

Automating Builds Using MSBuild



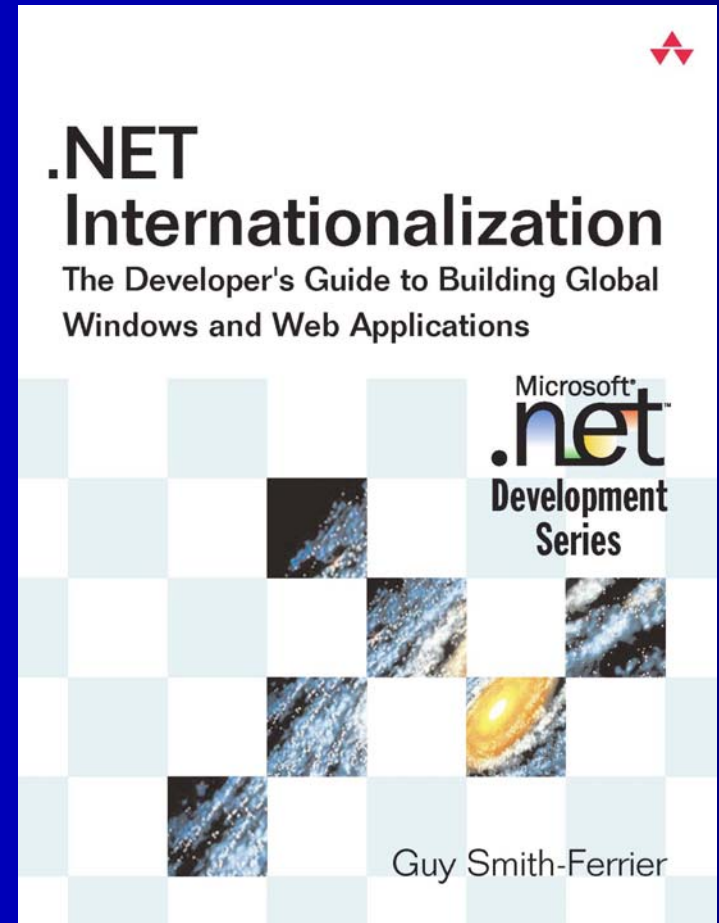
Guy Smith-Ferrier

guy@guysmithferrier.com

Blog: <http://www.guysmithferrier.com>

Author of...

- .NET Internationalization, Addison Wesley, ISBN 0321341384
- Visit <http://www.dotnet18n.com> to download the complete source code



Agenda

- MSBuild Fundamentals
- Items, Transforming, Metadata
- Standard Tasks
- Logging
- MSBuild Task Libraries
- Post Build Steps (inc. FxCop)
- Custom Tasks
- Building VS2003 Projects

Availability

- MSBuild is included in the .NET Framework 2.0
 - Visual Studio 2005 is not required
- MSBuild is included in the Windows Vista SDK

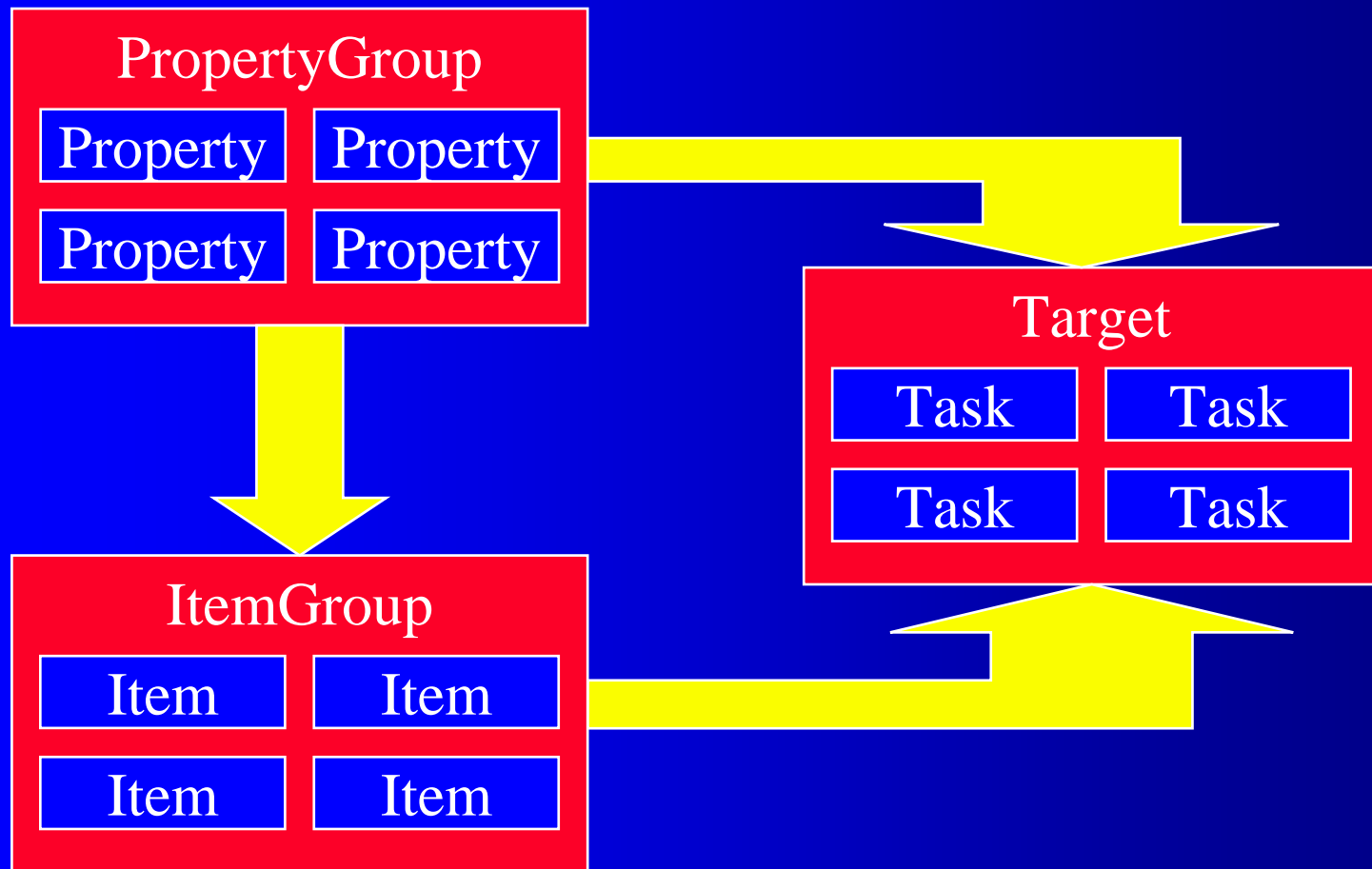
Information Sources

- Channel 9 MSBuild Home Page
 - <http://channel9.msdn.com/wiki/default.aspx/MSBuild.HomePage>
 - Nant Equivalents To MSBuild tasks
 - <http://channel9.msdn.com/wiki/default.aspx/MSBuild.EquivalentTasks>
- MSBuild Forum
 - <http://forums.microsoft.com/msdn/showforum.aspx?forumid=27&siteid=1>
- MSBuild Team Blog
 - <http://blogs.msdn.com/msbuild>
- Deploying .NET Applications with MSBuild and ClickOnce, Apress, Sayed Y. Hashimi and Sayed Ibrahim Hashimi, ISBN: 1-59059-652-8
- The Build Master, Addison-Wesley, Vincent Maraia, ISBN 0-321-33205-9
 - Microsoft's Software Configuration Management Best Practices

MSBuild Fundamentals

- Properties
 - PropertyGroups
- Items
 - ItemGroups
- Tasks
 - Targets

MSBuild Fundamentals (continued)



Properties

- Properties are name value pairs
 - "The variables of a build script"
 - e.g. An input directory
- Properties are part of a PropertyGroup

```
<PropertyGroup>  
    <OutputPath>\bin\Debug</OutputPath>  
</PropertyGroup>
```

- Properties are referenced as scalar values using `$()` syntax

```
$(OutputPath) // "\bin\Debug"
```

- Behave like properties in CSS files - they can be overridden
- Environment variables are available as properties

```
$(COMPUTERNAME)
```

Items

- Items are inputs
 - "The nouns of a build script"
 - e.g. a source file
- Items are part of an ItemGroup

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

- Items are referenced as groups using @() syntax

```
@(Compile) // "file1.cs", "file2.cs"
```

Tasks

- Tasks are steps to be performed in the build process
 - "The verbs of a build script"
- Tasks are part of a Target

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

- Tasks can use items and/or properties as inputs

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

Tasks (continued)

- Tasks can output items which are consumed by other tasks
- Many tasks are included in MSBuild libraries
- You can write custom tasks

Targets

- Targets are a collection of Tasks
 - "The sentences of a build script"
- Tasks are executed in the order in which they are defined in the Target

MSBuild Example 1

```
public class HelloWorld
{
    public static void Main(string[] args)
    {
        System.Console.WriteLine("Hello World");
    }
}
```

MSBuild Example 1 (continued)

```
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003">  
  <ItemGroup>  
    <Compile Include="HelloWorld.cs" />  
  </ItemGroup>  
  
  <Target Name="Build">  
    <Csc Sources="@ (Compile)" />  
  </Target>  
  
</Project>
```

MSBuild Example 1 (continued)

- Open a .NET Framework 2.0 command window
- Build the example using:-

```
msbuild /target:build
```

- Run the example using:-

```
HelloWorld.exe
```

MSBuild Example 2

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

  <PropertyGroup>
    <AssemblyName>HelloWorld</AssemblyName>
  </PropertyGroup>

  <ItemGroup>
    <Compile Include="HelloWorld.cs" />
  </ItemGroup>

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

  <Import Project="$(MSBuildBinPath)\Microsoft.CSharp.targets" />

</Project>
```

MSBuild Reserved Properties

MSBuildProjectDirectory	The absolute path of the project file
MSBuildProjectFile	The complete file name of the project file
MSBuildProjectExtension	The file name extension of the project file
MSBuildProjectFullPath	The absolute path and file name of the project file
MSBuildProjectName	The name of the project file without the extension
MSBuildBinPath	The absolute path of the MSBuild directory
MSBuildProjectDefaultTargets	The complete list of DefaultTargets targets
MSBuildExtensionsPath	The MSBuild custom targets folder under the Program Files directory

Incremental Builds

- If you tell a Target what its inputs and outputs are it will make a timestamp comparison between them and only perform the build if necessary

```
<Target Name="Build"  
  Inputs="@ (Compile) "  
  Outputs="$(AssemblyName) ">  
  <Csc Sources="@ (Compile) " />  
</Target>
```

Specifying Items

- Items can be specified individually

```
<Source Include="Form1.cs" />  
<Source Include="Form2.cs" />
```

- Items can be specified collectively

```
<Source Include="Form1.cs; Form2.cs" />
```

- Items can be specified using wildcards

```
<Source Include="Form?.cs" />  
<Source Include="*.cs" />
```

- Items can be excluded

```
<Source Include="*.cs" Exclude="Form1.cs" />
```

Transforming Items

- Item collections can be transformed using metadata

```
<ItemGroup>  
  <Source Include="*.resx" />  
</ItemGroup>
```

```
<Target Name="Show">  
  <Message Text="@ (Source->' %(filename).resources' )" />  
</Target>
```

Item Metadata

Metadata	Description
%(FullPath)	The full path of the item
%(RootDir)	The root directory of the item
%(Filename)	The file name of the item, without the extension
%(Extension)	The file name extension of the item
%(RelativeDir)	The directory path relative to the current working directory
%(Directory)	The directory of the item, without the root directory
%(RecursiveDir)	The recursed directory name
%(Identity)	The item specified in the Include attribute
%(ModifiedTime)	The timestamp from the last time the item was modified
%(CreatedTime)	The timestamp from when the item was created
%(AccessedTime)	The timestamp from the last time the time was accessed

Recursing Through Directories

- Use "**" to indicate the folders should be recursed

```
<SourceDir>C:\Books\I18N\Source\VS2005\**</SourceDir>
```

- Use metadata tags to get the recursed information

```
$(TargetDir)%(RecursiveDir)%(Filename)%(Extension)
```

Recurring Through Directories (continued)

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

  <PropertyGroup>
    <SourceDir>C:\Books\I18N\Source\VS2005\**</SourceDir>
    <TargetDir>C:\Program Files\DotNetI18N\Source\VS2005\
    </TargetDir>
  </PropertyGroup>

  <ItemGroup>
    <Source Include="$(SourceDir)\*.cs" />
  </ItemGroup>

  <Target Name="CopyFiles">
    <Copy SourceFiles="@ (Source)" DestinationFiles="@ (Source->
    '$ (TargetDir)%(RecursiveDir)%(Filename)%(Extension)') " />
  </Target>

</Project>
```

Conditions

```
<PropertyGroup>  
  <SourceDir>C:\Books\I18N\Source</SourceDir>  
  <VS2005SourceDir>$(SourceDir)\VS2005</VS2005SourceDir>  
  <VS2003SourceDir>$(SourceDir)\VS2003</VS2003SourceDir>  
</PropertyGroup>
```

```
<ItemGroup>  
  <SourceFiles Condition="'$(Source)'=='VS2003' "  
    Include="$(VS2003SourceDir)\**\*.cs" />  
  
  <SourceFiles Condition="'$(Source)'=='VS2005' "  
    Include="$(VS2005SourceDir)\**\*.cs" />  
</ItemGroup>
```

Choose / When / Otherwise

```
<Choose>
```

```
  <When Condition="'$(Source)' == 'VS2003' ">
```

```
    <ItemGroup>
```

```
      <SourceFiles Include="$(VS2003SourceDir)\**\*.cs" />
```

```
    </ItemGroup>
```

```
  </When>
```

```
  <When Condition="'$(Source)' == 'VS2005' ">
```

```
    <ItemGroup>
```

```
      <SourceFiles Include="$(VS2005SourceDir)\**\*.cs" />
```

```
    </ItemGroup>
```

```
  </When>
```

```
</Choose>
```

Standard MSBuild Tasks

Wrappers For Command Line Tools

AL	Wraps al.exe (Assembly Linker)
AspNetCompiler	Wraps aspnet_compiler.exe
Csc	Wraps csc.exe (C# Compiler)
GenerateResource	Converts .txt and .resx to .resources (like ResGen.exe)
LC	Wraps LC.exe (License Compiler)
MSBuild	Builds MSBuild projects from another MSBuild project
RegisterAssembly	Wraps RegAsm.exe (Assembly Registration Tool)
ResGen	Use GenerateResource instead
SGen	Wraps SGen.exe (XML Serializer Generator Tool)
SignFile	Signs the specified file using the specified certificate
Vbc	Wraps vbc.exe (Visual Basic.NET compiler)
VCBuild	Wraps vcbuild.exe (Visual C++ compiler)

Standard MSBuild Tasks

File And Directory Manipulation

Copy	Copies files on the filesystem to a new location
Delete	Deletes the specified files
FindUnderPath	Determines which items in the item collection exist
GetFrameworkPath	Retrieves the path to the .NET Framework
GetFrameworkSdkPath	Retrieves the path to the .NET Framework SDK
MakeDir	Creates directories
ReadLinesFromFile	Reads a list of items from a text file
RemoveDir	Removes directories
Touch	Sets the access and modification times of files
WriteLinesToFile	Writes the specified items to the specified text file

Standard MSBuild Tasks

Item And Property Manipulation

AssignCulture	Assigns culture identifiers to items
CreateItem	Populates item collections with the input items
CreateProperty	Populates properties with the values passed in
Exec	Runs the specified program or command with the specified arguments

Standard MSBuild Tasks

ClickOnce Tasks

<code>GenerateApplicationManifest</code>	Generates a ClickOnce application manifest
<code>GenerateBootstrapper</code>	Detects, downloads, and installs an application
<code>GenerateDeploymentManifest</code>	Generates a ClickOnce deployment manifest

Standard MSBuild Tasks

Assembly And COM Manipulation

GetAssemblyIdentity	Retrieves the assembly identities from the specified files and outputs the identity information.
ResolveAssemblyReference	Determines all assemblies that depend on the specified assemblies
ResolveComReference	Resolves type libraries to locations on disk
ResolveKeySource	Determines the strong name key source
ResolveNativeReference	Resolves native references
UnregisterAssembly	Unregisters the specified assemblies for COM interop purposes

Tasks That Pass Outputs To Other Tasks

```
<Target Name="CopyAndTouchFiles">
  <Copy
    SourceFiles="@Source"
    DestinationFiles="@Source->
      '$(TargetDir)%(RecursiveDir)%(Filename)%(Extension)'">

    <Output
      TaskParameter="CopiedFiles"
      ItemName="CopiedFiles"
    />

  </Copy>

  <Touch Time="1:01" Files="@CopiedFiles" />
</Target>
```

Opening Project Files As MSBuild Files

- In Solution Explorer, right click the project and select Unload Project then right click again and select Edit <Project Name>
 - Syntax highlighting
 - Intellisense

MSBuild Sidekick

- MSBuild Sidekick is a free GUI for editing MSBuild project files
 - <http://www.attrice.info/msbuild/index.htm>

MSBuild Command Line Parameters

Switch	Description
/help, /?	Displays usage information
/nologo	Hide the startup banner and copyright message
/version	Display version information only
@file	Insert command line settings from a text file
/noautoreponse	Do not auto-include the MSBuild.rsp file
/target:targets	Build these targets in this project
/property:name=value	Set or override these project-level properties
/logger:logger	Specifies the logger to use to log events
/consoleloggerparameters:parameters	Specifies the parameters to pass to the console logger
/verbosity:level	Display this amount of information in the event log
/noconsolelogger	Disable the default console logger
/validate:schema	Validates the project file, and builds the project

Build Steps

- Clean target
- Get latest source from version control
- Compile
- Execute static analysis (e.g. FxCop, style analyzer)
- Execute unit tests
- Calculate statistics (e.g. Code Coverage)
- Build deployment files
- Deploy files
- Generate build report
- Send build report

Logging

- MSBuild ships with two loggers for logging build output
 - ConsoleLogger
 - FileLogger
- The ConsoleLogger is used by default
- To use an additional logger use the MSBuild /logger switch
 - msbuild /logger:[LoggerClass,]LoggerAssembly[;LoggerParameters]

```
msbuild /logger:FileLogger,Microsoft.Build.Engine;LogFile=Results.txt
```

- To disable the ConsoleLogger use the MSBuild /noconsolelogger switch

Custom Loggers

- You can write your own loggers by implementing the ILogger interface

```
public interface ILogger
{
    void Initialize(IEventSource eventSource);
    void Shutdown();
    string Parameters { get; set; }
    LoggerVerbosity Verbosity { get; set; }
}
```

- There are a number of custom loggers available on the Internet including XmlLoggers
 - <http://geekswithblogs.net/kobush/articles/65849.aspx>

Debugging Builds

- Use the <Message> task to output debugging messages to the logger
- Use the msbuild /verbosity switch to set the level of detail in the output
 - /verbosity:quiet
 - /verbosity:minimal
 - /verbosity:normal
 - /verbosity:detailed
 - /verbosity:diagnostic

Microsoft Services (UK) Enterprise Solutions Build Framework (SBF)

- SBF is a build framework including a massive library of tasks
- Also called "Enterprise Solutions Build Framework" and ".NET SDC Solution Build & Deployment Process And Tools"
- Download from <http://www.gotdotnet.com> (search on "SBF" in "Code Gallery") or <http://www.buildframework.com>
- Tasks included cover:-
 - ActiveDirectory, BizTalk 2004, CAB, Certificates, Code Coverage, Component Services, Event Log, File, Folder, GAC, MSMQ, MSI, Performance Counters, Registry, Security, SourceSafe, SQL Server, Tools, Virtual Server, Web, WIX, XML, ZIP

MSBuild Community Tasks

- The MSBuild Community Tasks Project is an open source project for MSBuild tasks
- Download from <http://msbuildtasks.com/default.aspx>
- Tasks included cover:-
 - Application Pools, File Manipulation, FTP, EMail, Maths, Registry, NDoc, Regex, Services, SQL Commands, Subversion, ZIP, Visual SourceSafe, Web Directories, XML

Post Build Steps

- Static Analysis
 - FxCop
 - PREfast
- Unit Testing
 - Visual Studio Team Test
 - NUnit
- Code Coverage
 - CoverageEye.NET (www.gotdotnet.com)

Post Build Steps FxCop (Exec)

```
<Target Name="Build">  
  
  <Csc Sources="@ (Compile)" />  
  
  <Exec Command="FxCopCmd.exe  
/project: C: \CS2Tests\HelloWorld\HelloWorld.FxCop  
/out: C: \CS2Tests\HelloWorld\HelloWorld.FxCop.xml "  
WorkingDirectory="C: \Program Files\Microsoft FxCop 1.35" />  
  
</Target>
```

Post Build Steps FxCop (SDC Task)

```
<Target Name="Test">  
  
  <Tool s. FxCop  
    ContinueOnError="true"  
    Assemblies="@ (Assemblies->' %(FullPath) ')"  
    ProjectFilePath="$(MSBuildProjectDirectory)\Output\Main\Main. FxCop"  
    ProjectTemplateFilePath="$(MSBuildProjectDirectory)\FxCop\main. FxCop"  
    OutFileName="$(MSBuildProjectDirectory)\Output\FxCopErrors. xml "  
  >  
  
    <Output  
      TaskParameter="Total Violations"  
      ItemName="FxCopViolations"  
    />  
  
  </Tool s. FxCop>  
  
  <Message Text="@ (FxCopViolations) FxCop violations" />  
  
</Target>
```

Web Deployment Projects

- Web Deployment Projects are MSBuild projects for ASP.NET 2 web sites
- Web Deployment Projects add many build features to web projects including:-
 - Pre-build actions
 - Post-build actions
- Download the Web Deployment Package Visual Studio 2005 add-in from:-
 - <http://msdn.microsoft.com/asp.net/reference/infrastructure/wdp/default.aspx>

ProjectFileConverter

- Microsoft.Build.Conversion.ProjectFileConverter converts Visual Studio 2002 and 2003 projects to Visual Studio 2005 projects
- Microsoft might release msbuildconvert.exe in the future which is a command line wrapper for ProjectFileConverter
- There is no MSBuild task which wraps up the ProjectFileConverter class

ProjectFileConverter (continued)

```
ProjectFileConverter projectFileConverter =  
    new ProjectFileConverter();  
  
projectFileConverter.OldProjectFile =  
    @"\\VS2003\\WindowsApplication1\\WindowsApplication1.csproj";  
  
projectFileConverter.NewProjectFile =  
    @"\\VS2005\\WindowsApplication1\\WindowsApplication1.csproj";  
  
projectFileConverter.Convert(  
    @"C:\\WINDOWS\\Microsoft.NET\\Framework\\v2.0.50727");
```

Custom Tasks

- Create a new class library
- Add References to Microsoft.Build.Utilities, Microsoft.Build.Framework
- Classes inherit from Task and override the Execute method
- Properties can be marked with the [Required] or [Output] attributes
- Make Tasks available to MSBuild using `<UsingTask>`

Custom Tasks (continued)

```
public class ProjectFileConverterTask: Task
{
    private string oldProjectFile;
    [Required]
    public string OldProjectFile
    {
        get { return oldProjectFile; }
        set { oldProjectFile = value; }
    }
    private string newProjectFile;
    [Required]
    public string NewProjectFile
    {
        get { return newProjectFile; }
        set { newProjectFile = value; }
    }
}
```

Custom Tasks (continued)

```
public override bool Execute()
{
    ProjectFileConverter projectFileConverter =
        new ProjectFileConverter();

    projectFileConverter.OldProjectFile = oldProjectFile;

    projectFileConverter.NewProjectFile = newProjectFile;

    projectFileConverter.Convert(
        @"C:\WINDOWS\MicrosofT.NET\Framework\v2.0.50727");

    return true;
}
}
```

Custom Tasks (continued)

```
<Project
xml ns="http://schemas.microsoft.com/developer/msbuild/2003">

  <UsingTask TaskName="ProjectFileConverterTask"
  AssemblyFile="C:\Projects\Tasks\bin\Debug\Tasks.dll"/>

  <Target Name="Convert">
    <ProjectFileConverterTask
      OldProjectFile="C:\VS2003\WindowsApplication1.csproj"
      NewProjectFile="C:\VS2005\WindowsApplication1.csproj"/>
  </Target>

</Project>
```

MSBee (MSBuild Extras)

- MSBuild Extras - Toolkit for .NET 1.1
 - Part of the Power Toys for Visual Studio
 - Developed by the Developer Solutions Team
 - <http://www.codeplex.com/Wiki/View.aspx?ProjectName=MSBee>
- MSBee Strategy
 - MSBuild does not recognise VS2002/2003 project files
 - Convert the project to Visual Studio 2005
 - MSBee is a library of tasks which use the .NET Framework SDK 1.1

Microsoft.Build.Engine

```
Engine engine = new Engine();  
  
engine.BinPath =  
    @"c:\windows\microsoft.net\Framework\v2.0.50727";  
  
FileLogger logger = new FileLogger();  
  
logger.Parameters = @"logfile=C:\temp\build.log";  
  
engine.RegisterLogger(logger);  
  
engine.BuildProjectFile(@"c:\temp\valide.proj");
```

Summary

- MSBuild is a powerful and robust build engine
- It can be easily customized
 - Custom Tasks
 - Custom Loggers
 - Build your own build engine from the Engine class
- There are many add-ons for MSBuild which dramatically improve productivity
 - MSBuild SideKick
 - Solutions Build Framework
 - MSBuild Community Tasks
 - Web Deployment Projects
 - MSBee