MSBuild en Visual Studio

Automatización de la construcción

Realizado por: Adrián Rodríguez Escudero
Asignatura: Herramientas y Métodos de la Ingeniería del Software

Índice

1. Información general sobre MSBuild

Microsoft Build Engine (MSBuild) en la nueva plataforma de compilación de Microsoft y Visual Studio. MSBuild es completamente transparente en cuanto al modo en que procesa y compila el software, lo que permite a los desarrolladores organizar y generar productos en entornos de prueba de compilación en los que Visual Studio no está instalado. En este tema se proporciona una breve información general sobre:

- Los elementos básicos de un archivo de proyecto de MSBuild.
- Cómo se utiliza MSBuild para generar proyectos.
- Las características avanzadas de MSBuild.
- Cómo Visual Studio utiliza MSBuild para generar proyectos.

1.1. Archivo de proyecto

MSBuild presenta un nuevo formato de archivo de proyecto basado en código XML que es sencillo de comprender, fácil de ampliar y totalmente compatible con Microsoft. El formato de archivo de proyecto de MSBuild permite a los desarrolladores describir de forma exhaustiva los elementos que deben generarse y el modo en que debe hacerse con distintas plataformas y configuraciones. Además, el formato de archivo de proyecto permite a los desarrolladores crear reglas de generación reutilizables que se pueden factorizar en archivos independientes para que las generaciones se ejecuten sistemáticamente en los distintos proyectos del producto. En las secciones siguientes se describen algunos de los elementos básicos del formato de archivo de proyecto de MSBuild.

Elementos

Los elementos representan entradas en el sistema de generación y se agrupan en colecciones de elementos basadas en los nombres de colección definidos por el usuario. Estas colecciones de elementos se pueden utilizar como parámetros para tareas, las cuales utilizan elementos individuales incluidos en la colección para llevar a cabo los pasos del proceso de generación.
Los elementos se declaran en el archivo de proyecto creando un elemento con el nombre de la colección de elementos como elemento secundario de un elemento ItemGroup. Por ejemplo, el código siguiente crea una colección de elementos con nombre Compile que incluye dos archivos.

```
<ItemGroup>
   <Compile Include = "file1.cs"/>
   <Compile Include = "file2.cs"/>
</ItemGroup>
```

A lo largo del archivo de proyecto se hace referencia a las colecciones de elementos mediante la sintaxis @(ItemCollectionName). Por ejemplo, en el ejemplo anterior se hace referencia a la colección de elementos con @(Compile).
Los elementos se pueden declarar utilizando comodines y pueden contener metadatos adicionales para escenarios de generación más avanzados. Para obtener más información sobre elementos.

Propiedades

Las propiedades representan pares clave/valor que se pueden utilizar para configurar generaciones. Los elementos y las propiedades difieren de la manera siguiente:

- Los elementos se almacenan en colecciones, mientras que las propiedades contienen un valor escalar único.
- Los elementos no se pueden quitar de las colecciones de elementos, mientras que se puede modificar el valor de las propiedades una vez definido.
- Los elementos pueden contener metadatos y utilizar la notación %(ItemMetadata), mientras que las propiedades no pueden.

Las propiedades se declaran creando un elemento con el nombre de la propiedad como elemento secundario de un elemento PropertyGroup. Por ejemplo, el código siguiente crea una propiedad denominada BuildDir con un valor de Build.

```
<PropertyGroup>
  <BuildDir>Build</BuildDir>
</PropertyGroup>
```

A lo largo del archivo de proyecto se hace referencia a las propiedades con la sintaxis $(PropertyName). Por ejemplo, para hacer referencia a la propiedad en el ejemplo anterior, se utiliza $(BuildDir). Para obtener más información sobre propiedades.

Tareas

Las tareas son unidades reutilizables de código ejecutable que se utilizan en proyectos de MSBuild para realizar operaciones de compilación. Por ejemplo, una tarea podría compilar archivos de entrada o ejecutar una herramienta externa. Una vez creadas, las tareas las pueden compartir y reutilizar desarrolladores diferentes en distintos proyectos.
La lógica de ejecución de una tarea se escribe en código administrado y se asigna a MSBuild con el elemento UsingTask. Puede escribir una tarea creando un tipo administrado que implemente la interfaz ITask. Para obtener más información sobre cómo escribir tareas.
MSBuild se distribuye con un gran número de tareas comunes, como Copy, que copia archivos, MakeDir, que crea directorios, y Csc, que compila archivos de código fuente de Visual C#. Para obtener una lista completa de tareas disponibles e información de uso.
Para ejecutar una tarea en un archivo de proyecto de MSBuild, se crea un elemento con el nombre de la tarea como elemento secundario de un elemento Target. Las tareas normalmente aceptan parámetros, que se pasan como atributos del elemento. Las propiedades y las colecciones de elementos de MSBuild también se pueden utilizar como parámetros. Por ejemplo, el código siguiente llama a la tarea MakeDir y le pasa el valor de la propiedad BuildDir declarado en el ejemplo anterior.

```
<Target Name="MakeBuildDirectory">
  <MakeDir
    Directories="$(BuildDir)" />
</Target>
```

Destinos

Los destinos agrupan tareas en un orden particular y exponen secciones del archivo de proyecto como puntos de entrada en el proceso de generación. Los destinos se agrupan a menudo en secciones lógicas para permitir la expansión y aumentar la legibilidad. Dividir los pasos de generación en muchos destinos permite llamar a una parte del proceso de generación desde otros destinos sin necesidad de copiar dicha sección de código en cada destino. Por ejemplo, si varios puntos de entrada en el proceso de generación necesitan que se generen referencias, se puede crear un destino que genere referencias y ejecute dicho destino desde cada punto de entrada necesario.
Los destinos se declaran en el archivo de proyecto con el elemento **Target**. Por ejemplo, en el ejemplo de código siguiente se crea un destino denominado Compile que, a continuación, llama a la tarea **Csc** con la colección de elementos declarada en el ejemplo anterior.

```
<Target Name="Compile">
  <Csc Sources="@(Compile)" />
</Target>
```

En escenarios más avanzados, los destinos pueden describir relaciones entre ellos y llevar a cabo análisis de dependencia, lo que permite omitir secciones completas en el proceso de generación si dicho destino está actualizado. Para obtener más información sobre destinos.

1.2. Generar con MSBuild

Para ejecutar MSBuild desde la línea de comandos, se pasa un archivo de proyecto a MSBuild.exe con las opciones de la línea de comandos adecuadas. Las opciones de la línea de comandos permiten establecer propiedades, ejecutar destinos específicos y especificar registradores. Por ejemplo, para generar el archivo MyProj.proj con la propiedad Configuration establecida en Debug, se usaría la sintaxis de línea de comandos siguiente:

```
MSBuild.exe MyProj.proj /property:Configuration=Debug
```

1.3. Conceptos avanzados

MSBuild se puede utilizar para realizar operaciones más avanzadas durante las compilaciones, como son el registro de errores, las advertencias, los mensajes para la consola y otros registradores, la realización de análisis de dependencia en los destinos y el procesamiento por lotes de tareas y destinos en metadatos de elementos. Para obtener más información sobre estos conceptos avanzados.

1.4. Integración de Visual Studio

Visual Studio utiliza el formato de archivo de proyecto de MSBuild para almacenar información de compilación sobre proyectos administrados. El valor del proyecto agregado y modificado mediante Visual Studio se refleja en el archivo .*proj que se genera para cada proyecto. Visual Studio utiliza una instancia hospedada de MSBuild para generar proyectos administrados; es decir, un proyecto administrado se puede generar en Visual Studio y desde la línea de comandos (incluso sin tener Visual Studio instalado), con resultados idénticos. Para obtener más información acerca de cómo Visual Studio utiliza MSBuild.

2. MSBuild Project File Schema Reference

For the latest documentation on Visual Studio 2017, see Visual Studio 2017 Documentation.
Provides a table of all the MSBuild XML Schema elements with their available attributes and child elements. MSBuild uses project files to instruct the build engine what to build and how to build it. MSBuild project files are XML files that adhere to the MSBuild XML schema. This section documents the XML schema definition (.xsd) file for MSBuild.

2.1. MSBuild XML Schema Elements

The following table lists all of the MSBuild XML schema elements along with their child elements and attributes.

Element	Child Elements	Attributes
Choose Element (MSBuild)	Otherwise When	--
Import Element (MSBuild)	--	Condition Project
ImportGroup Element	Import	Condition
Item Element (MSBuild)	ItemMetaData	Condition Exclude Include Remove
ItemDefinitionGroup Element (MSBuild)	Item	Condition
ItemGroup Element (MSBuild)	Item	Condition
ItemMetadata Element (MSBuild)	Item	Condition
OnError Element (MSBuild)	--	Condition ExecuteTargets
Otherwise Element (MSBuild)	Choose ItemGroup PropertyGroup	--
Output Element (MSBuild)	--	Condition ItemName PropertyName TaskParameter

Parameter Element	--	Output ParameterType Required
ParameterGroup Element	Parameter	--
Project Element (MSBuild)	Choose Import ItemGroup ProjectExtensions PropertyGroup Target UsingTask	DefaultTargets InitialTargets ToolsVersion TreatAsLocalProperty xmlns
ProjectExtensions Element (MSBuild)	--	--
Property Element (MSBuild)	--	Condition
PropertyGroup Element (MSBuild)	Property	Condition
Target Element (MSBuild)	OnError Task	AfterTargets BeforeTargets Condition DependsOnTargets Inputs KeepDuplicateOutputs Name Outputs Returns
Task Element (MSBuild)	Output	Condition ContinueOnError Parameter
TaskBody Element (MSBuild)	Data	Evaluate
UsingTask Element (MSBuild)	ParameterGroup	AssemblyFile

	TaskBody	AssemblyName Condition TaskFactory TaskName
When Element (MSBuild)	Choose ItemGroup PropertyGroup	Condition

3. MSBuild Task Reference

For the latest documentation on Visual Studio 2017.
Tasks provide the code that runs during the build process. The tasks in the following list are included with MSBuild. When Visual C++ is installed, additional tasks are available that are used to build Visual C++ projects. In addition to the parameters listed in the topics in this section, each task also has the following parameters:

Parameter	Description
Condition	Optional String parameter. A Boolean expression that the MSBuild engine uses to determine whether this task will be executed. For information about the conditions that are supported by MSBuild, see Conditions.
ContinueOnError	Optional parameter. Can contain one of the following values: - **WarnAndContinue** or **true**. When a task fails, subsequent tasks in the Target element and the build continue to execute, and all errors from the task are treated as warnings. - **ErrorAndContinue**. When a task fails, subsequent tasks in the Target element and the build continue to execute, and all errors from the task are treated as errors. - **ErrorAndStop** or **false** (default). When a task fails, the remaining tasks in the Target element and the build aren't executed, and the entire Target element and the build is considered to have failed.

4. MSBuild Conditions

MSBuild supports a specific set of conditions that can be applied wherever a Condition attribute is allowed. The following table explains those conditions.

Condition	Description
'stringA' == 'stringB'	Evaluates to true if stringA equals stringB. For example: Condition="'$(CONFIG)'=='DEBUG'" Single quotes are not required for simple alphanumeric strings or boolean values. However, single quotes are required for empty values.
'stringA' != 'stringB'	Evaluates to true if stringA is not equal to stringB. For example: Condition="'$(CONFIG)'!='DEBUG'" Single quotes are not required for simple alphanumeric strings or boolean values. However, single quotes are required for empty values.
<, >, <=, >=	Evaluates the numeric values of the operands. Returns true if the relational evaluation is true. Operands must evaluate to a decimal or hexadecimal number. Hexadecimal numbers must begin with "0x". **Note:** In XML, the characters < and > must be escaped. The symbol < is represented as <. The symbol > is represented as >.
Exists('stringA')	Evaluates to true if a file or folder with the name stringA exists. For example: Condition="!Exists('$(builtdir)')" Single quotes are not required for simple alphanumeric strings or boolean values. However, single quotes are required for empty values.
HasTrailingSlash('stringA')	Evaluates to true if the specified string contains either a trailing backward slash (\) or forward slash (/) character. For example: Condition="!HasTrailingSlash('$(OutputPath)')" Single quotes are not required for simple alphanumeric strings or boolean values. However, single quotes are required for empty values.
!	Evaluates to true if the operand evaluates to false.
And	Evaluates to true if both operands evaluate to true.

Or	Evaluates to true if at least one of the operands evaluates to true.
()	Grouping mechanism that evaluates to true if expressions contained inside evaluate to true.
if (%expression%), $else$, $endif$	Checks whether the specified %expression% matches the string value of the passed custom template parameter. If the if condition evaluates to true, then its statements are run; otherwise, the $else$ condition is checked. If the $else$ condition is true, then its statements are run; otherwise, the $endif$ condition ends expression evaluation.

5. MSBuild Conditional Constructs

MSBuild provides a mechanism for either/or processing with the Choose, When, and Otherwise elements.

5.1. Using the Choose Element

The Choose element contains a series of When elements with Condition attributes that are tested in order from top to bottom until one evaluates to true. If more than one When element evaluates to true, only the first one is used. An Otherwise element, if present, will be evaluated if no condition on a When element evaluates to true.
Choose elements can be used as child elements of Project, When and Otherwise elements. When and Otherwise elements can have ItemGroup, PropertyGroup, or Choose child elements.

Example

The following example uses the Choose and When elements for either/or processing. The properties and items for the project are set depending on the value of the Configuration property.

```
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003" >
  <PropertyGroup>
    <Configuration Condition=" '$(Configuration)' == '' ">Debug</Configuration>
    <OutputType>Exe</OutputType>
    <RootNamespace>ConsoleApplication1</RootNamespace>
    <AssemblyName>ConsoleApplication1</AssemblyName>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>
  <Choose>
    <When Condition=" '$(Configuration)'=='Debug' ">
      <PropertyGroup>
        <DebugSymbols>true</DebugSymbols>
        <DebugType>full</DebugType>
        <Optimize>false</Optimize>
        <OutputPath>.\bin\Debug\</OutputPath>
        <DefineConstants>DEBUG;TRACE</DefineConstants>
      </PropertyGroup>
      <ItemGroup>
```

```
        <Compile Include="UnitTesting\*.cs" />
        <Reference Include="NUnit.dll" />
      </ItemGroup>
    </When>
    <When Condition=" '$(Configuration)'=='retail' ">
      <PropertyGroup>
        <DebugSymbols>false</DebugSymbols>
        <Optimize>true</Optimize>
        <OutputPath>.\bin\Release\</OutputPath>
        <DefineConstants>TRACE</DefineConstants>
      </PropertyGroup>
    </When>
  </Choose>
  <!-- Rest of Project -->
</Project>
```

6. MSBuild Reserved and Well-Known Properties

MSBuild provides a set of predefined properties that store information about the project file and the MSBuild binaries. These properties are evaluated in the same manner as other MSBuild properties. For example, to use the MSBuildProjectFile property, you type $(MSBuildProjectFile).
MSBuild uses the values in the following table to predefine reserved and well-known properties. Reserved properties cannot be overridden, but well-known properties can be overridden by using identically named environment properties, global properties, or properties that are declared in the project file.

6.1. Reserved and Well-Known Properties

The following table describes the MSBuild predefined properties.

Property	Description	Reserved or Well-Known
MSBuildBinPath	The absolute path of the folder where the MSBuild binaries that are currently being used are located (for example, C:\Windows\Microsoft.Net\Framework\versionNumber). This property is useful if you have to refer to files in the MSBuild directory. Do not include the final backslash on this property.	Reserved
MSBuildExtensionsPath	Introduced in the .NET Framework 4: there is no difference between the default values of MSBuildExtensionsPath and MSBuildExtensionsPath32. You can set the environment variable MSBUILDLEGACYEXTENSIONSPATH to a non-null value to enable the behavior of the default value of MSBuildExtensionsPath in earlier versions.	Well-Known

	In the .NET Framework 3.5 and earlier, the default value of MSBuildExtensionsPath points to the path of the MSBuild subfolder under the \Program Files\ or \Program Files (x86) folder, depending on the bitness of the current process. For example, for a 32-bit process on a 64-bit machine, this property points to the \Program Files (x86) folder. For a 64-bit process on a 64-bit machine, this property points to the \Program Files folder. Do not include the final backslash on this property. This location is a useful place to put custom target files. For example, your target files could be installed at \Program Files\MSBuild\MyFiles\Northwind.targets and then imported in project files by using this XML code: <Import Project="$(MSBuildExtensionsPath)\MyFiles\Northwind.targets"/>	
MSBuildExtensionsPath32	The path of the MSBuild subfolder under the \Program Files or \Program Files (x86) folder. This path always points to the 32-bit \Program Files folder on a 32-bit machine and \Program Files (x86) on a 64-bit machine. See also MSBuildExtensionsPath and MSBuildExtensionsPath64. Do not include the final backslash on this property.	Well-Known
MSBuildExtensionsPath64	The path of the MSBuild subfolder under the \Program Files folder. For a 64-bit machine, this path always points to the \Program Files folder. For a 32-bit machine, this path is blank. See also MSBuildExtensionsPath and MSBuildExtensionsPath32. Do not include the final backslash on this property.	Well-Known
MSBuildLastTaskResult	true if the previous task completed without any errors (even if there were warnings), or false if the previous task had errors. Typically, when an error occurs in a task, the error is the last thing that happens in that project. Therefore, the value of this property is never false, except in these scenarios: - When the ContinueOnError attribute of the Task Element (MSBuild) is set to WarnAndContinue (or true) or ErrorAndContinue. - When the Target has an OnError Element (MSBuild) as a child element.	Reserved
MSBuildNodeCount	The maximum number of concurrent processes that are used when building. This is the value that you specified for	Reserved

	/maxcpucount on the command line. If you specified /maxcpucount without specifying a value, then MSBuildNodeCount specifies the number of processors in the computer.	
MSBuildProgramFiles32	The location of the 32-bit program folder; for example, C:\Program Files (x86). Do not include the final backslash on this property.	Reserved
MSBuildProjectDefaultTargets	The complete list of targets that are specified in the DefaultTargets attribute of the Project element. For example, the following Project element would have an MSBuildDefaultTargets property value of A;B;C: <Project DefaultTargets="A;B;C" >	Reserved
MSBuildProjectDirectory	The absolute path of the directory where the project file is located, for example C:\MyCompany\MyProduct. Do not include the final backslash on this property.	Reserved
MSBuildProjectDirectoryNoRoot	The value of the MSBuildProjectDirectory property, excluding the root drive. Do not include the final backslash on this property.	Reserved
MSBuildProjectExtension	The file name extension of the project file, including the period; for example, .proj.	Reserved
MSBuildProjectFile	The complete file name of the project file, including the file name extension; for example, MyApp.proj.	Reserved
MSBuildProjectFullPath	The absolute path and complete file name of the project file, including the file name extension; for example, C:\MyCompany\MyProduct\MyApp.proj.	Reserved
MSBuildProjectName	The file name of the project file without the file name extension; for example, MyApp.	Reserved
MSBuildStartupDirectory	The absolute path of the folder where MSBuild is called. By using this property, you can build everything below a specific point in a project tree without creating dirs.proj files in every directory. Instead, you have just one project—for example, c:\traversal.proj, as shown here: <Project ...> <ItemGroup> <ProjectFiles Include="$(MSBuildStartupDirectory) ***.csproj"/> </ItemGroup> <Target Name="build"> <MSBuild Projects="@(ProjectFiles)"/> </Target> </Project> To build at any point in the tree, type: msbuild c:\traversal.proj Do not include the final backslash on this property.	Reserved

MSBuildThisFile	The file name and file extension portion of MSBuildThisFileFullPath.	Reserved
MSBuildThisFileDirectory	The directory portion of MSBuildThisFileFullPath. Include the final backslash in the path.	Reserved
MSBuildThisFileDirectoryNoRoot	The directory portion of MSBuildThisFileFullPath, excluding the root drive. Include the final backslash in the path.	Reserved
MSBuildThisFileExtension	The file name extension portion of MSBuildThisFileFullPath.	Reserved
MSBuildThisFileFullPath	The absolute path of the project or targets file that contains the target that is running. Tip: You can specify a relative path in a targets file that's relative to the targets file and not relative to the original project file.	Reserved
MSBuildThisFileName	The file name portion of MSBuildThisFileFullPath, without the file name extension.	Reserved
MSBuildToolsPath	The installation path of the MSBuild version that's associated with the value of MSBuildToolsVersion. Do not include the final backslash in the path. This property cannot be overridden.	Reserved
MSBuildToolsVersion	The version of the MSBuild Toolset that is used to build the project. Note: An MSBuild Toolset consists of tasks, targets, and tools that are used to build an application. The tools include compilers such as csc.exe and vbc.exe.	Reserved

7. Common MSBuild Project Properties

The following table lists frequently used properties that are defined in the Visual Studio project files or included in .targets files that MSBuild provides.

Project files in Visual Studio (.csproj,.vbproj, vcxproj, and others) contain MSBuild XML code that runs when you build a project by using the IDE. Projects typically import one or more .targets files to define their build process.

7.1. List of Common Properties and Parameters

Property or Parameter Name	Description
AdditionalLibPaths	Specifies additional folders in which compilers should look for reference assemblies.
AddModules	Causes the compiler to make all type information from the specified files available to the project you are compiling. This property is equivalent to the /addModules compiler switch.
ALToolPath	The path where AL.exe can be found. This property overrides the current version of AL.exe to enable use of a different version.
ApplicationIcon	The .ico icon file to pass to the compiler for embedding as a Win32 icon. The property is equivalent to the /win32icon compiler switch.
ApplicationManifest	Specifies the path of the file that is used to generate external User Account Control (UAC) manifest information. Applies only to Visual Studio projects targeting Windows Vista. In most cases, the manifest is embedded. However, if you use Registration Free COM or ClickOnce deployment, then the manifest can be an external file that is installed together with your application assemblies. For more information, see the NoWin32Manifest property in this topic.
AssemblyOriginatorKeyFile	Specifies the file that's used to sign the assembly (.snk or .pfx) and that's passed to the ResolveKeySource Task to generate the actual key that's used to sign the assembly.
AssemblySearchPaths	A list of locations to search during build-time reference assembly resolution. The order in which paths appear in this list is meaningful because paths listed earlier takes precedence over later entries.
AssemblyName	The name of the final output assembly after the project is built.
BaseAddress	Specifies the base address of the main output assembly. This property is equivalent to the /baseaddress compiler switch.
BaseOutputPath	Specifies the base path for the output file. If it is set, MSBuild will use OutputPath = $(BaseOutputPath)\$(Configuration)\. Example syntax: <BaseOutputPath>c:\xyz\bin\</BaseOutputPath>
BaseIntermediateOutputPath	The top-level folder where all configuration-specific intermediate output folders are created. The default value is obj\. The following code is an example: <BaseIntermediateOutputPath>c:\xyz\obj\</BaseIntermediateOutputPath>
BuildInParallel	A boolean value that indicates whether project references are built or cleaned in parallel when Multi-Proc MSBuild is used. The default value is true, which means that projects will be built in parallel if the system has multiple cores or processors.
BuildProjectReferences	A boolean value that indicates whether project references are built by MSBuild. Set false if you are building your project in the Visual Studio integrated development environment (IDE), true if otherwise.
CleanFile	The name of the file that will be used as the "clean cache." The clean cache is a list of generated files to be deleted during the cleaning operation. The file is put in the intermediate output path by the build process. This property specifies only file names that do not have path information.

CodePage	Specifies the code page to use for all source-code files in the compilation. This property is equivalent to the /codepage compiler switch.
CompilerResponseFile	An optional response file that can be passed to the compiler tasks.
Configuration	The configuration that you are building, either "Debug" or "Release."
CscToolPath	The path of csc.exe, the Visual C# compiler.
CustomBeforeMicrosoftCommonTargets	The name of a project file or targets file that is to be imported automatically before the common targets import.
DebugSymbols	A boolean value that indicates whether symbols are generated by the build. Setting **/p:DebugSymbols=false** on the command line disables generation of program database (.pdb) symbol files.
DefineConstants	Defines conditional compiler constants. Symbol/value pairs are separated by semicolons and are specified by using the following syntax: symbol1 = value1; symbol2 = value2 The property is equivalent to the /define compiler switch.
DefineDebug	A boolean value that indicates whether you want the DEBUG constant defined.
DefineTrace	A boolean value that indicates whether you want the TRACE constant defined.
DebugType	Defines the level of debug information that you want generated. Valid values are "full," "pdbonly," and "none."
DelaySign	A boolean value that indicates whether you want to delay-sign the assembly rather than full-sign it.
DisabledWarnings	Suppresses the specified warnings. Only the numeric part of the warning identifier must be specified. Multiple warnings are separated by semicolons. This parameter corresponds to the /nowarn switch of the vbc.exe compiler.
DisableFastUpToDateCheck	A boolean value that applies to Visual Studio only. The Visual Studio build manager uses a process called FastUpToDateCheck to determine whether a project must be rebuilt to be up to date. This process is faster than using MSBuild to determine this. Setting the DisableFastUpToDateCheck property to true lets you bypass the Visual Studio build manager and force it to use MSBuild to determine whether the project is up to date.
DocumentationFile	The name of the file that is generated as the XML documentation file. This name includes only the file name and has no path information.
ErrorReport	Specifies how the compiler task should report internal compiler errors. Valid values are "prompt," "send," or "none." This property is equivalent to the /errorreport compiler switch.
ExcludeDeploymentUrl	The GenerateDeploymentManifest Task adds a deploymentProvider tag to the deployment manifest if the project file includes any of the following elements: - UpdateUrl - InstallUrl - PublishUrl Using ExcludeDeploymentUrl, however, you can prevent the deploymentProvider tag from being added to the deployment manifest even if any of the above URLs are specified. To do this, add the following property to your project file:

	<ExcludeDeploymentUrl>true</ExcludeDeploymentUrl> **Note:** ExcludeDeploymentUrl is not exposed in the Visual Studio IDE and can be set only by manually editing the project file. Setting this property does not affect publishing within Visual Studio; that is, the deploymentProvider tag will still be added to the URL specified by PublishUrl.
FileAlignment	Specifies, in bytes, where to align the sections of the output file. Valid values are 512, 1024, 2048, 4096, 8192. This property is equivalent to the /filealignment compiler switch.
FrameworkPathOverride	Specifies the location of mscorlib.dll and microsoft.visualbasic.dll. This parameter is equivalent to the /sdkpath switch of the vbc.exe compiler.
GenerateDocumentation	A boolean parameter that indicates whether documentation is generated by the build. If true, the build generates documentation information and puts it in an .xml file together with the name of the executable file or library that the build task created.
IntermediateOutputPath	The full intermediate output path as derived from BaseIntermediateOutputPath, if no path is specified. For example, \obj\debug\. If this property is overridden, then setting BaseIntermediateOutputPath has no effect.
KeyContainerName	The name of the strong-name key container.
KeyOriginatorFile	The name of the strong-name key file.
NoWin32Manifest	Determines whether the compiler generates the default Win32 manifest into the output assembly. The default value of false means that the default Win32 manifest is generated for all applications. This property is equivalent to the /nowin32manifest compiler switch of vbc.exe.
ModuleAssemblyName	The name of the assembly that the compiled module is to be incorporated into. The property is equivalent to the /moduleassemblyname compiler switch.
NoLogo	A boolean value that indicates whether you want compiler logo to be turned off. This property is equivalent to the /nologo compiler switch.
NoStdLib	A boolean value that indicates whether to avoid referencing the standard library (mscorlib.dll). The default value is false.
NoVBRuntimeReference	A boolean value that indicates whether the Visual Basic runtime (Microsoft.VisualBasic.dll) should be included as a reference in the project.
NoWin32Manifest	A boolean value that indicates whether User Account Control (UAC) manifest information will be embedded in the application's executable. Applies only to Visual Studio projects targeting Windows Vista. In projects deployed using ClickOnce and Registration-Free COM, this element is ignored. False (the default value) specifies that User Account Control (UAC) manifest information be embedded in the application's executable. True specifies that UAC manifest information not be embedded. This property applies only to Visual Studio projects targeting Windows Vista. In projects deployed using ClickOnce and Registration-Free COM, this property is ignored. You should add NoWin32Manifest only if you do not want Visual Studio to embed any manifest information in the application's executable; this process is called

	virtualization. To use virtualization, set <ApplicationManifest> in conjunction with <NoWin32Manifest> as follows: - For Visual Basic projects, remove the <ApplicationManifest> node. (In Visual Basic projects, <NoWin32Manifest> is ignored when an <ApplicationManifest> node exists.) - For Visual C# projects, set <ApplicationManifest> to False and <NoWin32Manifest> to True. (In Visual C# projects, <ApplicationManifest> overrides <NoWin32Manifest>.)
Optimize	A boolean value that when set to true, enables compiler optimizations. This property is equivalent to the /optimize compiler switch.
OptionCompare	Specifies how string comparisons are made. Valid values are "binary" or "text." This property is equivalent to the /optioncompare compiler switch of vbc.exe.
OptionExplicit	A boolean value that when set to true, requires explicit declaration of variables in the source code. This property is equivalent to the /optionexplicit compiler switch.
OptionInfer	A boolean value that when set to true, enables type inference of variables. This property is equivalent to the /optioninfer compiler switch.
OptionStrict	A boolean value that when set to true, causes the build task to enforce strict type semantics to restrict implicit type conversions. This property is equivalent to the /optionstrict switch of the vbc.exe compiler.
OutputPath	Specifies the path to the output directory, relative to the project directory, for example, "bin\Debug".
OutputType	Specifies the file format of the output file. This parameter can have one of the following values: - Library. Creates a code library. (Default value.) - Exe. Creates a console application. - Module. Creates a module. - Winexe. Creates a Windows-based program. This property is equivalent to the /target switch of the vbc.exe compiler.
OverwriteReadOnlyFiles	A boolean value that indicates whether you want to enable the build to overwrite read-only files or trigger an error.
PdbFile	The file name of the .pdb file that you are emitting. This property is equivalent to the /pdb switch of the csc.exe compiler.
Platform	The operating system you are building for. Valid values are "Any CPU", "x86", and "x64".
RemoveIntegerChecks	A boolean value that indicates whether to disable integer overflow error checks. The default value is false. This property is equivalent to the /removeintchecks switch of the vbc.exe compiler.
SGenUseProxyTypes	A boolean value that indicates whether proxy types should be generated by SGen.exe. The SGen target uses this property to set the UseProxyTypes flag. This property defaults to true, and there is no UI to change this. To generate the serialization assembly for non-webservice types, add this property to the project file and set it to false before importing the Microsoft.Common.Targets or the C#/VB.targets.

SGenToolPath	An optional tool path that indicates where to obtain SGen.exe when the current version of SGen.exe is overridden.
StartupObject	Specifies the class or module that contains the Main method or Sub Main procedure. This property is equivalent to the /main compiler switch.
ProcessorArchitecture	The processor architecture that is used when assembly references are resolved. Valid values are "msil," "x86," "amd64," or "ia64."
RootNamespace	The root namespace to use when you name an embedded resource. This namespace is part of the embedded resource manifest name.
Satellite_AlgorithmId	The ID of the AL.exe hashing algorithm to use when satellite assemblies are created.
Satellite_BaseAddress	The base address to use when culture-specific satellite assemblies are built by using the CreateSatelliteAssemblies target.
Satellite_CompanyName	The company name to pass into AL.exe during satellite assembly generation.
Satellite_Configuration	The configuration name to pass into AL.exe during satellite assembly generation.
Satellite_Description	The description text to pass into AL.exe during satellite assembly generation.
Satellite_EvidenceFile	Embeds the specified file in the satellite assembly that has the resource name "Security.Evidence."
Satellite_FileVersion	Specifies a string for the File Version field in the satellite assembly.
Satellite_Flags	Specifies a value for the Flags field in the satellite assembly.
Satellite_GenerateFullPaths	Causes the build task to use absolute paths for any files reported in an error message.
Satellite_LinkResource	Links the specified resource files to a satellite assembly.
Satellite_MainEntryPoint	Specifies the fully-qualified name (that is, class.method) of the method to use as an entry point when a module is converted to an executable file during satellite assembly generation.
Satellite_ProductName	Specifies a string for the Product field in the satellite assembly.
Satellite_ProductVersion	Specifies a string for the ProductVersion field in the satellite assembly.
Satellite_TargetType	Specifies the file format of the satellite assembly output file as "library," "exe," or "win." The default value is "library."
Satellite_Title	Specifies a string for the Title field in the satellite assembly.
Satellite_Trademark	Specifies a string for the Trademark field in the satellite assembly.
Satellite_Version	Specifies the version information for the satellite assembly.
Satellite_Win32Icon	Inserts an .ico icon file in the satellite assembly.
Satellite_Win32Resource	Inserts a Win32 resource (.res file) into the satellite assembly.
SubsystemVersion	Specifies the minimum version of the subsystem that the generated executable file can use. This property is equivalent to the /subsystemversion compiler switch. For information about the default value of this property, see /subsystemversion (Visual Basic) or /subsystemversion (C# Compiler Options).
TargetCompactFramework	The version of the .NET Compact Framework that is required to run the application that you are building. Specifying this lets you reference certain framework assemblies that you may not be able to reference otherwise.

TargetFrameworkVersion	The version of the .NET Framework that is required to run the application that you are building. Specifying this lets you reference certain framework assemblies that you may not be able to reference otherwise.
TreatWarningsAsErrors	A boolean parameter that, if true, causes all warnings to be treated as errors. This parameter is equivalent to the /nowarn compiler switch.
UseHostCompilerIfAvailable	A boolean parameter that, if true, causes the build task to use the in-process compiler object, if it is available. This parameter is used only by Visual Studio.
Utf8Output	A boolean parameter that, if true, logs compiler output by using UTF-8 encoding. This parameter is equivalent to the /utf8Output compiler switch.
VbcToolPath	An optional path that indicates another location for vbc.exe when the current version of vbc.exe is overridden.
VbcVerbosity	Specifies the verbosity of the Visual Basic compiler's output. Valid values are "Quiet," "Normal" (the default value), or "Verbose."
VisualStudioVersion	Specifies the version of Visual Studio under which this project should be considered to be running. If this property isn't specified, MSBuild sets it to a reasonable default value. This property is used in several project types to specify the set of targets that are used for the build. If ToolsVersion is set to 4.0 or higher for a project, VisualStudioVersion is used to specify which sub-toolset to use.
WarningsAsErrors	Specifies a list of warnings to treat as errors. This parameter is equivalent to the /warnaserror compiler switch.
WarningsNotAsErrors	Specifies a list of warnings that are not treated as errors. This parameter is equivalent to the /warnaserror compiler switch.
Win32Manifest	The name of the manifest file that should be embedded in the final assembly. This parameter is equivalent to the /win32Manifest compiler switch.
Win32Resource	The file name of the Win32 resource to be embedded in the final assembly. This parameter is equivalent to the /win32resource compiler switch.

8. Common MSBuild Project Items

In MSBuild, an item is a named reference to one or more files. Items contain metadata such as file names, paths, and version numbers. All project types in Visual Studio have several items in common. These items are defined in the file microsoft.build.commontypes.xsd.

8.1. Common Items

The following is a list of all the common project items.

Reference

Represents an assembly (managed) reference in the project.

Item Name	Description
HintPath	Optional string. Relative or absolute path of the assembly.
Name	Optional string. The display name of the assembly, for example, "System.Windows.Forms."
FusionName	Optional string. Specifies the simple or strong fusion name for the item. When this attribute is present, it can save time because the assembly file does not have to be opened to obtain the fusion name.
SpecificVersion	Optional boolean. Specifies whether only the version in the fusion name should be referenced.
Aliases	Optional string. Any aliases for the reference.
Private	Optional boolean. Specifies whether the reference should be copied to the output folder. This attribute matches the **Copy Local** property of the reference that's in the Visual Studio IDE.

COMReference

Represents a COM (unmanaged) component reference in the project.

Item Name	Description
Name	Optional string. The display name of the component.
Guid	Optional string. A GUID for the component, in the form {12345678-1234-1234-1234-1234567891234}.
VersionMajor	Optional string. The major part of the version number of the component. For example, "5" if the full version number is "5.46."
VersionMinor	Optional string. The minor part of the version number of the component. For example, "46" if the full version number is "5.46."
LCID	Optional string. The LocaleID for the component.
WrapperTool	Optional string. The name of the wrapper tool that is used on the component, for example, "tlbimp."
Isolated	Optional boolean. Specifies whether the component is a reg-free component.

COMFileReference

Represents a list of type libraries that feed into the ResolvedComreference target.

Item Name	Description
WrapperTool	Optional string. The name of the wrapper tool that is used on the component, for example, "tlbimp."

NativeReference

Represents a native manifest file or a reference to such a file.

Item Name	Description
Name	Required string. The base name of the manifest file.

HintPath	Required string. The relative path of the manifest file.

ProjectReference

Represents a reference to another project.

Item Name	Description
Name	Optional string. The display name of the reference.
Project	Optional string. A GUID for the reference, in the form {12345678-1234-1234-1234-1234567891234}.
Package	Optional string. The path of the project file that is being referenced.

Compile

Represents the source files for the compiler.

Item Name	Description
DependentUpon	Optional string. Specifies the file this file depends on to compile correctly.
AutoGen	Optional boolean. Indicates whether the file was generated for the project by the Visual Studio integrated development environment (IDE).
Link	Optional string. The notational path to be displayed when the file is physically located outside the influence of the project file.
Visible	Optional boolean. Indicates whether to display the file in **Solution Explorer** in Visual Studio.
CopyToOutputDirectory	Optional string. Determines whether to copy the file to the output directory. Values are: 1. Never 2. Always 3. PreserveNewest

EmbeddedResource

Represents resources to be embedded in the generated assembly.

Item Name	Description
DependentUpon	Optional string. Specifies the file this file depends on to compile correctly
Generator	Required string. The name of any file generator that is run on this item.
LastGenOutput	Required string. The name of the file that was created by any file generator that ran on this item.
CustomToolNamespace	Required string. The namespace in which any file generator that runs on this item should create code.
Link	Optional string. The notational path is displayed if the file is physically located outside the influence of the project.
Visible	Optional boolean. Indicates whether to display the file in **Solution Explorer** in Visual Studio.

CopyToOutputDirectory	Optional string. Determines whether to copy the file to the output directory. Values are: 1. Never 2. Always 3. PreserveNewest
LogicalName	Required string. The logical name of the embedded resource.

Content

Represents files that are not compiled into the project, but may be embedded or published together with it.

Item Name	Description
DependentUpon	Optional string. Specifies the file this file depends on to compile correctly.
Generator	Required string. The name of any file generator that runs on this item.
LastGenOutput	Required string. The name of the file that was created by any file generator that was run on this item.
CustomToolNamespace	Required string. The namespace in which any file generator that runs on this item should create code.
Link	Optional boolean. Indicates whether to display the file in **Solution Explorer** in Visual Studio.
PublishState	Required string. The publish state of the content, either: - Default - Included - Excluded - DataFile - Prerequisite
IsAssembly	Optional boolean. Specifies whether the file is an assembly.
Visible	Optional boolean. Indicates whether to display the file in **Solution Explorer** in Visual Studio.
CopyToOutputDirectory	Optional string. Determines whether to copy the file to the output directory. Values are: 1. Never 2. Always 3. PreserveNewest

None

Represents files that should have no role in the build process.

Item Name	Description
DependentUpon	Optional string. Specifies the file this file depends on to compile correctly.
Generator	Required string. The name of any file generator that is run on this item.

LastGenOutput	Required string. The name of the file that was created by any file generator that ran on this item.
CustomToolNamespace	Required string. The namespace in which any file generator that runs on this item should create code.
Link	Optional string. The notational path to be displayed if the file is physically located outside the influence of the project.
Visible	Optional boolean. Indicates whether to display the file in **Solution Explorer** in Visual Studio.
CopyToOutputDirectory	Optional string. Determines whether to copy the file to the output directory. Values are: 1. Never 2. Always 3. PreserveNewest

BaseApplicationManifest

Represents the base application manifest for the build, and contains ClickOnce deployment security information.

CodeAnalysisImport

Represents the FxCop project to import.

Import

Represents assemblies whose namespaces should be imported by the Visual Basic compiler.

9. MSBuild Command-Line Reference

When you use MSBuild.exe to build a project or solution file, you can include several switches to specify various aspects of the process.

9.1. Syntax

```
MSBuild.exe [Switches] [ProjectFile]
```

9.2. Arguments

Argument	Description
ProjectFile	Builds the targets in the project file that you specify. If you don't specify a project file, MSBuild searches the current working directory for a file name extension that ends in "proj" and uses that file. You can also specify a Visual Studio solution file for this argument.

9.3. Switches

Switch	Short form	Description
/help	/? or /h	Display usage information. The following command is an example: msbuild.exe /?
/detailedsummary	/ds	Show detailed information at the end of the build log about the configurations that were built and how they were scheduled to nodes.
/ignoreprojectextensions:extensions	/ignore:extensions	Ignore the specified extensions when determining which project file to build. Use a semicolon or a comma to separate multiple extensions, as the following example shows: /ignoreprojectextensions:.vcproj,.sln
/maxcpucount[:number]	/m[:number]	Specifies the maximum number of concurrent processes to use when building. If you don't include this switch, the default value is 1. If you include this switch without specifying a value, MSBuild will use up to the number of processors in the computer. The following example instructs MSBuild to build using three MSBuild processes, which allows three projects to build at the same time: msbuild myproject.proj /maxcpucount:3
/noautoresponse	/noautorsp	Don't include any MSBuild.rsp files automatically.
/nodeReuse:value	/nr:value	Enable or disable the re-use of MSBuild nodes. You can specify the following values: - True. Nodes remain after the build finishes so that subsequent builds can use them (default). - False. Nodes don't remain after the build completes. A node corresponds to a project that's executing. If you include the /maxcpucount switch, multiple nodes can execute concurrently.
/nologo		Don't display the startup banner or the copyright message.
/preprocess[:filepath]	/pp[:filepath]	Create a single, aggregated project file by inlining all the files that would be imported during a build, with their boundaries marked. You can use this switch to more easily determine which files are being imported, from where the files are being imported, and which files contribute to the build. When you use this switch, the project isn't built. If you specify a filepath, the aggregated project file is output to the file. Otherwise, the output appears in the console window.
/property:name=value	/p:name=value	Set or override the specified project-level properties, where name is the property name and value is the property value. Specify each property separately, or use a semicolon or comma to separate multiple properties, as the following example shows:

		/property:WarningLevel=2;OutDir=bin\Debug
/target:targets	/t:targets	Build the specified targets in the project. Specify each target separately, or use a semicolon or comma to separate multiple targets, as the following example shows: /target:Resources;Compile If you specify any targets by using this switch, they are run instead of any targets in the DefaultTargets attribute in the project file. A target is a group of tasks.
/toolsversion:version	/tv:version	Specifies the version of the Toolset to use to build the project, as the following example shows: /toolsversion:3.5 By using this switch, you can build a project and specify a version that differs from the version that's specified in the Project Element (MSBuild). For MSBuild 4.5, you can specify the following values for version: 2.0, 3.5, and 4.0. If you specify 4.0, the VisualStudioVersion build property specifies which sub-toolset to use. For more information, see the Sub-toolsets section of Toolset (ToolsVersion). A Toolset consists of tasks, targets, and tools that are used to build an application. The tools include compilers such as csc.exe and vbc.exe. For more information about Toolsets, see Toolset (ToolsVersion), Standard and Custom Toolset Configurations, and Multitargeting. **Note:** The toolset version isn't the same as the target framework, which is the version of the .NET Framework on which a project is built to run.
/validate:[schema]	/val[schema]	Validate the project file and, if validation succeeds, build the project. If you don't specify schema, the project is validated against the default schema. If you specify schema, the project is validated against the schema that you specify. The following setting is an example: /validate:MyExtendedBuildSchema.xsd
/verbosity:level	/v:level	Specifies the amount of information to display in the build log. Each logger displays events based on the verbosity level that you set for that logger. You can specify the following verbosity levels: q[uiet], m[inimal], n[ormal], d[etailed], and diag[nostic]. The following setting is an example: /verbosity:quiet
/version	/ver	Display version information only. The project isn't built.
@file		Insert command-line switches from a text file. If you have multiple files, you specify them separately.

9.4. Switches for Loggers

Switch	Short form	Description
/consoleloggerparameters: parameters	/clp:parameters	Pass the parameters that you specify to the console logger, which displays build information in the console window. You can specify the following parameters: - **PerformanceSummary**. Show the time that's spent in tasks, targets, and projects. - **Summary**. Show the error and warning summary at the end. - **NoSummary**. Don't show the error and warning summary at the end. - **ErrorsOnly**. Show only errors. - **WarningsOnly**. Show only warnings. - **NoItemAndPropertyList**. Don't show the list of items and properties that would appear at the start of each project build if the verbosity level is set to diagnostic. - **ShowCommandLine**. Show TaskCommandLineEvent messages. - **ShowTimestamp**. Show the timestamp as a prefix to any message. - **ShowEventId**. Show the event ID for each started event, finished event, and message. - **ForceNoAlign**. Don't align the text to the size of the console buffer. - **DisableConsoleColor**. Use the default console colors for all logging messages. - **DisableMPLogging**. Disable the multiprocessor logging style of output when running in non-multiprocessor mode. - **EnableMPLogging**. Enable the multiprocessor logging style even when running in non-multiprocessor mode. This logging style is on by default. - **Verbosity**. Override the **/verbosity** setting for this logger. Use a semicolon or comma to separate multiple parameters, as the following example shows: /consoleloggerparameters:PerformanceSummary;NoSummary /verbosity:minimal
/distributedFileLogger	/dfl	Log the build output of each MSBuild node to its own file. The initial location for these files is the current directory. By default, the files are named "MSBuildNodeId.log". You can use the **/fileLoggerParameters** switch to specify the location of the files and other parameters for the fileLogger. If you name a log file by using the **/fileLoggerParameters** switch, the distributed logger will use that name as a template and

		append the node ID to that name when creating a log file for each node.
/distributedlogger: central logger* forwarding logger	/dl:central logger*forwarding logger	Log events from MSBuild, attaching a different logger instance to each node. To specify multiple loggers, specify each logger separately. You use the logger syntax to specify a logger. For the logger syntax, see the **/logger** switch below. The following examples show how to use this switch: /dl:XMLLogger,MyLogger,Version=1.0.2,Culture=neutral /dl:MyLogger,C:\My.dll*ForwardingLogger,C:\Logger.dll
/fileLogger [number]	/fl[number]	Log the build output to a single file in the current directory. If you don't specify number, the output file is named msbuild.log. If you specify number, the output file is named msbuildn.log, where n is number. Number can be a digit from 1 to 9. You can use the **/fileLoggerParameters** switch to specify the location of the file and other parameters for the fileLogger.
/fileloggerparameters:[number] parameters	/flp:[number] parameters	Specifies any extra parameters for the file logger and the distributed file logger. The presence of this switch implies that the corresponding **/filelogger[**number**]** switch is present. Number can be a digit from 1 to 9. You can use all parameters that are listed for **/consoleloggerparameters**. You can also use one or more of the following parameters: - **LogFile**. The path to the log file into which the build log is written. The distributed file logger prefixes this path to the names of its log files. - **Append**. Determines whether the build log is appended to the log file or overwrites it. When you set the switch, the build log is appended to the log file. When the switch is not present, the contents of an existing log file are overwritten. If you include the append switch, no matter whether it is set to true or false, the log is appended. If you do not include the append switch, the log is overwritten. In this case the file is overwritten: msbuild myfile.proj /l:FileLogger,Microsoft.Build.Engine;logfile=MyLog.log In this case the file is appended: msbuild myfile.proj /l:FileLogger,Microsoft.Build.Engine;logfile=MyLog.log;append=true In this case the file is appended: msbuild myfile.proj /l:FileLogger,Microsoft.Build.Engine;logfile=MyLog.log;append=false

		- **Encoding**. Specifies the encoding for the file (for example, UTF-8, Unicode, or ASCII). The following example generates separate log files for warnings and errors: /flp1:logfile=errors.txt;errorsonly /flp2:logfile=warnings.txt;warningsonly The following examples show other possibilities: /fileLoggerParameters:LogFile=MyLog.log;Append;Verbosity=diagnostic;Encoding=UTF-8 /flp:Summary;Verbosity=minimal;LogFile=msbuild.sum /flp1:warningsonly;logfile=msbuild.wrn /flp2:errorsonly;logfile=msbuild.err
/logger: logger	/l:logger	Specifies the logger to use to log events from MSBuild. To specify multiple loggers, specify each logger separately. Use The following syntax for logger: [``LoggerClass``,]``LoggerAssembly``[;``LoggerParameters``] Use the following syntax for LoggerClass: [``PartialOrFullNamespace``.]``LoggerClassName  You don't have to specify the logger class if the assembly contains exactly one logger.  Use the following syntax for LoggerAssembly: {``AssemblyName``[,``StrongName``] &#124; AssemblyFile``} Logger parameters are optional and are passed to the logger exactly as you enter them. The following examples use the **/logger** switch. /logger:XMLLogger,MyLogger,Version=1.0.2,Culture=neutral /logger:XMLLogger,C:\Loggers\MyLogger.dll;OutputAsHTML
/noconsolelogger	/noconlog	Disable the default console logger, and don't log events to the console.

Example

The following example builds the rebuild target of the MyProject.proj project.

```
MSBuild.exe MyProject.proj /t:rebuild
```

Example

You can use MSBuild.exe to perform more complex builds. For example, you can use it to build specific targets of specific projects in a solution. The following example rebuilds the project NotInSolutionFolder and cleans the project InSolutionFolder, which is in the NewFolder solution folder.

```
msbuild SlnFolders.sln /t:NotInSolutionfolder:Rebuild;NewFolder\InSolutionFolder:Clean
```

10. MSBuild .Targets Files

MSBuild includes several .targets files that contain items, properties, targets, and tasks for common scenarios. These files are automatically imported into most Visual Studio project files to simplify maintenance and readability.

Projects typically import one or more .targets files to define their build process. For example a Visual C# project created by Visual Studio will import Microsoft.CSharp.targets which imports Microsoft.Common.targets. The Visual C# project itself will define the items and properties specific to that project, but the standard build rules for a Visual C# project are defined in the imported .targets files.
The $(MSBuildToolsPath) value specifies the path of these common .targets files. If the ToolsVersion is 4.0, the files are in the following location: WindowsInstallationPath\Microsoft.NET\Framework\v4.0.30319\

10.1. Common .Targets Files

.Targets file	Description
Microsoft.Common.targets	Defines the steps in the standard build process for Visual Basic and Visual C# projects. Imported by the Microsoft.CSharp.targets and Microsoft.VisualBasic.targets files, which include the following statement: <Import Project="Microsoft.Common.targets" />
Microsoft.CSharp.targets	Defines the steps in the standard build process for Visual C# projects. Imported by Visual C# project files (.csproj), which include the following statement: <Import Project="$(MSBuildToolsPath)\Microsoft.CSharp.targets" />
Microsoft.VisualBasic.targets	Defines the steps in the standard build process for Visual Basic projects. Imported by Visual Basic project files (.vbproj), which include the following statement: <Import Project="$(MSBuildToolsPath)\Microsoft.VisualBasic.targets" />

11. MSBuild Well-known Item Metadata

The following table describes the metadata assigned to every item upon creation. In each example, the following item declaration was used to include the file C:\MyProject\Source\Program.cs in the project.

```
<ItemGroup>
   <MyItem Include="Source\Program.cs" />
</ItemGroup>
```

Item Metadata %(FullPath) Description Contains the full path of the item. For example: C:\MyProject\Source\Program.cs
Item Metadata %(RootDir) Description Contains the root directory of the item. For example: C:\
Item Metadata %(Filename) Description Contains the file name of the item, without the extension. For example: Program
Item Metadata %(Extension) Description Contains the file name extension of the item. For example: .cs
Item Metadata %(RelativeDir) Description Contains the path specified in the Include attribute, up to the final backslash (\). For example: Source\

Item Metadata
%(Directory)

Description
Contains the directory of the item, without the root directory. For example:

MyProject\Source\

Item Metadata
%(RecursiveDir)

Description
If the Include attribute contains the wildcard **, this metadata specifies the part of the path that replaces the wildcard. For more information on wildcards.

If the folder C:\MySolution\MyProject\Source\ contains the file Program.cs, and if the project file contains this item:

```
<ItemGroup>

<MyItem Include="C:\**\Program.cs" />

</ItemGroup>
```

then the value of %(MyItem.RecursiveDir) would be MySolution\MyProject\Source\.

Item Metadata
%(Identity)

Description
The item specified in the Include attribute.. For example:

Source\Program.cs

Item Metadata
%(ModifiedTime)

Description
Contains the timestamp from the last time the item was modified. For example:

2004-07-01 00:21:31.5073316

Item Metadata
%(CreatedTime)

Description
Contains the timestamp from when the item was created. For example:

2004-06-25 09:26:45.8237425

Item Metadata
%(AccessedTime)

Description
Contains the timestamp from the last time the item was accessed.

2004-08-14 16:52:36.3168743

12. MSBuild Response Files

Response (.rsp) files are text files that contain MSBuild.exe command line switches. Each switch can be on a separate line or all switches can be on one line. Comment lines are prefaced with a **#** symbol. The **@** switch is used to pass another response file to MSBuild.exe.

The auto-response file is a special .rsp file that MSBuild.exe automatically uses when building a project. This file, MSBuild.rsp, must be in the same directory as MSBuild.exe, otherwise it will not be found. You can edit this file to specify default command line switches to MSBuild.exe. For example, if you use the same logger every time you build a project, you can add the **/logger** switch to MSBuild.rsp, and MSBuild.exe will use the logger every time a project is built.

14. WPF MSBuild Reference

Windows Presentation Foundation (WPF) extends Microsoft build engine (MSBuild) with additional build support, which is documented in this section.

In This Section

.Targets Files
Describes WPF .Targets files.
Task Reference
Lists the available WPF build tasks.
Microsoft.Build.Tasks
A build task assembly.
Microsoft.Build.Tasks.Deployment.Bootstrapper
A build task deployment bootstrapper assembly.
Microsoft.Build.Tasks.Deployment.ManifestUtilities
A build task deployment manifest utility assembly.
Microsoft.Build.Tasks.Hosting
A build task hosting assembly.
Microsoft.Build.Tasks.Windows
A build task windows assembly.

15. Special Characters to Escape

Special characters must be escaped only if they have special meaning in the context in which they are being used. For example, the asterisk (*) is a special character only in the "Include" and "Exclude" attributes of an item definition, or in a call to CreateItem. In all other cases, the asterisk is treated as a literal asterisk. While you do not need to escape asterisks everywhere in project files, doing so does no harm.

Use the notation %xx in place of the special character, where xx represents the hexadecimal value of the ASCII character. For example, to use an asterisk (*) as a literal character, use the value %2A.

The full list of special characters to escape follows:

Character	Description
%	Percent sign, used to reference metadata.
$	Dollar sign, used to reference properties.
@	At sign, used to reference item lists.
(	Open parenthesis, used in lists.
)	Close parenthesis, used in lists.
`	Apostrophe (or tick mark), used in conditions and other expressions.
;	

Description Semicolon, a list separator.
Character ? Description Question mark, a wildcard character when describing a file spec in an item's Include/Exclude section.
Character * Description Asterisk, a wildcard character when describing a file spec in an item's Include/Exclude section.


IN LUMINE SAPIENTIA
UNIVERSITAS ALMERIENSIS