Учимся готовить MSBuild

Mikhail Filippov, JetBrains

План

- Историческая справка
- Основы написания билд-скриптов MSBuild
- Примеры решения проблем с билд скриптами
- Особенности применения MSBuild в .NET Core
- Вопросы

Историческая справка

- MSBuild 2.0, 3.5, 4.0 - часть .NET Framework
- MSBuild 12.0,14.0,15.0 - входит в состав Visual Studio + доступен без Visual Studio как Microsoft Build Tools
- .NET Core и попытки изобрести новую build систему на базе project.json

Основные строительные блоки MSBuild

- Property - скалярные величины, обычно применяются для хранения параметров билда
- Item - массивы значений с метадатой, применяются для группировки наборов элементом например файлов
- Import - включение одних проектов в другие
- Task - Минимальные единицы работы в MSBuild
- Project - корневой элемент любого MSBuild файла (csproj, vbproj, xproj 😄)

Простой проект

```
<!-- Определение Project -->
<Project>
  <PropertyGroup>
    <!-- Определение Property -->
    <Configuration>Debug</Configuration>
  </PropertyGroup>
  <ItemGroup>
  <!-- Определение Item -->
    <SourceFiles Include="src.cs"/>
  </ItemGroup>
  <!-- Определение Target -->
  <Target Name="Build">
    <!-- Вызов Task -->
    <Message Text="Configuration=$(Configuration)"/>
    <Message Text="SourceFiles=@(SourceFiles)"/>
    </Target>
</Project>
```

```
$ msbuild /nologo test.proj
Build:
  Configuration=Debug
  SourceFiles=src.cs
Build succeeded.
```

Properties

Статические и динамические

```
<Project>
  <PropertyGroup>
    <!-- Статическое Property -->
    <Configuration>Debug</Configuration>
  </PropertyGroup>
  <Target Name="Build">
    <!-- Динамические Property -->
    <CreateProperty Value="bin/$(Configuration)">
      <Output TaskParameter="Value" PropertyName="OldProp" />
    </CreateProperty>
    <PropertyGroup>
      <NewProp>bin/$(Configuration)</NewProp>
    </PropertyGroup>
    <Message Text="OldProp=$(OldProp)"/>
    <Message Text="NewProp=$(NewProp)"/>
  </Target>
</Project>
```

```
$ msbuild /nologo test.proj
Build:
  OldProp=bin/Debug
  NewProp=bin/Debug
Build succeeded.
```

Properties

Могут задаваться из командной строки и переменных окружения

```
<Project>
    <PropertyGroup>
        <Platform Condition="'$(Platform)' == ''">x64</Platform>
        <Configuration>Debug</Configuration>
    </PropertyGroup>
    <Target Name="Build">
        <Message Text="Configuration=$(Configuration)"/>
        <Message Text="Platform=$(Platform)" />
    </Target>
</Project>
```

```
$ Platform=x86 msbuild /nologo test.proj /p:Configuration=Release
Build:
  Platform=x86
  Configuration=Release
Build succeeded.
```

Properties

Существует набор предопределенных свойств

```
<Project>
  <Target Name="Build">
    <Message
     Text="MSBuildThisFileName=$(MSBuildThisFileName)"/>
    <Message
     Text="MSBuildProjectDirectory=$(MSBuildProjectDirectory)"/>
    <Message
      Text="MSBuildProjectFile=$(MSBuildProjectFile)"/>
    </Target>
</Project>
```

```
$ msbuild /nologo test.proj
Build:
  MSBuildThisFileName=test
  MSBuildProjectDirectory=/private/tmp
  MSBuildProjectFile=test.proj

Build succeeded.
```

* - https://docs.microsoft.com/en-us/visualstudio/msbuild/msbuild-reserved-and-well-known-properties

Properties

Для свойств важен порядок их определения

```
<Project>
  <PropertyGroup>
    <InvalidOutputPath>bin/$(Configuration)</InvalidOutputPath>
    <Configuration>Debug</Configuration>
    <DebugOutputPath>bin/$(Configuration)</DebugOutputPath>
    <Configuration>Release</Configuration>
    <ReleaseOutputPath>bin/$(Configuration)</ReleaseOutputPath>
  </PropertyGroup>
  <Target Name=“Build">
    <Message Text="InvalidOutputPath=$(InvalidOutputPath)" />
    <Message Text="Configuration=$(Configuration)"/>
    <Message Text="DebugOutputPath=$(DebugOutputPath)"/>
    <Message Text="ReleaseOutputPath=$(ReleaseOutputPath)"/>
  </Target>
</Project>
```

```
$ msbuild /nologo test.proj
Build:
  InvalidOutputPath=bin/
  Configuration=Release
  DebugOutputPath=bin/Debug
  ReleaseOutputPath=bin/Release
Build succeeded.
```

Items

Статические и динамические

```
<Project>
  <ItemGroup>
    <!-- Статический Item -->
    <SourceFiles Include="1.txt"/>
  </ItemGroup>
  <Target Name="Build">
    <!-- Динамические Item -->
    <CreateItem Include="2.txt">
      <Output TaskParameter="Include" ItemName="SourceFiles" />
    </CreateItem>
    <ItemGroup>
      <SourceFiles Include="3.txt"/>
    </ItemGroup>
    <Message Text="SourceFiles=@(SourceFiles)"/>
  </Target>
</Project>
```

```
$ msbuild /nologo test.proj
Build:
  SourceFiles=1.txt;2.txt;3.txt
Build succeeded.
```

Items

Могут содержать metadata и item transformation

```
<Project>
  <ItemGroup>
    <SourceFiles Include="1.txt">
      <ShouldDeployOnServer>Yes</ShouldDeployOnServer>
    </SourceFiles>
    <SourceFiles Include="2.txt">
      <ShouldDeployOnServer>No</ShouldDeployOnServer>
    </SourceFiles>
  </ItemGroup>
  <Target Name="Build">
    <Message
      Text="Metadata=@(SourceFiles->'%(ShouldDeployOnServer)')"/>
  </Target>
</Project>
```

```
$ msbuild /nologo test.proj
Build:
  Metadata=Yes;No
Build succeeded.
```

Items

Well-known metadata

```
<Project>
  <PropertyGroup>
    <Dest>dest</Dest>
  </PropertyGroup>
  <ItemGroup>
    <SourceFiles Include="src\**\*.txt" />
  </ItemGroup>
  <Target Name="Build">
    <Copy SourceFiles="@(SourceFiles)"
      DestinationFiles=
"@(SourceFiles->'$(Dest)\%(RecursiveDir)%(Filename)%(Extension)')"/>
  </Target>
</Project>
```

```
$  msbuild /nologo test.proj
Build:
  Creating directory "dest/1".
  Copying file from "src/1/1.txt" to "dest/1/1.txt".
  Copying file from "src/1/2.txt" to "dest/1/2.txt".
  Creating directory "dest/2".
  Copying file from "src/2/1.txt" to "dest/2/1.txt".
  Copying file from "src/2/2.txt" to "dest/2/2.txt".
Build succeeded.
```

* https://docs.microsoft.com/en-us/visualstudio/msbuild/msbuild-well-known-item-metadata

Property Functions

Доступны для вызова:

- Методы и свойства класса String.
- Статические методы стандартной библиотеки
- Специальные функции MSBuild

* https://docs.microsoft.com/en-us/visualstudio/msbuild/property-functions

Property Functions

Примеры

```
<Project>
  <PropertyGroup>
    <OutputPath>bin\Debug\</OutputPath>
    <Configuration>
      $(OutputPath.TrimEnd(‘/').Replace('bin/',''))
    </Configuration>
    <Guid>$([System.Guid]::NewGuid())</Guid>
    <RegEx>
     ‘$([System.Text.RegularExpressions.Regex]::IsMatch($(OutputPath),
      ‘\w/\w'))'
    </RegEx>
  </PropertyGroup>
  <Target Name="Build">
    <Message Text="Guid=$(Guid)"/>
    <Message Text="Configuration=$(Configuration)"/>
    <Message Text="RegEx=$(RegEx)"/>
  </Target>
</Project>
```

```
$ msbuild /nologo test.proj
Build:
  Guid=fed78c0a-12b7-48f8-899a-d0131422cf99
  Configuration=Debug
  RegEx='True'
Build succeeded.
```

Tasks

Создаем простую Task для MSBuild

```
public class HelloDotNext : ITask {
  public IBuildEngine BuildEngine { get; set; }
  public ITaskHost HostObject { get; set; }
  public bool Execute() {
    var loggingHelper = new TaskLoggingHelper(this);
    loggingHelper
      .LogMessageFromText("Hello .NEXT", MessageImportance.High);
    return true;
  }
}
```

```
<Project>
  <UsingTask TaskName="HelloDotNext"
    AssemblyFile="$(MSBuildProjectDirectory)\HelloDotNext.dll"/>
  <Target Name="Build">
    <HelloDotNext/>
  </Target>
</Project>
```

```
$  msbuild /nologo test.proj
Build:
  Hello .NEXT
Build succeeded.
```

Tasks

Передаем параметры в Task и получаем Output

```
public class HelloDotNext : Task {
  [Required] public string Name { get; set; }
  [Output] public string Greeting { get; set; }
  public override bool Execute() {
    Log.LogMessageFromText(
      "Name=" + Name, MessageImportance.High);
    Greeting = "Hello " + Name;
    return true;
  }
}
```

```
<Project>
  <UsingTask AssemblyFile=“...” TaskName="HelloDotNext" />
  <Target Name="Build">
    <HelloDotNext Name=".NEXT">
      <Output TaskParameter="Greeting" PropertyName="Greeting"/>
    </HelloDotNext>
    <Message Text="$(Greeting)"/>
  </Target>
</Project>
```

```
$  msbuild /nologo test.proj
Build:
  Name=.NEXT
  Hello .NEXT
Build succeeded.
```

Tasks

Inline task

```
<Project>
 <UsingTask TaskName="HelloDotNext" TaskFactory="CodeTaskFactory"
  AssemblyFile="$(MSBuildToolsPath)\Microsoft.Build.Tasks.Core.dll">
   <ParameterGroup>
     <Name Required="true"/>
     <Greeting Output="true"/>
   </ParameterGroup>
   <Task>
     <Code Type="Fragment" Language="cs">
       <![CDATA[
       Log.LogMessageFromText("Name=" + Name, MessageImportance.High);
       Greeting = "Hello " + Name;
       ]]>
     </Code>
   </Task>
  </UsingTask>
  <Target Name="Build">...</Target>
</Project>
```

```
$  msbuild /nologo test.proj
Build:
  Name=.NEXT
  Hello .NEXT
Build succeeded.
```

Targets

Зависимости через DependsOnTargets

```
<Project DefaultTargets="Build">
  <PropertyGroup>
    <BuildDependsOn>Compile</BuildDependsOn>
  </PropertyGroup>
  <Target Name="Compile">
    <Message Text="Compile"/>
  </Target>
  <Target Name="Build" DependsOnTargets="$(BuildDependsOn)">
    <Message Text="Build"/>
  </Target>
  <Target Name="PrepareResources">
    <Message Text="PrepareResources"/>
  </Target>
  <PropertyGroup>
    <BuildDependsOn>PrepareResources;$(BuildDependsOn)</BuildDependsOn>
  </PropertyGroup>
</Project>
```

```
$  msbuild /nologo test.proj
PrepareResources:
  PrepareResources
Compile:
  Compile
Build:
  Build
Build succeeded.
```

Targets

Зависимости через BeforeTargets и AfterTargets

```
<Project DefaultTargets="Build">
  <Target Name="Build">
    <Message Text="Build"/>
  </Target>
  <Target Name="Compile" BeforeTargets="Build">
    <Message Text="Compile"/>
  </Target>
  <Target Name="Publish" AfterTargets="Build">
    <Message Text="Publish"/>
  </Target>
</Project>
```

```
$  msbuild /nologo test.proj
Compile:
  Compile
Build:
  Build
Publish:
  Publish
Build succeeded.
```

Imports

Включение одного проекта в другой

```
<!--test.targets-->
<Project>
  <Target Name="Build">
    <Message Text="Build $(Configuration)"/>
  </Target>
</Project>
<!--test.proj-->
<Project DefaultTargets="Build">
  <PropertyGroup>
    <Configuration>Release</Configuration>
  </PropertyGroup>
  <Import Project="test.targets" />
</Project>
```

```
$  msbuild /nologo test.proj
Build:
  Build Release
Build succeeded.
```

Imports

DefaultTargets и InitialTargets

```
<!--test.targets-->
<Project DefaultTargets="Test1" InitialTargets="Test2">
  <Target Name="Test1">
    <Message Text="Test1"/>
  </Target>
  <Target Name="Test2">
    <Message Text="Test2"/>
  </Target>
  <Target Name="Build">
    <Message Text="Build"/>
  </Target>
</Project>
<!--test.proj-->
<Project DefaultTargets="Build">
  <Import Project="test.targets" />
</Project>
```

```
$  msbuild /nologo test.proj
Test2:
  Test2
Build:
  Build
Build succeeded.
```

Imports

Определение результатов импорта /preprocess

```
<!--test.targets-->
<Project DefaultTargets="Test1" InitialTargets="Test2">
  <Target Name="Build"><Message Text="Build"/></Target>
  <Target Name="Test1"><Message Text="Test1"/></Target>
  <Target Name="Test2"><Message Text="Test2"/></Target>
</Project>
<!--test.proj-->
<Project DefaultTargets="Build">
  <Target Name="T1"><Message Text="T1"/></Target>
  <Import Project="test.targets"/>
  <Target Name="T2"><Message Text="T2"/></Target>
</Project>
```

Imports

Определение результатов импорта /preprocess

```
$ msbuild /nologo /pp test.proj
<!--===/private/tmp/test.proj===-->
<Project DefaultTargets="Build" InitialTargets="Test2">
  <Target Name="T1">
    <Message Text="T1" />
  </Target>
  <!--=<Import Project="test.targets"> /private/tmp/test.targets=-->
  <Target Name="Build">
    <Message Text="Build" />
  </Target>
  <Target Name="Test1">
    <Message Text="Test1" />
  </Target>
  <Target Name="Test2">
    <Message Text="Test2" />
  </Target>
  <!--===  </Import> /private/tmp/test.proj===-->
  <Target Name="T2">
    <Message Text="T2" />
  </Target>
</Project>
```

Как MSBuild исполняет проект

1. Устанавливаются все глобальные свойства, переменные окружения и параметры командной строки

2. Вычисляются статические Properties и выполняются Imports

3. Вычисляются Item Definitions

4. Вычисляются Items

5. Вычисляются UsingTasks

6. Начинает исполнение Target

Важность порядка определения Items и Properties

```
<Project DefaultTargets="Build">
  <PropertyGroup>
    <DeployPath>$(OutputPath)</DeployPath>
  </PropertyGroup>
  <ItemGroup>
    <OutputPathItem Include="$(OutputPath)" />
  </ItemGroup>
  <PropertyGroup>
    <Configuration>Debug</Configuration>
    <OutputPath>bin\$(Configuration)\</OutputPath>
  </PropertyGroup>
  <Target Name="Build">
    <Message Text="Configuration: '$(Configuration)'" />
    <Message Text="OutputPath: '$(OutputPath)'"/>
    <Message Text="DeployPath: '$(DeployPath)'" />
    <Message Text="OutputPathItem: '@(OutputPathItem)'" />
    </Target>
</Project>
```

```
$  msbuild /nologo test.proj
Build:
  Configuration: 'Debug'
  OutputPath: 'bin/Debug/'
  DeployPath: ''
  OutputPathItem: 'bin/Debug/'
Build succeeded.
```

Task batching

Выполнение Task для каждой уникальной metadata

```
<Project DefaultTargets="Build">
  <ItemGroup>
    <SourceFiles Include="1.txt" Group="Group1" />
    <SourceFiles Include="2.txt" Group="Group1" />
    <SourceFiles Include="3.txt" Group="Group2" />
  </ItemGroup>
  <Target Name="Build">
    <Message Text="Build %(SourceFiles.Group) @(SourceFiles)" />
  </Target>
</Project>
```

```
$  msbuild /nologo test.proj
Build:
  Build Group1 1.txt;2.txt
  Build Group2 3.txt
Build succeeded.
```

Task batching

Сложности с Task batching

```
<Project DefaultTargets="Build">
  <ItemGroup>
    <SrcFiles Include="1.txt" Group="Group1" />
    <SrcFiles Include="2.txt" Group="Group2" />
    <DstFiles Include="3.txt" Group="Group1" />
    <DstFiles Include="4.txt" Group="Group2" />
  </ItemGroup>
  <Target Name="Build">
    <Message
      Text="BuildGroup: '%(SrcFiles.Group)', '%(DstFiles.Group)'
            SrcFiles='@(SrcFiles)' DstFiles='@(DstFiles)'" />
  </Target>
</Project>
```

```
$  msbuild /nologo test.proj
Build:
  BuildGroup: 'Group1', '' SrcFiles='1.txt' DstFiles=''
  BuildGroup: 'Group2', '' SrcFiles='2.txt' DstFiles=''
  BuildGroup: '', 'Group1' SrcFiles='' DstFiles='3.txt'
  BuildGroup: '', 'Group2' SrcFiles='' DstFiles='4.txt'
Build succeeded.
```

Task batching

Группировка Items с использованием Shared metadata

```
<Project DefaultTargets="Build">
  <ItemGroup>
    <SourceFiles Include="1.txt" DoAction="Action1" />
    <SourceFiles Include="2.txt" DoAction="Action1" />
    <SourceFiles Include="3.txt" DoAction="Action2" />
  </ItemGroup>
  <Target Name="Build">
    <Message Text="Action1: SourceFiles=‘@(SourceFiles)’”
      Condition="'%(DoAction)' == 'Action1'" />
    <Message Text="Action2: SourceFiles=‘@(SourceFiles)'"
      Condition="'%(DoAction)' == 'Action2'" />
  </Target>
</Project>
```

```
$  msbuild /nologo test.proj
Build:
  Action1: SourceFiles='1.txt;2.txt'
  Action2: SourceFiles='3.txt'
Build succeeded.
```

Task batching

Проблемы группировки с использованием Shared metadata

```
<Project DefaultTargets="Build">
  <ItemGroup>
    <SourceFiles Include="1.txt" DoAction="Action1" />
    <SourceFiles Include="2.txt" DoAction="Action1" />
    <SourceFiles Include="3.txt" DoAction="Action2" />
  </ItemGroup>
  <Target Name="Build">
    <ItemGroup>
      <SourceFiles Include="4.txt"/>
    </ItemGroup>
    <Message Text="Action1: SourceFiles='@(SourceFiles)'"
        Condition="'%(DoAction)' == 'Action1'" />
    <Message Text="Action2: SourceFiles='@(SourceFiles)'"
        Condition="'%(DoAction)' == 'Action2'" />
  </Target>
</Project>
```

Task batching

Проблемы группировки с использованием Shared metadata

```
$  msbuild /nologo test.proj
/private/tmp/test.proj(11,9): error MSB4096: The item "4.txt"
in item list "SourceFiles" does not define a value for metadata
"DoAction".  In order to use this metadata, either qualify it
by specifying %(SourceFiles.DoAction), or ensure that all items
in this list define a value for this metadata.

Build FAILED.
```

Task batching

Решение с помощью ItemDefinitionGroup

```
<Project DefaultTargets="Build">
  <ItemGroup>
    <SourceFiles Include="1.txt" DoAction="Action1" />
    <SourceFiles Include="2.txt" DoAction="Action1" />
    <SourceFiles Include="3.txt" DoAction="Action2" />
  </ItemGroup>
  <ItemDefinitionGroup>
    <SourceFiles>
      <DoAction>None</DoAction>
    </SourceFiles>
  </ItemDefinitionGroup>
  <Target Name="Build">
    <ItemGroup>
      <SourceFiles Include="4.txt"/>
    </ItemGroup>
    <Message Text="Action1: SourceFiles=‘@(SourceFiles)'"
             Condition="'%(DoAction)' == 'Action1'" />
    <Message Text="Action2: SourceFiles=‘@(SourceFiles)'"
             Condition="'%(DoAction)' == 'Action2'" />
  </Target>
</Project>
```

Task batching

Решение с помощью ItemDefinitionGroup

```
$  msbuild /nologo test.proj
Build:
  Action1: SourceFiles='1.txt;2.txt'
  Action2: SourceFiles='3.txt'

Build succeeded.
```

Target batching

Выполнение Target для каждой уникальной metadata

```
<Project DefaultTargets="Build">
  <ItemGroup>
    <Configuration Include="Debug"/>
    <Configuration Include="Release"/>
  </ItemGroup>
  <Target Name="Build" Outputs="%(Configuration.Identity)">
    <Message Text="Build @(Configuration)"/>
  </Target>
</Project>
```

```
$  msbuild /nologo test.proj
Build:
  Build Debug
Build:
  Build Release
Build succeeded.
```

Incremental build

Инкрементальный билд на основе Inputs и Outputs

```
<Target Name="Build"
  Inputs="@(SourceFiles)"
  Outputs=
    "@(SourceFiles->'$(Dest)\%(RecursiveDir)%(Filename)%(Extension)')">

  <Message Text="SourceFiles='@(SourceFiles)'"/>
  <Copy SourceFiles="@(SourceFiles)"
    DestinationFiles=
   "@(SourceFiles->'$(Dest)\%(RecursiveDir)%(Filename)%(Extension)')" />
</Target>
```

Incremental build

Инкрементальный билд на основе Inputs и Outputs

```
$ msbuild /nologo test.proj
Build:
  SourceFiles=‘src/1.txt;src/2.txt;src/sub/1.txt;src/sub/2.txt’
  Creating directory "dest".
  Copying file from "src/1.txt" to "dest/1.txt".
  Copying file from "src/2.txt" to "dest/2.txt".
  Creating directory "dest/sub".
  Copying file from "src/sub/1.txt" to "dest/sub/1.txt".
  Copying file from "src/sub/2.txt" to "dest/sub/2.txt".
Build succeeded.
$ msbuild /nologo test.proj
Build:
Skipping target "Build" because all output files are up-to-date
with respect to the input files.
Build succeeded.
$ touch src/1.txt
$ msbuild /nologo test.proj
Build:
Building target "Build" partially, because some output files are
out of date with respect to their input files.
  SourceFiles=‘src/1.txt’
  Copying file from "src/1.txt" to "dest/1.txt".
Done Building Project "/private/tmp/test.proj" (default targets).
Build succeeded.
```

Incremental build

Дополнительная информация о инкрементальных билдах

- Инкрементальность C++ и FileTracker

- Когда пишите свои собственные Task, которые производят какие-либо файлы обязательно добавляйте их в Item забывайте FileWrites чтобы корректно работал Clean

Встраиваем внешний генератор в билд

Наивный путь: вызываем генератор с помощью Exec

```
<Project DefaultTargets="Build">
  <ItemGroup>
    <GeneratorSources Include="*.mdl" />
    <Compile Include="1.cs" />
    <Compile Include="generated\*.cs">
      <SubType>Code</SubType>
      <Link>src\CSharp\Stub</Link>
    </Compile>
  </ItemGroup>
  <Target Name="Generate" BeforeTargets="Build">
    <RemoveDir Directories="generated" />
    <MakeDir Directories="generated" />
    <Exec
 Command="touch generated\%(GeneratorSources.Identity).gen.cs"/>
  </Target>
  <Target Name="Build">
    <Message Text="Compile='@(Compile)'" />
  </Target>
</Project>
```

Встраиваем внешний генератор в билд

Наивный путь: вызываем генератор с помощью Exec

```
$  msbuild /nologo test.proj
Generate:
  Directory "generated" doesn't exist. Skipping.
  Creating directory "generated".
  echo 1 > generated/1.mdl.gen.cs
  echo 2 > generated/2.mdl.gen.cs
Build:
  Compile=‘1.cs'
Build succeeded.
```

Встраиваем внешний генератор в билд

Добавляем сгенерированные Items в Compile

```
<Target Name="Generate" BeforeTargets="Build">
  <RemoveDir Directories="generated" />
  <MakeDir Directories="generated" />
  <Exec
Command="touch generated\%(GeneratorSources.Identity).gen.cs" />
  <!-- Добавляем сгенерированные Items в Compile -->
  <ItemGroup>
    <Compile Include="generated\*.cs">
      <SubType>Code</SubType>
      <Link>src\CSharp\Stub</Link>
    </Compile>
  </ItemGroup>
</Target>
```

```
$  msbuild /nologo test.proj
Generate:
  Removing directory "generated".
  Creating directory "generated".
  echo 1 > generated/1.mdl.gen.cs
  echo 2 > generated/2.mdl.gen.cs
Build:
  Compile=‘1.cs;generated/1.mdl.gen.cs;generated/
2.mdl.gen.cs;generated/1.mdl.gen.cs;generated/2.mdl.gen.cs’
Build succeeded.
```

Встраиваем внешний генератор в билд

Очищаем Compile от дубликатов

```
<Target Name="Generate" BeforeTargets="Build">
  <RemoveDir Directories="generated" />
  <MakeDir Directories="generated" />
  <Exec
Command="touch generated\%(GeneratorSources.Identity).gen.cs" />
  <ItemGroup>
    <Compile Include="generated\*.cs">
      <SubType>Code</SubType>
      <Link>src\CSharp\Stub</Link>
    </Compile>
    <!-- Очищаем CompileX ->
    <CompileX Remove="@(CompileX)"/>
  </ItemGroup>
  <RemoveDuplicates Inputs=“@(Compile)">
    <!-- Очищаем Compile от дубликатов -->
    <Output TaskParameter="Filtered" ItemName="CompileX" />
  </RemoveDuplicates>
  <ItemGroup>
    <!-- Очищаем Compile ->
    <Compile Remove=“@(Compile->Identity)”/>
    <!-- Заполняем Compile -->
   <Compile Include="@(CompileX)"/>
  </ItemGroup>
</Target>
```

Встраиваем внешний генератор в билд

Очищаем Compile от дубликатов

```
$  msbuild /nologo test.proj
Generate:
  Removing directory "generated".
  Creating directory "generated".
  echo 1 > generated/1.mdl.gen.cs
  echo 2 > generated/2.mdl.gen.cs
Build:
  Compile='1.cs;generated/1.mdl.gen.cs;generated/2.mdl.gen.cs;'
Build succeeded.
```

```
$  rm 2.mdl
$  msbuild /nologo test.proj
Build started 5/17/2017 3:04:47 PM.
Generate:
  Removing directory "generated".
  Creating directory "generated".
  echo 1 > generated/1.mdl.gen.cs
Build:
  Compile='1.cs;generated/1.mdl.gen.cs;generated/2.mdl.gen.cs'
```

Встраиваем внешний генератор в билд

Проверяем что все файлы из Compile существуют

```
<Target Name="Generate" BeforeTargets=“Build”>
  ...
  <RemoveDuplicates Inputs="@(Compile)">
    <Output TaskParameter="Filtered" ItemName="CompileX" />
  </RemoveDuplicates>
  <ItemGroup>
    <!-- Проверяем наличие файла перед включением в Compile -->
    <Compile Include="@(CompileX)"
      Condition="Exists('%(CompileX.FullPath)')"/>
  </ItemGroup>
</Target>
```

```
$  msbuild /nologo test.proj
Generate:
  Removing directory "generated".
  Creating directory "generated".
  echo 1 > generated/1.mdl.gen.cs
Build:
  Compile='1.cs;generated/1.mdl.gen.cs'
Build succeeded.
```

Встраиваем внешний генератор в билд

Пробуем добавить Incremental build

```
<Target Name="Generate"
  BeforeTargets="Build"
  Inputs="@(GeneratorSources)"
  Outputs="@(Compile)">
    ...
</Target>
```

```
$  msbuild /nologo test.proj
Generate:
  Removing directory "generated".
  Creating directory "generated".
  echo 1 > generated/1.mdl.gen.cs
Build:
  Compile='1.cs;generated/1.mdl.gen.cs'
Build succeeded.

$  msbuild /nologo test.proj
Generate:
  Removing directory "generated".
  Creating directory "generated".
  echo 1 > generated/1.mdl.gen.cs
Build:
  Compile='1.cs;generated/1.mdl.gen.cs'
Build succeeded.
```

Встраиваем внешний генератор в билд

Сохраняем результаты последней генерации в файл

```
<Target Name="Generate" BeforeTargets=“Build"
  Inputs="@(GeneratorSources)" Outputs="@(PrevOutputs)">
  ...
  <ItemGroup>
    <Compile Include="@(CompileX)"
      Condition="Exists('%(CompileX.FullPath)')"/>
    <_GeneratorOutputItems Include="generated\*.cs" />
  </ItemGroup>
  <WriteLinesToFile
    File=“prevInputs.dump"
    Lines=“@(GeneratorSources)"
    Overwrite="True" />
  <WriteLinesToFile
    File="prevOutputs.dump"
    Lines=“@(_GeneratorOutputItems)"
    Overwrite="True" />
</Target>
```

Встраиваем внешний генератор в билд

Читаем результаты предыдущей генерации

```
<Target Name="LoadPreviousItems" BeforeTargets="Generate">
  <ReadLinesFromFile
    File="prevOutputs.dump"
    Condition="Exists('prevOutputs.dump')">
      <Output TaskParameter="Lines" ItemName="PrevOutputs" />
  </ReadLinesFromFile>
  <ReadLinesFromFile
    File="prevInputs.dump"
    Condition="Exists('prevInputs.dump')">
      <Output TaskParameter="Lines" ItemName="PrevInputs" />
  </ReadLinesFromFile>
  <PropertyGroup>
    <_ShouldRegen>False</_ShouldRegen>
    <_ShouldRegen Condition="'@(PrevOutputs)'==''">True</_ShouldRegen>
    <_ShouldRegen
Condition="'@(PrevInputs)'!='@(GeneratorSources)'">True</_ShouldRegen>
  </PropertyGroup>
  <ItemGroup>
    <PrevOutputs Include="$([System.Guid]::NewGuid())"
      Condition="$(_ShouldRebuildModel) == True" />
    </ItemGroup>
</Target>
```

Встраиваем внешний генератор в билд

```
$  msbuild /nologo test.proj
Generate:
  Removing directory "generated".
  Creating directory "generated".
  echo 1 > generated/1.mdl.gen.cs
Build:
  Compile=‘1.cs;generated/1.mdl.gen.cs;’
Build succeeded.

$  msbuild /nologo test.proj
Generate:
Skipping target "Generate" because all output files are up-to-
date with respect to the input files.
Build:
  Compile='1.cs;generated/1.mdl.gen.cs;'
Build succeeded.
```

Отладка билд скриптов

Логгеры:

- FileLogger
- ConsoleLogger
- BinaryLogger
- CustomLogger

Уровни логирования:

- Quiet
- Minimal
- Normal
- Detailed
- Diagnostic

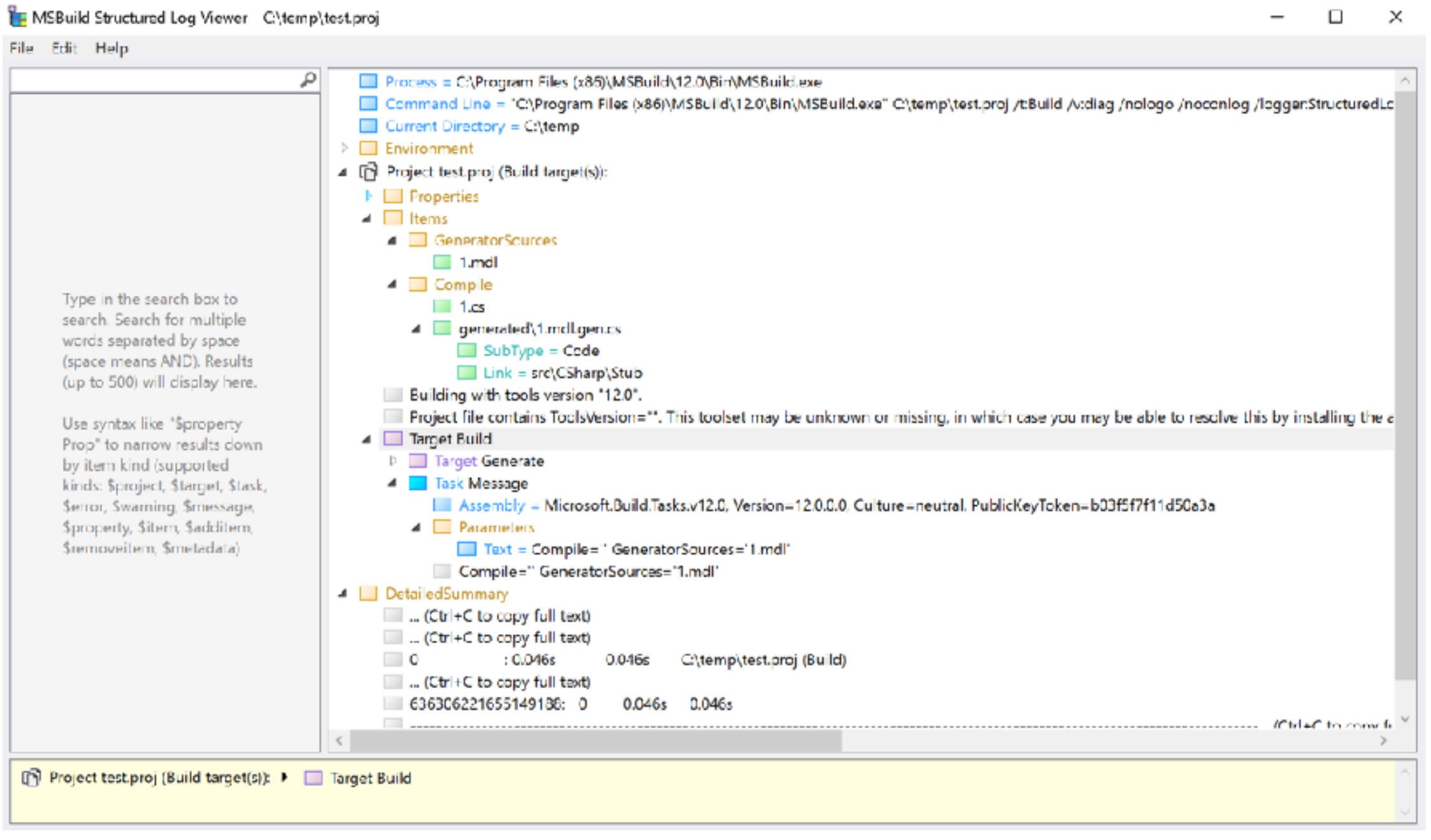
Отладка билд скриптов

```
$ msbuild /nologo /v:detailed test.proj
Evaluation started ("/private/tmp/test.proj")
Evaluation finished ("/private/tmp/test.proj")
Project "/private/tmp/test.proj" on node 1 (default targets).
Building with tools version "15.0".
Project file contains ToolsVersion="". This toolset may be
unknown or missing, in which case you may be able to resolve
this by installing the appropriate version of MSBuild, or the
build may have been forced to a particular ToolsVersion for
policy reasons. Treating the project as if it had
ToolsVersion="15.0". For more information, please see http://
go.microsoft.com/fwlink/?LinkId=293424.
Target "Build" in project "/private/tmp/test.proj" (entry
point):
Using "Message" task from assembly "Microsoft.Build.Tasks.Core,
Version=15.1.0.0, Culture=neutral,
PublicKeyToken=b03f5f7f11d50a3a".
Task "Message"
  Build
Done executing task "Message".
Done building target "Build" in project "test.proj".
Done Building Project "/private/tmp/test.proj" (default
targets).
Build succeeded.
```

Отладка билд скриптов

Полезный инструмент: MSBuildStructuredLog

https://github.com/KirillOsenkov/MSBuildStructuredLog



Отладка билд скриптов

Отладка собственных Task и Logger, а так же внутренностей MSBuild возможна:

- Под Windows с помощью специальной переменной окружения: MSBUILDDEBUGONSTART=1
- Под mono: --debug --debugger-agent=transport=dt_socket, server=y, suspend=y, address=127.0.0.1:55555

MSBuild + .NET Core

Было:

```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="12.0" DefaultTargets="Build" xmlns="http://schemas.microsoft.co
  <Import Project="$(MSBuildExtensionsPath)\$(MSBuildToolsVersion)\Microsoft.Common.pr
    <PropertyGroup>
        <Configuration Condition=" '$(Configuration)' == '' ">Debug</Configuration>
        <Platform Condition=" '$(Platform)' == '' ">AnyCPU</Platform>
        <ProjectGuid>{889D5F85-6ABC-481E-90CD-CB92482B07AC}</ProjectGuid>
                <ProjectTypeGuids>{FAE04EC0-301F-11D3-BF4B-00C04F79EFBC}</ProjectTypeG
        <OutputType>Exe</OutputType>
        <AppDesignerFolder>Properties</AppDesignerFolder>
        <RootNamespace>ConscleApplication7</RootNamespace>
        <AssemblyName>ConsoleApplication7</AssemblyName>
        <TargetFrameworkVersion>v4.5</TargetFrameworkVersion>
        <FileAlignment>512</FileAlignment>
    </PropertyGroup>
    <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">
        <PlatformTarget>AnyCPU</PlatformTarget>
        <DebugSymbols>true</DebugSymbols>
        <DebugType>full</DebugType>
        <Optimize>false</Optimize>
        <OutputPath>bin\Debug\</OutputPath>
        <DefineConstants>DEBUG;TRACE</DefineConstants>
        <ErrorReport>prompt</ErrorReport>
        <WarningLevel>4</WarningLevel>
    </PropertyGroup>
    <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Release|AnyCPU' ">
```

Стало:

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp1.1</TargetFramework>
  </PropertyGroup>
</Project>
```

Особенности .NET Core

Используем пакеты, не собранные под .NET Standard

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>netstandard1.6</TargetFramework>
    <OutputType>Library</OutputType>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference
      Include="Newtonsoft.Json " Version="8.0.3" />
  </ItemGroup>
</Project>
```

Особенности .NET Core

Используем пакеты, не собранные под .NET Standard

```
$ dotnet restore test.proj
  Restoring packages for /private/tmp/test.proj...
/usr/local/share/dotnet/sdk/1.0.3/NuGet.targets(97,5): error :
Package Newtonsoft.Json 8.0.3 is not compatible with
netstandard1.6 (.NETStandard,Version=v1.6). Package
Newtonsoft.Json 8.0.3 supports: [/private/tmp/test.proj]
  Errors in /private/tmp/test.proj
      Package Newtonsoft.Json 8.0.3 is not compatible with
netstandard1.6 (.NETStandard,Version=v1.6). Package
Newtonsoft.Json 8.0.3 supports:
        - net20 (.NETFramework,Version=v2.0)
        - net35 (.NETFramework,Version=v3.5)
        - net40 (.NETFramework,Version=v4.0)
        - net45 (.NETFramework,Version=v4.5)
        - portable-dnxcore50+net45+win8+wp8+wpa81
(.NETPortable,Version=v0.0,Profile=net45+wp80+win8+wpa81+dnxcor
e50)
        - portable-net40+sl5+win8+wp8+wpa81
(.NETPortable,Version=v0.0,Profile=Profile328)
      One or more packages are incompatible
with .NETStandard,Version=v1.6.
```

Особенности .NET Core

Указываем Property PackageTargetFallback

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>netstandard1.6</TargetFramework>
    <OutputType>Library</OutputType>
    <PackageTargetFallback
      Condition="'$(TargetFramework)'=='netstandard1.6'">
            portable-net45+win81
    </PackageTargetFallback>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference
      Include="Newtonsoft.Json " Version="8.0.3" />
  </ItemGroup>
</Project>
```

```
$ dotnet restore test.proj
  Restoring packages for /private/tmp/test.proj...
  Generating MSBuild file /private/tmp/obj/
test.proj.nuget.g.props.
  Writing lock file to disk. Path: /private/tmp/obj/
project.assets.json
  Restore completed in 314.82 ms for /private/tmp/test.proj.
```

Особенности .NET Core

Важно ли расширение файла в проектах с Sdk

```
<Project Sdk="Microsoft.NET.Sdk">
    <PropertyGroup>
        <TargetFramework>netstandard1.6</TargetFramework>
        <OutputType>Library</OutputType>
    </PropertyGroup>
</Project>
```

```
$ dotnet build test.proj
Microsoft (R) Build Engine version 15.1.1012.6693
Copyright (C) Microsoft Corporation. All rights reserved.

/usr/local/share/dotnet/sdk/1.0.3/
Microsoft.Common.CurrentVersion.targets(2630,7): error MSB4057:
The target "CreateManifestResourceNames" does not exist in the
project. [/private/tmp/test.proj]

Build FAILED.

    0 Warning(s)
    1 Error(s)
```

Особенности .NET Core

Разбираемся в чем проблема в файле: Sdks/Microsoft.NET.Sdk/Sdk/Sdk.targets

```
<PropertyGroup Condition="'$(LanguageTargets)' == ''">
  <LanguageTargets
       Condition="'$(MSBuildProjectExtension)' == '.csproj'">
    $(MSBuildToolsPath)\Microsoft.CSharp.targets
  </LanguageTargets>
  <LanguageTargets
      Condition="'$(MSBuildProjectExtension)' == '.vbproj'">
    $(MSBuildToolsPath)\Microsoft.VisualBasic.targets
  </LanguageTargets>
    <!-- If LanguageTargets isn't otherwise set, then just import
the common targets.
     This should allow the restore target to run,
     which could bring in NuGet packages that set the
LanguageTargets to something else.
     This means support for different
     languages could either be supplied via an SDK or via a NuGet
package. -->
    <LanguageTargets Condition="'$(LanguageTargets)' == ''">
        $(MSBuildToolsPath)\Microsoft.Common.CurrentVersion.targets
    </LanguageTargets>
</PropertyGroup>
```

Особенности .NET Core

Чиним проблему с расширением файла

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>netstandard1.6</TargetFramework>
    <OutputType>Library</OutputType>
    <LanguageTargets>
      $(MSBuildToolsPath)\Microsoft.CSharp.targets
    </LanguageTargets>
  </PropertyGroup>
</Project>
```

```
$ dotnet build test.proj
Microsoft (R) Build Engine version 15.1.1012.6693
Copyright (C) Microsoft Corporation. All rights reserved.

  test -> /private/tmp/bin/Debug/netstandard1.6/test.dll

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

Особенности .NET Core

Легко пакуем NuGet в артефакты билда

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>netstandard1.6</TargetFramework>
    <OutputType>Library</OutputType>
  </PropertyGroup>
  <PropertyGroup>
    <PackageId>MSBuildDotNetPackage</PackageId>
    <Authors>Mikhail Filippov</Authors>
  </PropertyGroup>
</Project>
```

```
$ dotnet pack test.csproj
Microsoft (R) Build Engine version 15.1.1012.6693
Copyright (C) Microsoft Corporation. All rights reserved.

  test -> /private/tmp/bin/Debug/netstandard1.6/test.dll

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

Особенности .NET Core

Не работает Target BeforeBuild

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFrameworks>netstandard1.6</TargetFrameworks>
    <OutputType>Library</OutputType>
  </PropertyGroup>
  <Target Name="BeforeBuild">
    <Error Text="This aproar didn't work"/>
  </Target>
  <Target Name="GenerateMyStuff" BeforeTargets="PrepareResources">
    <Message Text="From GenerateMyStuff target" Importance="High"/>
  </Target>
</Project>
```

```
$ dotnet build  test.csproj
Microsoft (R) Build Engine version 15.1.1012.6693
Copyright (C) Microsoft Corporation. All rights reserved.

  From GenerateMyStuff target
  test -> /private/tmp/test1/bin/Debug/netstandard1.6/test.dll

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

Особенности .NET Core

Что на самом деле такое “PackageReference”.
Как это связано с obj/project.assets.json.
Почему в .NET Core проектах всегда нужно помнить про dotnet restore.

Нужно быть очень аккуратным

История одного бага

```
$ msbuild /nologo ConsoleApplication20.sln
ValidateSolutionConfiguration:
  Building solution configuration "Debug|Any CPU".
Project "/tmp/ConsoleApplication20/
CoreCompile:
Skipping target "CoreCompile" because all output files are up-
to-date with respect to the input files.
CopyFilesToOutputDirectory:
  ConsoleApplication20 -> /tmp/ConsoleApplication20/
ConsoleApplication20/bin/Debug/ConsoleApplication20.exe
  Copying file from "obj/Debug/ConsoleApplication20.pdb" to
"bin/Debug/ConsoleApplication20.pdb".
IncrementalClean:
  Deleting file "/tmp/ConsoleApplication20/
ConsoleApplication20/bin/Debug//ConsoleApplication20.pdb".
Done Building Project "/tmp/ConsoleApplication20/
ConsoleApplication20/ConsoleApplication20.csproj" (default
targets).
Done Building Project "/tmp/ConsoleApplication20/
ConsoleApplication20.sln" (default targets).

Build succeeded.
```

Нужно быть очень аккуратным

Одно чиним - другое ломаем 😄

https://github.com/mono/msbuild/commit/9836044be2ca09a4df2405001b93372f092800ef

FixFilePath for %(Filename) item modifier

```
This manifested as a bug when building any project with Xamarin Studio
on OSX.  The debug file from `obj/x86/Debug/Console009.pdb` would end up
in `bin/Debug/obj/x86/Debug/Console009.pdb`, instead of
`bin/Debug/Console009.pdb`. A subsequent build worked correctly though!
And it would build correctly on the command line.

Looking at the logs, one of the differences was that in the first build
we had ..

    BaseIntermediateOutputPath     = obj\

.. and the second one had

    BaseIntermediateOutputPath     = obj/

This also affected `_DebugSymbolsIntermediatePath` which ended up with
backslashes, `obj\x86\Debug\Console009.pdb`.

    <_DebugSymbolsOutputPath Include="@(_DebugSymbolsIntermediatePath->'$(OutDir)%(Filename)%(Extension)')" />

So, when `_DebugSymbolsOutputPath` tried to get `%(Filename)` in the
transform, it ended up with `obj\x86\Debug\Console009` as the filename,
because of the backlashes.
```

```
$ cat obj/Debug/ConsoleApplication20.csproj.FileListAbsolute.txt
ConsoleApplication20/bin/Debug//ConsoleApplication20.pdb
ConsoleApplication20/bin/Debug//ConsoleApplication20.exe
ConsoleApplication20/obj/Debug/ConsoleApplication20.exe
ConsoleApplication20/obj/Debug/ConsoleApplication20.pdb
```

Выводы

- Хорошо написанные билд скрипты экономят время разработчикам и позволяют решать нетривиальные задачи
- MSBuild активно развивается, и вы можете принять в этом участие: https://github.com/Microsoft/msbuild
- Изучайте инструменты которыми пользуетесь это может сэкономить много времени

Вопросы?