LEARNING msbuild

#msbuild

Table of Contents

About

Chapter 1: Getting started with msbuild

Remarks

This section provides an overview of what msbuild is, and why a developer might want to use it.

It should also mention any large subjects within msbuild, and link out to the related topics. Since the Documentation for msbuild is new, you may need to create initial versions of those related topics.

Examples

Installation or Setup

MSBuild 2015

On Windows there are three choices to get MSBuild:

- Install Visual Studio 2015
- Download Microsoft Build Tools which includes VB and C# compilers.
- Build from Source

On Linux

- Build from Source using this guide

Creating Custom MSBuild Targets

```
<PropertyGroup>
     <!-- Definition of a Property named "TestCondition". A PropertyGroup may also be placed
inside a Target. -->
    <TestCondition>True</TestCondition>
</PropertyGroup>

<!-- This Target will run after the "Clean" Target, subject to a Condition. -->
<Target Name="SpecificTarget" AfterTargets="Clean" Condition=" '$(TestCondition)' == 'True' ">
    <!-- Displaying a custom message -->
    <Message Text="Here is my Specific Target" Importance="Low" />
    <!-- Here come your specific code. -->
</Target>
```

Hello World

HelloWorld.proj

```
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003"
DefaultTargets="SayHello">
```

```
    <!-- Properties can be passed as command line parameters. i.e. /p:Name=MyName
    or /p:Name="My Name" (Use quotes if the value includes spaces) -->
    <PropertyGroup>
        <Name Condition="'$(Name)'==''">Rolo</Name>
    </PropertyGroup>

    <!-- Items can't be passed as command line parameters. -->
    <!-- Items can include metadata. i.e. URL -->
    <ItemGroup>
        <People Include="World"/>
        <People Include="StackOverflow">
            <URL>http://stackoverflow.com</URL>
        </People>
        <People Include="Google">
            <URL>http://google.com</URL>
        </People>
    </ItemGroup>

    <!-- Targets can be called using it's name. i.e. /t:SayHello -->
    <Target Name="SayHello">
        <!-- You can have as many Tasks as required inside a Target. -->
        <!-- Tasks can be executed conditionally. -->
        <Message Condition="'%(People.URL)'==''" Text="Hello %(People.Identity), my name is
$(Name)! "/>
        <Message Condition="'%(People.URL)'!=''" Text="Hello %(People.Identity), my name is
$(Name)!. Your URL is %(People.URL) "/>
    </Target>
</Project>
```

Execute with:

- msbuild HelloWorld.proj
- msbuild HelloWorld.proj /p:Name="John Doe"
- msbuild HelloWorld.proj /p:Name="Batman" /t:SayHello

Read Getting started with msbuild online: https://riptutorial.com/msbuild/topic/1150/getting-started-with-msbuild

Chapter 2: Common Item Types: ProjectReference

Introduction

A `ProjectReference` defines a reference to another project.

Parameters

Parameter	Details
`Include` (attribute)	Path to project file
`Project` (metadata)	Project GUID, in the form {00000000-0000-0000-0000-000000000000}
`ReferenceOutputAssembly` (metadata)	Boolean specifying whether the outputs of the project referenced should be passed to the compiler. Default is true.
`SpecificVersion` (metadata)	Whether the exact version of the assembly should be used.
`Targets` (metadata)	Semicolon-separated list of targets in the referenced projects that should be built. Default is the value of `$(ProjectReferenceBuildTargets)` whose default is blank, indicating the default targets.
`OutputItemType` (metadata)	Item type to emit target outputs into. Default is blank. If `ReferenceOutputAssembly` is set to "true" (default) then target outputs will become references for the compiler.
`EmbedInteropTypes` (metadata)	Optional boolean. Whether the types in this reference need to embedded into the target assembly - interop asemblies only

Remarks

When the `OutputItemType` parameter is used, additional parameters (metadata) may be applicable. For example, when `OutputItemType` is set to `Content`, `CopyToOutputDirectory` can be used.

Parameter	Details
`CopyToOutputDirectory` (metadata)	Optional string. Determines whether to copy the file to the output directory. Values: `Never`, `Always`, `PreserveNewest`.

Examples

Simple ProjectReference

```
<ItemGroup>
  <ProjectReference Include="Foo.csproj">
    <Project>{01234567-0123-0123-0123-0123456789AB}</Project>
    <Name>Foo</Name>
  </ProjectReference>
</ItemGroup>
```

Read Common Item Types: ProjectReference online:
https://riptutorial.com/msbuild/topic/9236/common-item-types--projectreference

Chapter 3: Frequently-used Tasks

Examples

Copying files

```
<ItemGroup>
     <DataToCopy Include="*.cs;*.aspx" />
</ItemGroup>
<Copy SourceFiles="@(DataToCopy)" DestinationFolder="SourceCopiedFolder" />
```

Deleting files

```
<ItemGroup>
    <FilesToDelete Include="*.tmp" />
</ItemGroup>

<Delete Files="@(FilesToDelete)" />
```

Creating a new directory

```
<PropertyGroup>
    <DirectoryToCreate>NewDirectory</DirectoryToCreate>
</PropertyGroup>
<MakeDir Directories="$(DirectoryToCreate)" />
```

Removing an existing directory

```
<PropertyGroup>
    <DirectoryToRemove>TempData</DirectoryToRemove>
</PropertyGroup>
<RemoveDir Directories="$(DirectoryToRemove)" />
```

Running a custom command

```
<Exec Command="echo Hello World" />
```

Displaying a custom message

```
<PropertyGroup>
    <CustomMessage>Hello World</CustomMessage>
    <MessageImportance>Low</MessageImportance> <!-- Low / Normal / High -->
</PropertyGroup>
<Message Text="$(CustomMessage)" Importance="$(MessageImportance)" />
```

Running MSBuild on another project / solution

```
<PropertyGroup>
    <LinkedSolution>LinkedSolution.sln</LinkedSolution>
    <BuildType>Build</BuildType> <!-- Build / Rebuild -->
    <BuildArchitecture>x86</BuildArchitecture> <!-- x86 / 64 -->
    <BuildConfiguration>Debug</BuildConfiguration> <!-- Debug / Release -->
</PropertyGroup>
<MSBuild Projects="$(LinkedSolution)"
         Targets="$(BuildType)"
         Properties="Architecture=$(BuildArchitecture);Configuration=$(BuildConfiguration)" />
```

Read Frequently-used Tasks online: https://riptutorial.com/msbuild/topic/5196/frequently-used-tasks

Chapter 4: Order of Property and Item Evaluation

Remarks

For more detail, see Property and Item Evaluation Order on the MSDN documentation page *Comparing Properties and Items*.

Examples

Example illustrating the order of evaluation

MSBuild evaluates `PropertyGroup`, `Choose` and `ItemGroup` elements that are directly under the `Project` element before those that are in `Target` elements.

- Directly under the `Project` element, `PropertyGroup` and `Choose` elements are evaluated in the order in which they appear, and then `ItemGroup` elements are evaluated in the order in which they appear.
- In `Target` elements `PropertyGroup` and `ItemGroup` share equal precedence and are evaluated in the order in which they appear.

Within files referenced via `Import`, MSBuild evaluates `PropertyGroup`, `Choose` and `ItemGroup` in the same manner as above, and as though the imported files' content appeared inline where the `Import` is located.

The comments below provide property values and item counts before and after MSBuild evaluates selected lines.

```
<Project DefaultTargets="FooTarget"
xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
    <Target Name="FooTarget">
        <ItemGroup>
            <!-- '$(FooProp)' == '2', '@(FooItem->Count())' == '1' -->
            <FooItem Include="foo value B" />
            <!-- '$(FooProp)' == '2', '@(FooItem->Count())' == '2' -->
        </ItemGroup>
        <PropertyGroup>
            <!-- '$(FooProp)' == '2', '@(FooItem->Count())' == '2' -->
            <FooProp>3</FooProp>
            <!-- '$(FooProp)' == '3', '@(FooItem->Count())' == '2' -->
        </PropertyGroup>
    </Target>
    <ItemGroup>
        <!-- '$(FooProp)' == '2', '@(FooItem->Count())' == '0' -->
        <FooItem Include="foo value A" />
        <!-- '$(FooProp)' == '2', '@(FooItem->Count())' == '1' -->
    </ItemGroup>
    <PropertyGroup>
        <!-- '$(FooProp)' == '', '@(FooItem->Count())' == '0' -->
        <FooProp>1</FooProp>
        <!-- '$(FooProp)' == '1', '@(FooItem->Count())' == '0' -->
    </PropertyGroup>
    <Choose>
        <When Condition=" '$(FooProp)' == '1' ">
            <!-- '$(FooProp)' == '1', '@(FooItem->Count())' == '0' -->
            <FooProp>2</FooProp>
            <!-- '$(FooProp)' == '2', '@(FooItem->Count())' == '0' -->
        </When>
    </Choose>
</Project>
```

Read Order of Property and Item Evaluation online:
https://riptutorial.com/msbuild/topic/6262/order-of-property-and-item-evaluation

Chapter 5: Order of Target Execution

Remarks

From MSDN: Target Build Order

Determining the Target Build Order

MSBuild determines the target build order as follows:

1. InitialTargets targets are run.
2. Targets specified on the command line by the /target switch are run. If you specify no targets on the command line, then the DefaultTargets targets are run. If neither is present, then the first target encountered is run.
3. The Condition attribute of the target is evaluated. If the Condition attribute is present and evaluates to false, the target isn't executed and has no further effect on the build.
4. Before a target is executed, its DependsOnTargets targets are run.
5. Before a target is executed, any target that lists it in a BeforeTargets attribute is run.
6. Before a target is executed, its Inputs attribute and Outputs attribute are compared. If MSBuild determines that any output files are out of date with respect to the corresponding input file or files, then MSBuild executes the target. Otherwise, MSBuild skips the target.
7. After a target is executed or skipped, any target that lists it in an AfterTargets attribute is run.

Examples

DependsOnTargets

Define a sequence of Targets (`Target1`, then `Target2`) that must execute before `Target3`. Note that an execution request for `Target3` is required to cause `Target1` and `Target2` to be executed.

```
<Target Name="Target3" DependsOnTargets="Target1;Target2">
</Target>

<Target Name="Target2">
</Target>

<Target Name="Target1">
</Target>
```

AfterTargets

Define a Target (`Target1`) for which an execution request will cause `Target2` to be executed afterward.

```
<Target Name="Target2" AfterTargets="Target1">
</Target>

<Target Name="Target1">
</Target>
```

BeforeTargets

Define a Target (`Target2`) for which an execution request will cause `Target1` to be executed beforehand.

```
<Target Name="Target2">
</Target>

<Target Name="Target1" BeforeTargets="Target2">
</Target>
```

Read Order of Target Execution online: https://riptutorial.com/msbuild/topic/6611/order-of-target-execution

Credits

S. No	Chapters	Contributors
1	Getting started with msbuild	Community, Didier Aupest, Eugenio Miró, Rolo, weir
2	Common Item Types: ProjectReference	weir
3	Frequently-used Tasks	Didier Aupest, weir
4	Order of Property and Item Evaluation	weir
5	Order of Target Execution	weir