework)\</OutputPath>
  </PropertyGroup>
```

در حال حاضر حداقل تنظیمات کامپایل حالت debug، در فایل پروژه موجود است. مقدار OutputPath آن‌را به نحو فوق تغییر دهید تا خروجی نهایی را در پوشه‌ای مانند bin\Debug\NET40 ایجاد کند.
بدیهی است اگر حالت release هم وجود دارد، نیاز است مقدار OutputPath آن‌را نیز به همین ترتیب ویرایش کرد.

ج) افزودن تنظیمات کامپایل دات نت 4.5 به پروژه جاری

```
<PropertyGroup Condition=" '$(Framework)' == 'NET45' And '$(Configuration)|$(Platform)' ==
'Debug|AnyCPU'">
    <TargetFrameworkVersion>v4.5</TargetFrameworkVersion>
    <PlatformTarget>AnyCPU</PlatformTarget>
    <DebugSymbols>true</DebugSymbols>
    <DebugType>full</DebugType>
    <Optimize>false</Optimize>
    <OutputPath>bin\$(Configuration)\$(Framework)\</OutputPath>
    <DefineConstants>DEBUG;TRACE;NET45</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>

  <PropertyGroup Condition=" '$(Framework)' == 'NET45' And '$(Configuration)|$(Platform)' ==
'Release|AnyCPU' ">
    <TargetFrameworkVersion>v4.5</TargetFrameworkVersion>
    <PlatformTarget>AnyCPU</PlatformTarget>
    <DebugType>pdbonly</DebugType>
    <Optimize>true</Optimize>
    <OutputPath>bin\$(Configuration)\$(Framework)\</OutputPath>
    <DefineConstants>TRACE;NET45</DefineConstants>
```

```
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>
```

در اینجا تنظیمات حالت debug و release مخصوص دات نت 4.5 را مشاهده می‌کنید. برای نگارش‌های دیگر، تنها کافی است مقدار TargetFrameworkVersion را ویرایش کنید.
همچنین اگر به DefineConstants آن دقت کنید، مقدار NET45 نیز به آن اضافه شده‌است. این مورد سبب می‌شود که بتوانید در پروژه‌ی جاری، شرطی‌هایی را ایجاد کنید که کدهای آن فقط در حین کامپایل برای دات نت 4.5 به خروجی اسمبلی نهایی اضافه شوند:

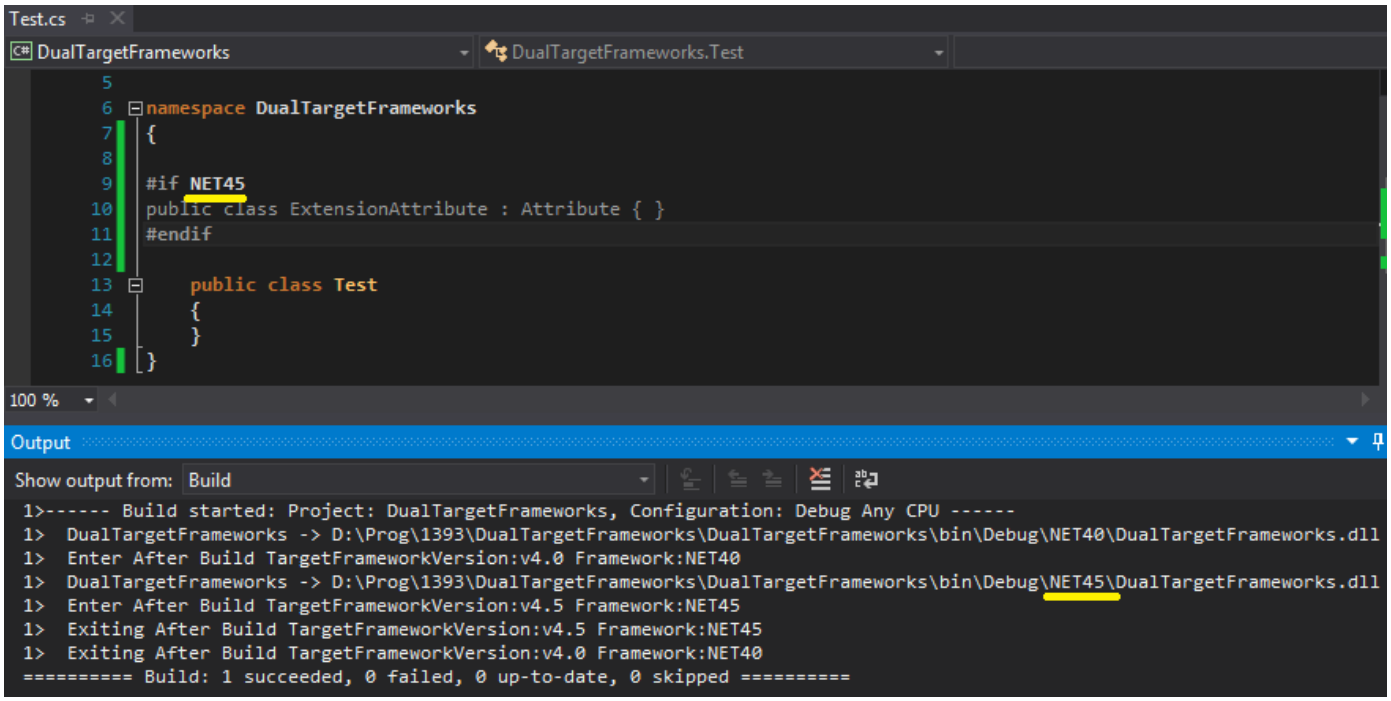
```
#if NET45
public class ExtensionAttribute : Attribute { }
#endif
```

د) افزودن تنظیمات پس از build

در انتهای فایل csproj قسمت AfterBuild به صورت کامنت شده موجود است. آن‌را به نحو ذیل تغییر دهید:

```
<Target Name="AfterBuild">
    <Message Text="Enter After Build TargetFrameworkVersion:$(TargetFrameworkVersion)
Framework:$(Framework)" Importance="high" />
    <MSBuild Condition=" '$(Framework)' != 'NET45'" Projects="$(MSBuildProjectFile)"
Properties="Framework=NET45" RunEachTargetSeparately="true" />
    <Message Text="Exiting After Build TargetFrameworkVersion:$(TargetFrameworkVersion)
Framework:$(Framework)" Importance="high" />
  </Target>
```

این تنظیم سبب می‌شود تا کامپایل مخصوص دات نت 4.5 نیز به صورت خودکار فعال گردد و خروجی آن در مسیر bin\Debug\NET45 به صورت جداگانه‌ای قرار گیرد.



برای آزمایش بیشتر، فایل csproj نهایی را از اینجا می‌توانید دریافت کنید:

DualTargetFrameworks.zip

نظرات خوانندگان

نویسنده: مهدی نقدی
تاریخ: ۱۳۹۳/۰۶/۳۰ ۲۰:۵۶

خیلی موضوع خوبی بود. واقعا خسته نباشید.
اگر از کتابخانه ای استفاده کنیم که DLL دات نت ۴ و ۴.۵ جداگانه ای رو ارائه داده باشه، چطور میشه این موضوع را پوشش داد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۶/۳۰ ۲۲:۱

ویژگی‌های Condition ذکر شده در متن، در مورد ارجاعات هم قابل تنظیم است:

```
<Reference Include="MyAssembly"  Condition=".........">
  <SpecificVersion>False</SpecificVersion>
  <HintPath>path\to\MyAssembly.dll</HintPath>
</Reference>
```

همچنین در مورد فایل‌های سورس:

```
<Compile Include="Class20.cs" Condition=" ....." />
```