Jeffrey van Gogh is Software Development Engineer bij Microsoft Corporation

MSBuild is de nieuwe motor onder de kap van Visual Studio

DE BUILD ENGINE VOOR GROTE PROJECTEN

Visual Studio 2005 biedt een compleet nieuwe build engine. Deze engine heet MSBuild en is gebouwd, omdat klanten een build engine wilden hebben die niet alleen makkelijk te gebruiken is vanuit Visual Studio, maar ook krachtig genoeg is om grote projecten te bouwen. Zo krachtig dat Visual Studio zelf nu met MSBuild gebouwd wordt.

Aangezien elke nieuwe computertaal traditiegetrouw wordt geïntroduceerd met een Hello World-voorbeeld, kan dat in dit MSBuild-artikel ook niet ontbreken. MSBuild gebruikt XML voor de structuur van zijn taal. De grammaticale hoofdpunten zijn Project, Targets, Tasks, PropertyGroups en ItemGroups.

- Project is de root XML-node en geeft aan dat dit een MSBuild-project is, en welke targets moeten worden uitgevoerd bij een standaardcompilatie.
- Targets zijn de eenheid van compilatie. Een target heeft input, output en dependencies. (bijvoorbeeld compileer .cs file naar .dll).
- Tasks zijn commando's die worden uitgevoerd om de target te voltooien. (bijvoorbeeld voer CSC.exe uit).
- PropertyGroups zijn definities van variabelen die kunnen worden gebruikt in MSBuild (bijvoorbeeld welke debug-flags moeten meegegeven aan CSC.exe).
- ItemGroups zijn definities van variabele lijsten (bijvoorbeeld de lijst van .cs-files die moeten worden gecompileerd).

```
<Project  DefaultTargets="HelloTarget"
          xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
   <Target Name="HelloTarget">
      <Message Text="Hello World!" />
   </Target>
</Project>
```

Codevoorbeeld 1.

```
<Project  DefaultTargets="Build"
          xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
   <PropertyGroup>
      <TargetType>exe</TargetType>
      <OutputAssembly>HelloWorld.exe</OutputAssembly>
   </PropertyGroup>
   <ItemGroup>
      <Sources Include="HelloWorld.cs" />
   </ItemGroup>
   <Target Name="Build" Inputs="@(Sources)"
          Outputs="$(OutputAssembly)">
      <CSC Sources="@(Sources)"
          OutputAssembly="$(OutputAssembly)"
          TargetType="$(TargetType)" />
   </Target>
</Project>
```

Codevoorbeeld 2.

Met deze informatie zijn we klaar voor HelloWorld.proj; zie codevoorbeeld 1. Dit project is uit te voeren door een Visual Studio command-prompt te openen (onder Start, Programs, Visual Studio, Visual Studio Tools). Sla het bestand op en voer uit: *MSBuild HelloWorld.proj*. Dit geeft de output zoals te zien is in afbeelding 1.

De traditionele HelloWorld werkt. Aangezien we geen volledige programma's in MSBuild, maar in .NET willen schrijven, bekijken we een voorbeeld waar HelloWorld in .NET is geschreven en door MSBuild wordt gebouwd. Maak eerst je favoriete HelloWorld.cs (of .VB, in dat geval verander je in het voorbeeld CSC door VBC). Maak vervolgens een msbuild-project met de volgende code zoals te zien is in codevoorbeeld 2. Wat opvalt is dat we nu PropertyGroups en ItemGroups gebruiken. Variabelen gedefinieerd in PropertyGroups kunnen met $(variablename) worden gebruikt in zowel targets als tasks. Is een variabele in een PropertyGroup meer keren gedefinieerd, dan wordt de laatst toegewezen waarde gebruikt. Variabelen in ItemGroups kunnen met @(variablename) worden gebruikt. Voor variabelen in ItemGroups geldt dat elke nieuwe definitie wordt toegevoegd aan de lijst met waarden. Aan de target hebben we inputs en outputs toegevoegd. MSBuild kijkt bij aanroep van de target of de input en output op dat moment bestaan. Als ze beide bestaan, vergelijkt MSBuild de timestamps van de bestanden. Als de output ouder is dan de input, dan wordt de target uitgevoerd, anders wordt deze overgeslagen. Als je MSBuild tweemaal achter elkaar los-

```
Microsoft (R) Build Engine Version 2.0.50727.42
[Microsoft .NET Framework, Version 2.0.50727.42]
Copyright (C) Microsoft Corporation 2005. All rights reserved.

Build started 6/30/2006 7:04:39 AM.

_____________________________________________________
Project "C:\MSBuildArtikel\HelloWorld.proj" (default targets):

Target HelloTarget:
    Hello World!

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:00.45
```

Afbeelding 1. Output van MSBuild HelloWorld.proj

laat op dit nieuwe project, zul je zien dat HelloWorld slechts eenmaal wordt gecompileerd.

Visual Studio-integratie

Laten we met behulp van de informatie hierboven naar een Visual Studio-project kijken. Om het simpel te houden heb ik een HelloWorld Console-project aangemaakt. Bij het openen van dit project in een texteditor zie je een bestand dat veel lijkt op wat we eerder zagen; zie codevoorbeeld 3.

We kunnen hier een nieuw concept zien: condities. Op bijna elke XML-tag in een MSBuild-project kan een conditie worden geplaatst. Dit wordt in dit geval gebruikt om de instellingen voor verschillende platformen (x86/x64 enzovoort) en configuraties (debug/release) te specificeren. Het valt op dat dit project geen targets bevat (op het commentaar na). De

```
<Project DefaultTargets="Build"
         xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <Configuration Condition=" '$(Configuration)' == '' ">Debug
    </Configuration>
    <Platform Condition=" '$(Platform)' == '' ">AnyCPU</Platform>
    <ProductVersion>8.0.50727</ProductVersion>
    <SchemaVersion>2.0</SchemaVersion>
    <ProjectGuid>{7FB52BE0-55E1-49FD-8F1D-187C30379F57}</ProjectGuid>
    <OutputType>Exe</OutputType>
    <AppDesignerFolder>Properties</AppDesignerFolder>
    <RootNamespace>HelloWorldVS</RootNamespace>
    <AssemblyName>HelloWorldVS</AssemblyName>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' =='Debug|AnyCPU' ">
    <DebugSymbols>true</DebugSymbols>
    <DebugType>full</DebugType>
    <Optimize>false</Optimize>
    <OutputPath>bin\Debug\</OutputPath>
    <DefineConstants>DEBUG;TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' =='Release|AnyCPU' ">
    <DebugType>pdbonly</DebugType>
    <Optimize>true</Optimize>
    <OutputPath>bin\Release\</OutputPath>
    <DefineConstants>TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>
  <ItemGroup>
    <Reference Include="System" />
    <Reference Include="System.Data" />
    <Reference Include="System.Xml" />
  </ItemGroup>
  <ItemGroup>
    <Compile Include="Program.cs" />
    <Compile Include="Properties\AssemblyInfo.cs" />
  </ItemGroup>
  <Import Project="$(MSBuildBinPath)\Microsoft.CSharp.targets" />
  <!-- To modify your build process, add your task inside one of the
    targets below and uncomment it.  Other similar extension points
    exist, see Microsoft.Common.targets.
  <Target Name="BeforeBuild">
  </Target>
  <Target Name="AfterBuild">
  </Target>
  -->
</Project>
```

Codevoorbeeld 3.

DefaultTargets aan het begin van het project geven aan dat dit 'Build' is, maar de build is niet te vinden. Dit komt doordat de buildtarget voorgedefinieerd is in een bestand genaamd 'Microsoft.Common.Targets'. Dat bestand wordt geïmporteerd door 'Microsoft.CSharp.Targets' dat vervolgens in ons project wordt geïmporteerd met behulp van de <Import> tag. In een

TIP

Om de bron van het project in Visual Studio zelf te bewerken (en Intellisense te krijgen), volg je de volgende stappen:

- Open het project normaal
- Ga naar de 'Solution Explorer'
- Klik met de rechtermuisknop op het project
- Selecteer 'Unload Project' (het project geeft nu aan 'unavailable')
- Klik nogmaals met de rechtermuisknop op het project
- Selecteer 'Edit Project'

Om het project weer te openen:

- Klik nogmaals met de rechtermuisknop op het project
- Selecteer 'Reload Project'
- Antwoord ja op de vraag of de broncode voor het project gesloten mag.

TIP

Als je targets wilt toevoegen aan meer projecten, dan kun je dit doen door een eigen targets-file te maken en deze te importeren in de projecten waar je de targets wilt gebruiken. Visual Studio geeft je dan wel een waarschuwing dat dit een niet standaardproject is (omdat er willekeurige code kan worden uitgevoerd). Als je de targets-file vertrouwt, kun je deze waarschuwing uitschakelen door de targets-file toe te voegen aan de registry onder HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\VisualStudio\8.0\MSBuild\SafeImports

Je kunt dit vermijden door je targets-file op te slaan als "Program Files\MSBuild\v2.0\Custom.After.Microsoft.Common.targets". Dit bestand wordt automatisch geïmporteerd door Microsoft.Common.targets zonder bericht. Nadeel is dat dit niet werkt in een groepsituatie waar je de projecten en targets wilt inchecken in een source-controlsysteem.

```
<PropertyGroup>
  <BuildDependsOn>
    BeforeBuild;
    CoreBuild;
    AfterBuild
  </BuildDependsOn>
</PropertyGroup>
<Target Name="Build"
        DependsOnTargets="$(BuildDependsOn)"
        Outputs="$(TargetPath)"/>
<Target Name="BeforeBuild"/>
<Target Name="AfterBuild"/>
<Target Name="CoreBuild"
        Inputs="@(Compile);@(Reference);
        Outputs="@(DocFileItem);
        @(IntermediateAssembly); >
  <Csc DebugType="$(DebugType)"
       DefineConstants="$(DefineConstants)"
       Optimize="$(Optimize)"
       OutputAssembly="@(IntermediateAssembly)"
       Platform="$(Platform)"
       References="@(References)"
       Sources="@(Compile)"
       TargetType="$(OutputType)"
       WarningLevel="$(WarningLevel)"
       />
</Target>
```

Codevoorbeeld 4.

versimpelde weergave, is de buildtarget als volgt gedefinieerd; zie codevoorbeeld 4. De targets in Microsoft.CSharp.targets en Microsoft.Common.targets weten dus al welke stappen ze moeten uitvoeren om een C#-project te bouwen. In ons project hoeven we alleen de waardes te definiëren voor de properties en items die gebruikt worden door de targets. In codevoorbeeld 4 zagen we dat er twee lege targets werden gedefinieerd: BeforeBuild en AfterBuild. Deze zijn aanwezig, zodat we Microsoft.Sharp.targets kunnen gebruiken en eigen stappen kunnen toevoegen aan de build. Om dit te doen, verwijder je de commentaar begin- en eindblocks uit ons project en voeg je eigen taken toe aan de BeginBuild- en AfterBuild-targets. Een ding is hier belangrijk: zorg dat de import van Microsoft.CSharp.targets hoger in het project is dan de herdefinitie van Before/AfterBuild. MSBuild maakt namelijk een ronde door alle geïmporteerde projecten en gebruikt de definitie die het laatst wordt gevonden. Verder is het handig te weten dat bijna elk msbuild-project, hoeveel het ook gespecialiseerd is, in Visual

```
using System;
using System.IO;
using System.Text;
using System.CodeDom;
using System.Reflection;
using System.CodeDom.Compiler;
using System.Collections.Generic;
using Microsoft.CSharp;
using Microsoft.VisualBasic;
using Microsoft.Build.Framework;
using Microsoft.Build.Utilities;

namespace BuildVersionTask
{
  public class BuildVersion : Task
  {
    private byte m_major;
    private byte m_minor;
    private string m_language;
    private string m_temporarySourceFile;
    private List<ITaskItem> m_generatedSource;
    private DateTime m_projectStartDate;

    public byte Major
    {
      get { return m_major; }
      set { m_major = value; }
    }

    public byte Minor
    {
      get { return m_minor; }
      set { m_minor = value; }
    }

    public string Language
    {
      get { return m_language; }
      set { m_language = value; }
    }

    public DateTime ProjectStartDate
    {
      get { return m_projectStartDate; }
      set { m_projectStartDate = value; }
    }

    [Output]
    public ITaskItem[] GeneratedSource
    {
      get { return m_generatedSource.ToArray(); }
      set
      {
        m_generatedSource.Clear();
        m_generatedSource.AddRange(value);
      }
    }

    public string TemporarySourceFile
    {
      get { return m_temporarySourceFile; }
      set { m_temporarySourceFile = value; }
    }
```

```
    public override bool Execute()
    {
      FileInfo info = new FileInfo(m_temporarySourceFile);
      //Only write new file if it wasn't written before today.
      if (info.LastWriteTime < DateTime.Today)
      {
        //Use System.CodeDom to write a source file with assembly
        //attributes.
        CodeAttributeDeclaration assemblyVersionAttribute =
          new CodeAttributeDeclaration( new CodeTypeReference(
            typeof(AssemblyVersionAttribute)));
        //we version the assembly version without the date and
        //assembly file version with the date as described in
        //CLR via C# by Jeff Richter
        Version version = new Version(m_major, m_minor, 0, 0);
        CodeAttributeArgument versionArgument = new CodeAttributeArgument(
          new CodePrimitiveExpression(version.ToString()));
        assemblyVersionAttribute.Arguments.Add(versionArgument);

        TimeSpan timeSinceProjectStart = DateTime.Now - m_projectStartDate;

        CodeAttributeDeclaration assemblyFileVersionAttribute =
          new CodeAttributeDeclaration( new CodeTypeReference(
              typeof(AssemblyFileVersionAttribute)));
        Version fileVersion = new Version(m_major, m_minor,
          timeSinceProjectStart.Days, 0);
        CodeAttributeArgument fileVersionArgument =
          new CodeAttribute Argument(
            new CodePrimitiveExpression( fileVersion.ToString()));
        assemblyFileVersionAttribute.Arguments.Add(fileVersionArgument);

        CodeCompileUnit unit = new CodeCompileUnit();
        unit.AssemblyCustomAttributes.Add(assemblyVersionAttribute);
        unit.AssemblyCustomAttributes.Add(assemblyFileVersionAttribute);

        CodeDomProvider provider;
        switch (m_language)
        {
          case "C#":
            provider = new CSharpCodeProvider();
            break;
          case "VB":
            provider = new VBCodeProvider();
            break;
          default:
            throw new NotSupportedException("Unsupported language: " +
              m_language);
        }
        CodeGeneratorOptions options = new CodeGeneratorOptions();
        using (StreamWriter writer = new StreamWriter(m_temporarySourceFile))
        {
          provider.GenerateCodeFromCompileUnit( unit, writer, options);
        }
      }
      //Provide source file to target as output
      //so it can be added to the list of files to compile
      m_generatedSource = new List<ITaskItem>();
      m_generatedSource.Add(new TaskItem(m_temporarySourceFile));
      //Everything succeeded like planned, so return true to notify MSBuild
      return true;
    }
  }
}
```

Codevoorbeeld 5.

```
<Target Name="BeforeBuild">
  <BuildVersion Major="1" Minor="3"
    ProjectStartDate="1/1/2006"
    Language="$(Language)"   TemporarySourceFile="$(IntermediateOutput
          Path)\Version$(DefaultLanguageSourceExtension)">
    <Output ItemName="Compile" TaskParameter="GeneratedSource"/>
  </BuildVersion>
</Target>
```

Codevoorbeeld 6.

Studio kan worden geladen. De enige restricties zijn:

- Noem het bestand .csproj of .vbproj om de juiste Intellisense te krijgen
- Zorg dat de volgende twee properties aanwezig zijn voor conversie bij nieuwe releases van Visual Studio:

```
<ProductVersion>8.0.50727</ProductVersion>
<SchemaVersion>2.0</SchemaVersion>
```

- Zorg dat de volgende property aanwezig is voor correcte dependency-tracking:

```
<ProjectGuid>{<unique-guid>}</ProjectGuid>
```

Een unieke GUID kun je genereren door een Visual Studio command-prompt te openen en 'uuidgen –c' uit te voeren.

Tasks

Tot nu toe hebben we twee tasks gezien: Message en Csc. Er zijn natuurlijk veel meer taken die worden uitgevoerd tijdens een build-proces. Bovendien moeten we ook nieuwe taken kunnen toevoegen. In MSBuild is elke task een .Net-class. Deze classes implementeren Microsoft.Build.Framework.ITask of zijn afgeleid van Microsoft.Build.Utilities.Task. Properties op deze classes zijn beschikbaar in de msbuild-projectfiles als de attributen op het task-element. Om te zien welke tasks standaard beschikbaar zijn, open je %WINDIR%\Microsoft.net\Framework\v2.0.50727\Microsoft.Build.Tasks.dll in ILDasm.
Of kijk op http://msdn2.microsoft.com/en-us/library/7z253716.aspx. Een van de tasks die je tot je beschikking moet hebben, is een task die de versienummers van een assembly automatisch aanpast. Deze zit niet standaard in MSBuild, dus laten we die zelf implementeren; zie codevoorbeeld 5.

Een aantal dingen valt hier bij op:

- Een van de properties is gemarkeerd met het [Output]-attribuut. Dit zorgt ervoor dat MSBuild weet dat deze property kan worden gebruikt als output van de task.
- Bijna alle properties zijn primitieve waarden (string, bool, DateTime enzovoort), behalve de lijst van GeneratedSource. ITaskItem is de interface die een waarde uit een ItemGroup representeert. ITaskItem[] is een volledige ItemGroup.
- Exceptions worden gebruikt om fatale problemen aan de gebruiker te melden.

Nu we de task hebben geschreven, kunnen we deze gebruiken in ons project. Om dit te doen, moeten we eerst de huidige (niet datumafhankelijke) versie uit ons project halen. Deze is te vinden in AssemblyInfo.

Tip: Als je zelf tasks gaat ontwikkelen, dan wil je deze natuurlijk ook debuggen. Een handige truc is om msbuild met de volgende switch aan te roepen om allerlei nuttige informatie te zien over de waardes van properties enitemgroups en welke targets/tasks worden aangeroepen: /verbosity:diagnostic.
Deze zelfde instelling is te verkrijgen in Visual Studio door naar Tools, Options, Projects & Solutions, Build and Run te gaan en voor "MSBuild project build output verbosity" op diagnostic te zetten.

cs/vb. Daarna kunnen we onze task aan ons project toevoegen. Eerst moeten we MSBuild vertellen waar het de task vandaan moet halen:

```
<UsingTask
  AssemblyFile="C:\BuildVersionTask\BuildVersionTask.dll"
  TaskName="BuildVersionTask.BuildVersion"/>
```

In dit geval wijzen we MSBuild naar de exacte dll. Als we de dll in de Global Assembly Cache plaatsen en daar vandaan willen halen, dan moeten we het AssemblyName-attribuut gebruiken in plaats van AssemblyFile:

```
<UsingTask
  AssemblyName="BuildVersionTask, Version=1.3.0.0"
  TaskName="BuildVersionTask.BuildVersion"/>
```

Nu MSBuild de task kan vinden, is het tijd om onze task uit te voeren voordat de build plaatsvindt; zie codevoorbeeld 6.

We gebruiken hierbij drie variabelen die al zijn gedefinieerd door MSBuild:

- $(IntermediateOutputPath): de folder waar MSBuild tijdelijke build-bestanden plaatst (normaal obj\debug of obj\release)
- $(Language): de taal van het project (bijvoorbeeld C# of VB)
- $(DefaultLanguageSourceExtension): de standaard extensie van de source-bestanden inclusief de punt (bijvoorbeeld .cs of .vb)

.
Verder zien we dat de task een subelement <Output> heeft. Dit geeft MSBuild de opdracht om de waarde van de property GeneratedSource (van het type ITaskItem[]) te plaatsen in de ItemGroup 'Compile'. En zoals we al eerder zagen, was dit nu net de ItemGroup die MSBuild gebruikt om te compiler te vertellen welke source-bestanden te compileren.
Na het laden van het project en het uitvoeren van een compile, moeten we nu ons gegenereerde versienummer zien in de gecompileerde assembly (bijvoorbeeld met ILDasm of onder eigenschappen in Windows Verkenner).

Nog meer aanpassingsmogelijkheden

MSBuild biedt nog meer aanpassingsmogelijkheden dan we in dit artikel hebben laten zien. Er zijn allerlei utility-classes om meer uitgebreide tasks te schrijven. Er is een objectmodel om zelf projecten te creëren en te wijzigen. Dit alles maakt MSBuild tot een nieuwe motor onder de (build) kap van Visual Studio die een hele hoop extra pk's levert.

Jeffrey Van Gogh is Software Development Engineer bij Microsoft in Redmond. Hij werkt in het Visual Studio Code Analysis Team als een van de ontwikkelaars van FxCop. Hij is momenteel werkzaam aan een project om Code Analysis rapportage over Visual Studio te integreren in de build van Visual Studio met behulp van MSBuild. Voor vragen over dit artikel kun je hem een mail sturen: Jeffrey.van.Gogh@microsoft.com

Referenties

MSDN's officiële MSBuild-documentatie: http://msdn2.microsoft.com/en-us/library/wea2sca5.aspx
De blog van het MSBuild-team met tips & tricks en nieuwtjes over de volgende versie van MSBuild: http://blogs.msdn.com/msbuild
Het officiële forum van het MSBuild-team, hier kun je met al je vragen terecht over MSBuild, configuratie, tasks, objectmodel et cetera: http://forums.microsoft.com/MSDN/ShowForum.aspx?ForumID=27&SiteID=1
MSBuild-wiki, hier kun je allerlei (onofficiële) documentatie vinden over MSBuild en zelf dingen toevoegen en wijzigen: http://channel9.msdn.com/wiki/default.aspx/MSBuild.HomePage