

# Leveraging MSBuild 4.0





---

# ACCENTIENT EDUCATION SERIES

Committed to training success

---

+1 (877) 710-0841 ■ [www.accentient.com](http://www.accentient.com)

## Leveraging MSBuild 4.0

<b>Course Number:</b>	<b>MSB</b>
<b>Version:</b>	<b>1.1</b>
<b>For software version:</b>	<b>4.0</b>

Copyright © 2011 Accentient, Inc. All rights reserved.

No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose without the express written permission of Accentient, Inc.

**Images and Artwork**

Images and artwork in this book were obtained through Flickr and are licensed under a Creative Commons 3.0 license. Visit <http://shrinkster.com/1buy> for more information.

All trademarks referenced are the property of their respective owners

---

**Disclaimer**

While Accentient takes great care to ensure the accuracy and quality of these materials, all material is provided without any warranty whatsoever, including, but not limited to, the implied warranties of merchantability or fitness for a particular purpose.

# **Leveraging MSBuild 4.0**

## **Module 2 Visual C++ Support**

### **Topics**

- The .vcxproj file
- Migrating from earlier versions
- The C++ Build Process
- Configuring build behavior
- Parallel Builds
- Incremental Builds
- Property Sheets and Pages
- Lab

## MSBuild in Visual C++ 2010

- Starting with the 2010 version, Visual C++ now uses MSBuild as its build engine
  - MSBuild replaces VCBUILD
- MSBuild's scalability and performance is appealing to C++ developers

## .VCXPROJ project files

- This is the new, MSBuild formatted project file
  - Contrast this to the .vcproj file extension (VCBuild)
- .VCXPROJ files have a specific layout
  - If this layout is not followed, the build results may not be like you expected

```
<Project DefaultTargets="Build" ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003" >  
  <ItemGroup Label="ProjectConfigurations" />  
  <PropertyGroup Label="Globals" />  
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.default.props" />  
  <PropertyGroup Label="Configuration" />  
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.props" />  
  <ImportGroup Label="ExtensionSettings" />  
  <ImportGroup Label="PropertySheets" />  
  <PropertyGroup Label="UserMacros" />  
  <PropertyGroup />  
  <ItemDefinitionGroup />  
  <ItemGroup />  
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.targets" />  
  <ImportGroup Label="ExtensionTargets" />  
</Project>
```

- **Note:** The *Label* attribute is only used by Visual Studio

## Anatomy of a .VCXPROJ File

```
<Project DefaultTargets="Build" ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003" >  
  <ItemGroup Label="ProjectConfigurations" />  
  <PropertyGroup Label="Globals" />  
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.default.props" />  
  <PropertyGroup Label="Configuration" />  
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.props" />  
  <ImportGroup Label="ExtensionSettings" />  
  <ImportGroup Label="PropertySheets" />  
  <PropertyGroup Label="UserMacros" />  
  <PropertyGroup />  
  <ItemDefinitionGroup />  
  <ItemGroup />  
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.targets" />  
  <ImportGroup Label="ExtensionTargets" />  
</Project>
```

This contains the project configurations known to the project such as Debug|Win32 and Release|Win32.

## Anatomy of a .VCXPROJ File

```
<Project DefaultTargets="Build" ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003" >  
  <ItemGroup Label="ProjectConfigurations" />  
  <PropertyGroup Label="Globals" />  
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.default.props" />  
  <PropertyGroup Label="Configuration" />  
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.props" />  
  <ImportGroup Label="ExtensionSettings" />  
  <ImportGroup Label="PropertySheets" />  
  <PropertyGroup Label="UserMacros" />  
  <PropertyGroup />  
  <ItemDefinitionGroup />  
  <ItemGroup />  
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.targets" />  
  <ImportGroup Label="ExtensionTargets" />  
</Project>
```

This contains project level settings such as ProjectGuid, RootNamespace, etc.

These properties are not normally overridden elsewhere in the file.

This group is not configuration-dependent and so only one Globals group generally exists in the project file.

## Anatomy of a .VCXPROJ File

```
<Project DefaultTargets="Build" ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003" >
  <ItemGroup Label="ProjectConfigurations" />
  <PropertyGroup Label="Globals" />
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.default.props" />
  <PropertyGroup Label="Configuration" />
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.props" />
  <ImportGroup Label="ExtensionSettings" />
  <ImportGroup Label="PropertySheets" />
  <PropertyGroup Label="UserMacros" />
  <PropertyGroup />
  <ItemDefinitionGroup />
  <ItemGroup />
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.targets" />
  <ImportGroup Label="ExtensionTargets" />
</Project>
```

This property sheet contains the default settings for a VC++ project.

It contains definitions of all the project settings such as Platform, PlatformToolset, OutputPath, TargetName, UseOfAtl, etc. and also all the item definition group defaults for each known item group.

In general, properties in this file are not tool-specific.

## Anatomy of a .VCXPROJ File

```
<Project DefaultTargets="Build" ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003" >
  <ItemGroup Label="ProjectConfigurations" />
  <PropertyGroup Label="Globals" />
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.default.props" />
  <PropertyGroup Label="Configuration" />
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.props" />
  <ImportGroup Label="ExtensionSettings" />
  <ImportGroup Label="PropertySheets" />
  <PropertyGroup Label="UserMacros" />
  <PropertyGroup />
  <ItemDefinitionGroup />
  <ItemGroup />
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.targets" />
  <ImportGroup Label="ExtensionTargets" />
</Project>
```

This property group has a configuration condition and comes in multiple copies, one per configuration.

This property group hosts configuration-wide properties that control the inclusion of system property sheets in Microsoft.Cpp.props.

Example: if you define the property `<CharacterSet>Unicode</CharacterSet>`, then the system property sheet `microsoft.Cpp.unicodesupport.props` will be included.

## Anatomy of a .VCXPROJ File

```
<Project DefaultTargets="Build" ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003" >
  <ItemGroup Label="ProjectConfigurations" />
  <PropertyGroup Label="Globals" />
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.default.props" />
  <PropertyGroup Label="Configuration" />
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.props" />
  <ImportGroup Label="ExtensionSettings" />
  <ImportGroup Label="PropertySheets" />
  <PropertyGroup Label="UserMacros" />
  <PropertyGroup />
  <ItemDefinitionGroup />
  <ItemGroup />
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.targets" />
  <ImportGroup Label="ExtensionTargets" />
</Project>
```

This property sheet (directly or via imports) defines the default values for many tool-specific properties such as the compiler's Optimization, WarningLevel properties, Midl tool's TypeLibraryName property, etc.

In addition, it imports various system property sheets based on configuration properties defined in the property group immediately above.

## Anatomy of a .VCXPROJ File

```
<Project DefaultTargets="Build" ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003" >
  <ItemGroup Label="ProjectConfigurations" />
  <PropertyGroup Label="Globals" />
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.default.props" />
  <PropertyGroup Label="Configuration" />
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.props" />
  <ImportGroup Label="ExtensionSettings" />
  <ImportGroup Label="PropertySheets" />
  <PropertyGroup Label="UserMacros" />
  <PropertyGroup />
  <ItemDefinitionGroup />
  <ItemGroup />
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.targets" />
  <ImportGroup Label="ExtensionTargets" />
</Project>
```

This group contains imports for the property sheets that are part of Build Customizations (or Custom Build Rules as this feature was called in earlier editions).

A Build Customization is defined by up to three files – a .targets file, a .props file, and an .xml file.

This import group contains the imports for the .props file.

## Anatomy of a .VCXPROJ File

```
<Project DefaultTargets="Build" ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003" >
  <ItemGroup Label="ProjectConfigurations" />
  <PropertyGroup Label="Globals" />
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.default.props" />
  <PropertyGroup Label="Configuration" />
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.props" />
  <ImportGroup Label="ExtensionSettings" />
  <ImportGroup Label="PropertySheets" />
  <PropertyGroup Label="UserMacros" />
  <PropertyGroup />
  <ItemDefinitionGroup />
  <ItemGroup />
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.targets" />
  <ImportGroup Label="ExtensionTargets" />
</Project>
```

This group contains the imports for user property sheets.

These are the property sheets you add through the Property Manager view in Visual Studio.

The order in which these imports are listed is relevant and is reflected in the Property Manager.

The project file normally contains multiple instances of this kind of import group, one for each project configuration.

## Anatomy of a .VCXPROJ File

```
<Project DefaultTargets="Build" ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003" >
  <ItemGroup Label="ProjectConfigurations" />
  <PropertyGroup Label="Globals" />
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.default.props" />
  <PropertyGroup Label="Configuration" />
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.props" />
  <ImportGroup Label="ExtensionSettings" />
  <ImportGroup Label="PropertySheets" />
  <PropertyGroup Label="UserMacros" />
  <PropertyGroup />
  <ItemDefinitionGroup />
  <ItemGroup />
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.targets" />
  <ImportGroup Label="ExtensionTargets" />
</Project>
```

UserMacros are used as variables to customize your build process.

Example: you can define a user macro to define your custom output path as \$(CustomOutputPath) and use it to define other variables.

This property group houses such properties.

These properties are not populated by the Visual Studio IDE though.

## Anatomy of a .VCXPROJ File

```
<Project DefaultTargets="Build" ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003" >
  <ItemGroup Label="ProjectConfigurations" />
  <PropertyGroup Label="Globals" />
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.default.props" />
  <PropertyGroup Label="Configuration" />
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.props" />
  <ImportGroup Label="ExtensionSettings" />
  <ImportGroup Label="PropertySheets" />
  <PropertyGroup Label="UserMacros" />
  <PropertyGroup />
  <ItemDefinitionGroup />
  <ItemGroup />
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.targets" />
  <ImportGroup Label="ExtensionTargets" />
</Project>
```

This property group normally comes with a configuration condition attached. You will also see multiple instances of the property group, one per configuration.

It differs in its identity from the other property groups above by the fact that it does not have a label.

This group contains project configuration level settings. These settings apply to all files that are part of the specified item group.

Build Customization item definition metadata also gets initialized here.

## Anatomy of a .VCXPROJ File

```
<Project DefaultTargets="Build" ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003" >
  <ItemGroup Label="ProjectConfigurations" />
  <PropertyGroup Label="Globals" />
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.default.props" />
  <PropertyGroup Label="Configuration" />
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.props" />
  <ImportGroup Label="ExtensionSettings" />
  <ImportGroup Label="PropertySheets" />
  <PropertyGroup Label="UserMacros" />
  <PropertyGroup />
  <ItemDefinitionGroup />
  <ItemGroup />
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.targets" />
  <ImportGroup Label="ExtensionTargets" />
</Project>
```

Similar to the property group immediately above, but it contains item definitions and item definition metadata instead of properties.

## Anatomy of a .VCXPROJ File

```
<Project DefaultTargets="Build" ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003" >
  <ItemGroup Label="ProjectConfigurations" />
  <PropertyGroup Label="Globals" />
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.default.props" />
  <PropertyGroup Label="Configuration" />
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.props" />
  <ImportGroup Label="ExtensionSettings" />
  <ImportGroup Label="PropertySheets" />
  <PropertyGroup Label="UserMacros" />
  <PropertyGroup />
  <ItemDefinitionGroup />
  <ItemGroup />
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.targets" />
  <ImportGroup Label="ExtensionTargets" />
</Project>
```

Contains the items (source files, etc.) in the project.

You will generally have multiple item groups – one per item type.

## Anatomy of a .VCXPROJ File

```
<Project DefaultTargets="Build" ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003" >
  <ItemGroup Label="ProjectConfigurations" />
  <PropertyGroup Label="Globals" />
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.default.props" />
  <PropertyGroup Label="Configuration" />
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.props" />
  <ImportGroup Label="ExtensionSettings" />
  <ImportGroup Label="PropertySheets" />
  <PropertyGroup Label="UserMacros" />
  <PropertyGroup />
  <ItemDefinitionGroup />
  <ItemGroup />
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.targets" />
  <ImportGroup Label="ExtensionTargets" />
</Project>
```

Defines (directly or via imports) VC++ targets such as build, clean, etc.

## Anatomy of a .VCXPROJ File

```
<Project DefaultTargets="Build" ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003" >
  <ItemGroup Label="ProjectConfigurations" />
  <PropertyGroup Label="Globals" />
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.default.props" />
  <PropertyGroup Label="Configuration" />
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.props" />
  <ImportGroup Label="ExtensionSettings" />
  <ImportGroup Label="PropertySheets" />
  <PropertyGroup Label="UserMacros" />
  <PropertyGroup />
  <ItemDefinitionGroup />
  <ItemGroup />
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.targets" />
  <ImportGroup Label="ExtensionTargets" />
</Project>
```

This group contains imports for the Build Customization target files.

## Migrating from VC++ 2008 (and Earlier)

- IDE Conversion
  - Visual Studio 2010 comes with a built-in project upgrader
  - A wizard runs when you try to open an older VC++ project
    - VC6 (.dsp and .dsw files)
    - VS2002-2008 (.vcproj and .sln files)
  - Backup your files first, or have the wizard do it for you

## Visual Studio Conversion Wizard

- If the solution or project is under source control
  - It will be checked out automatically during the conversion
  - Make sure the correct Source Control Plug-in is active
  - Make sure no files are exclusively checked out
- If the solution or project is not under source control
  - Ensure that all files have read/write permissions
- The wizard can't upgrade Smart Device projects

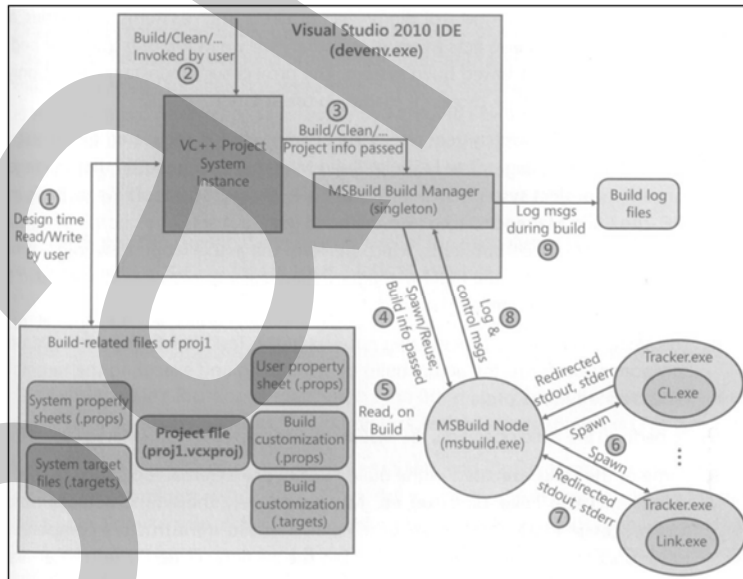
## Command-Line Conversion

- Great if you have a lot of projects to convert
  - Especially when they are not in the same solution
- Devenv.exe
  - This is the Visual Studio 2010 IDE
  - `devenv.exe /upgrade YourSolution.sln`
  - `devenv.exe /upgrade YourProject.vcproj`
- Vcupgrade.exe
  - A new tool located in %VS100COMNTOOLS% whose sole purpose is to upgrade VC project files
  - Vcupgrade.exe can convert from older .dsp format
  - `vcupgrade.exe YourProject.dsp`

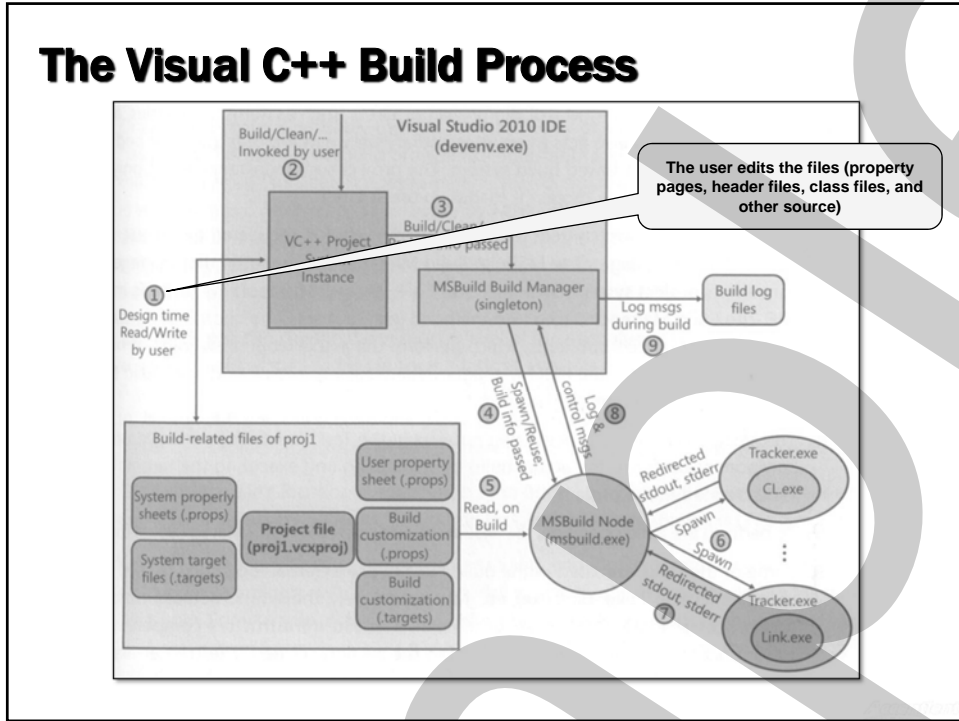
## What Gets Converted

What	From	To
Project File	.dsp/.vcproj	.vcxproj + .vcxproj.filters
Solution File	.dsw/.sln	.sln
Property Sheets	.vsprops	.props
Build Customizations	.rules	.xml + .props + .targets

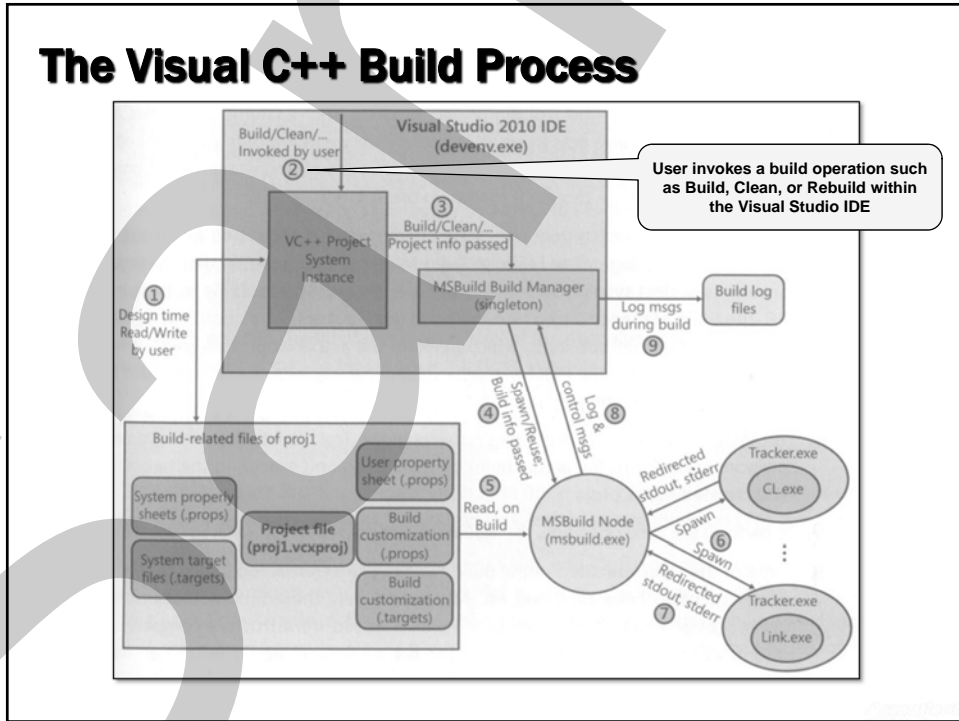
## The Visual C++ Build Process



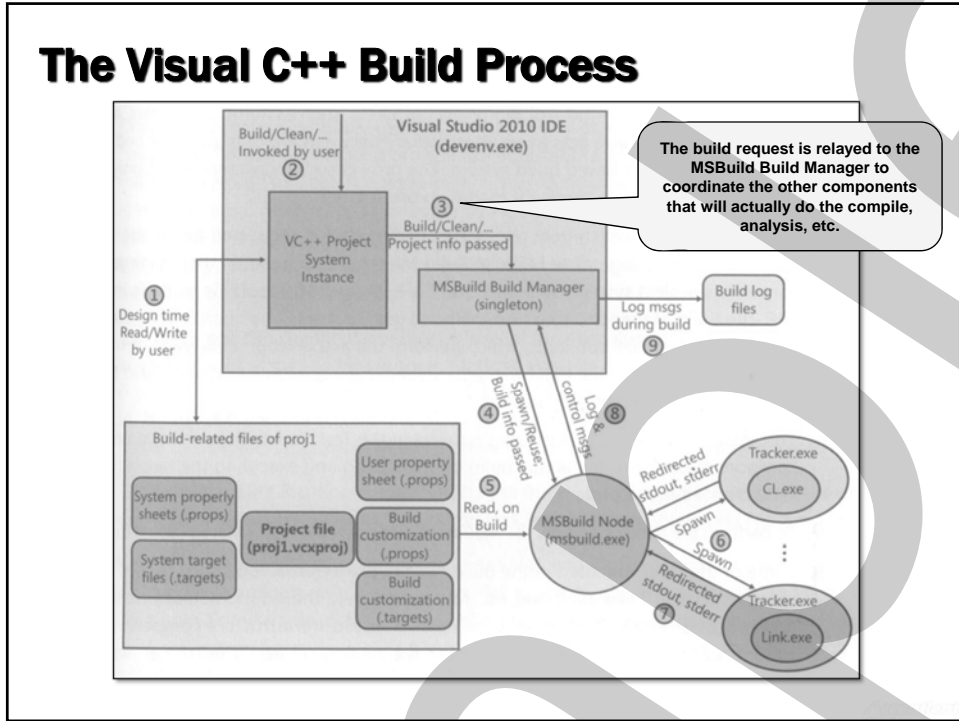
## The Visual C++ Build Process



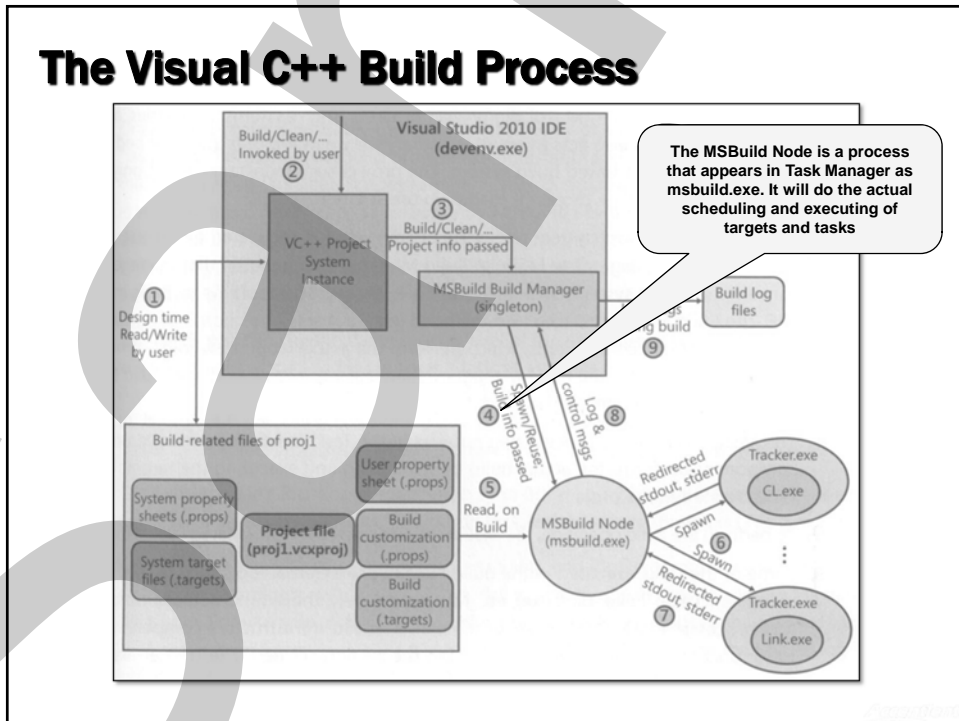
## The Visual C++ Build Process



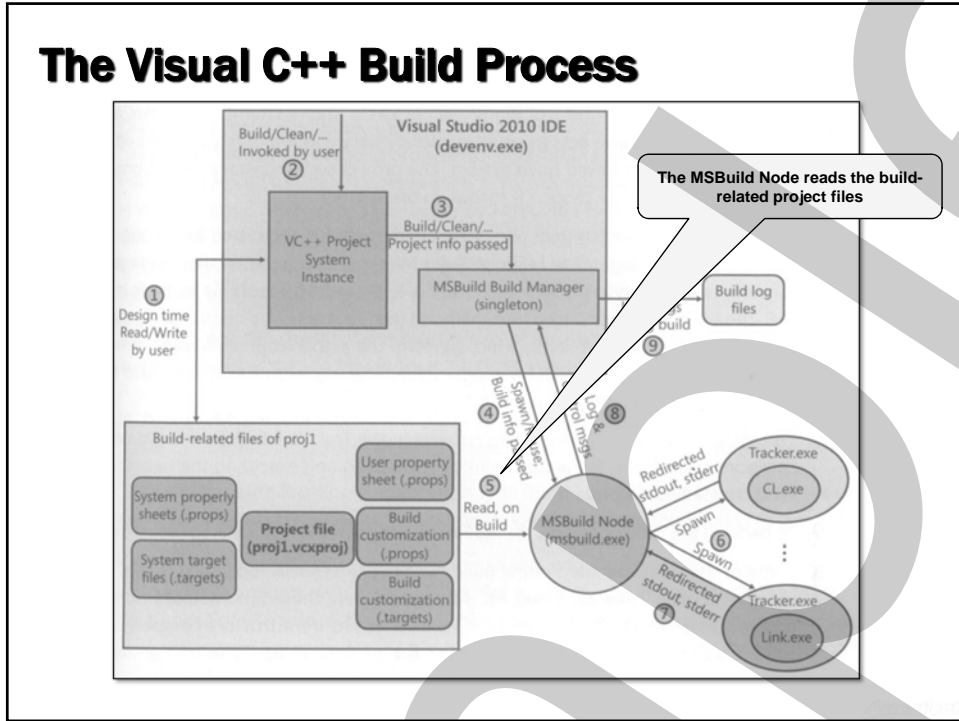
## The Visual C++ Build Process



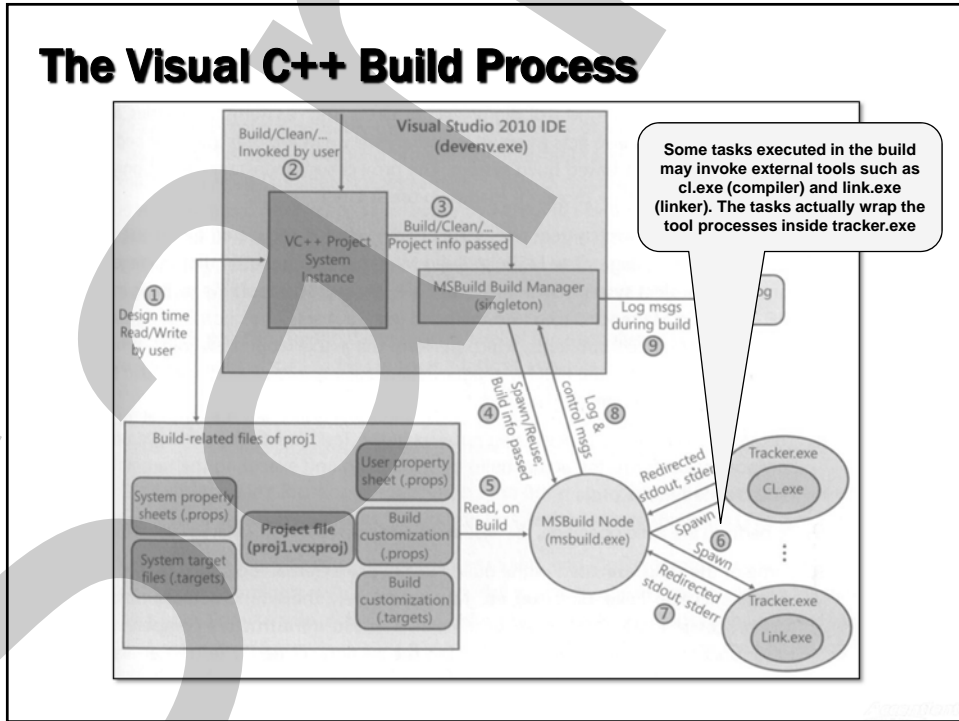
## The Visual C++ Build Process



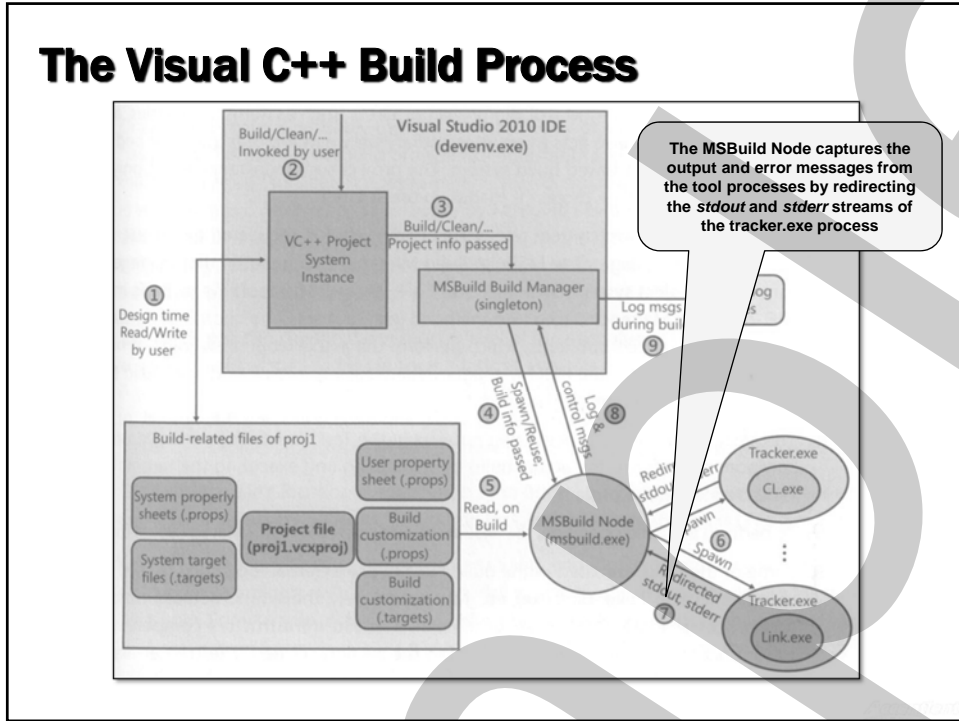
## The Visual C++ Build Process



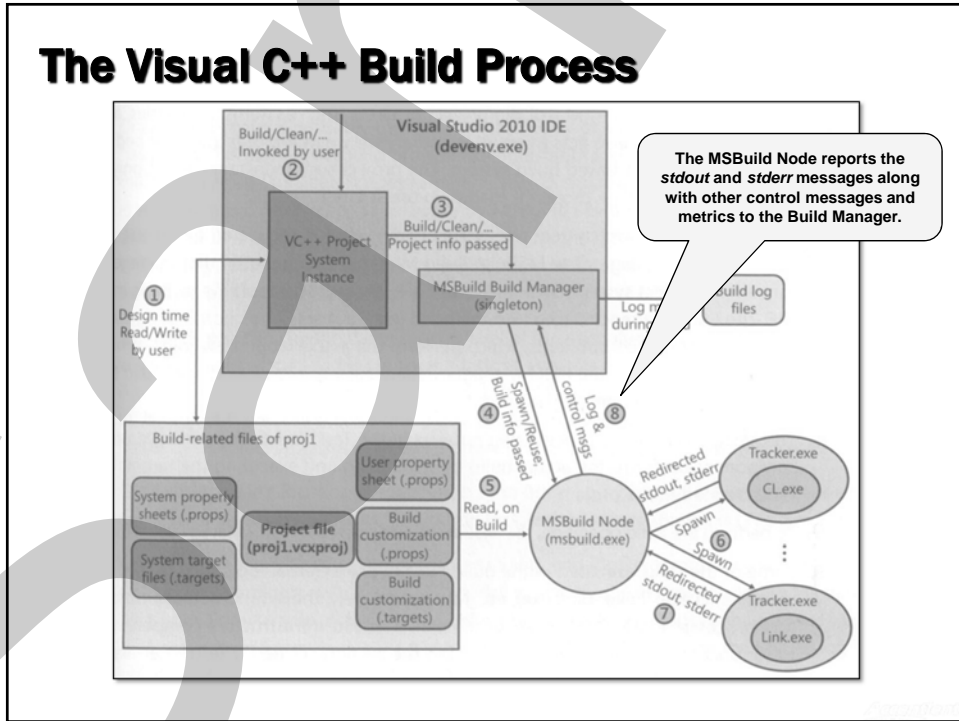
## The Visual C++ Build Process



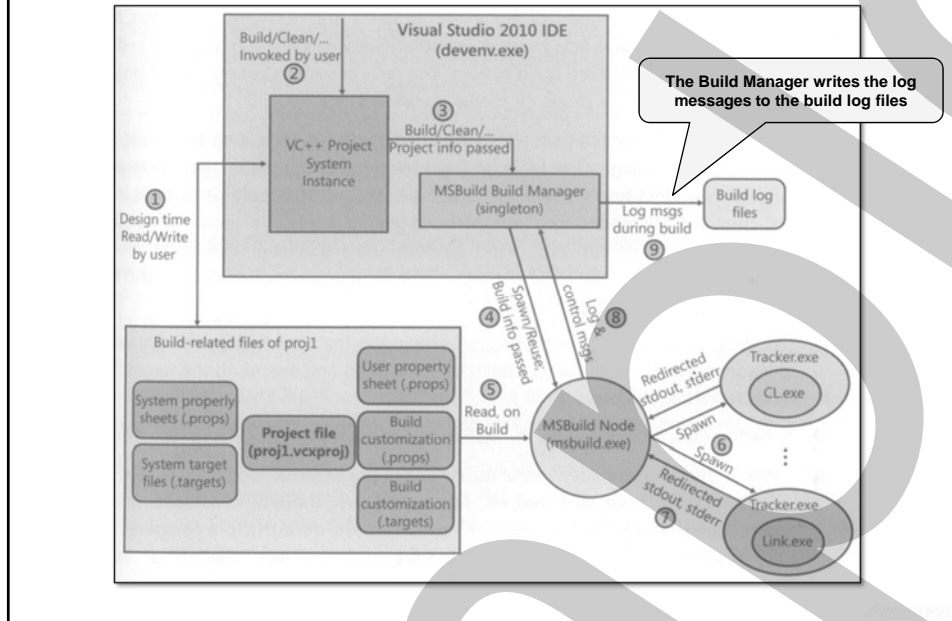
## The Visual C++ Build Process



## The Visual C++ Build Process



## The Visual C++ Build Process



## Visual C++ Builds are Out-of-Process

- The Build Manager will be executing one or more external tools
  - These tools will be executed out-of-process
- This enables parallel builds
  - Each build process can run on its own CPU/core

## MSBuild Tasks Specific to Visual C++

Task	Description
<b>BscMake</b>	Wraps the Microsoft Browse Information Maintenance Utility tool (bscmake.exe)
<b>CL</b>	Wraps the Visual C++ compiler tool (cl.exe)
<b>CPPClean</b>	Deletes the temporary files that MSBuild creates when a Visual C++ project is built
<b>LIB</b>	Wraps the Microsoft 32-Bit Library Manager tool (lib.exe)
<b>Link</b>	Wraps the Visual C++ linker tool (link.exe)
<b>MIDL</b>	Wraps the Microsoft Interface Definition Language (MIDL) compiler tool (midl.exe)
<b>MT</b>	Wraps the Microsoft Manifest Tool (mt.exe)
<b>RC</b>	Wraps the Microsoft Windows Resource Compiler tool (rc.exe)
<b>SetEnv</b>	Sets or deletes the value of a specified environment variable
<b>VCMMessage</b>	Logs warning messages and error messages during a build
<b>XDCMake</b>	Wraps the XML Documentation tool (xdcmake.exe), which merges XML document comment (.xdc) files into an .xml file
<b>XSD</b>	Wraps the XML Schema Definition tool (xsd.exe), which generates schema or class files from a source

## Build Parallelism

- “Build Parallelism” is the act of enlisting multiple cores (CPUs) to simultaneously compile the projects in a solution and/or the files in a project
- There are two types of build parallelisms that MSBuild can take advantage of
  - Project level
  - File level
- **Note:** Virtual Machines (VMs) don’t virtualize cores and may not support more than one core

## Project-Level Parallelism

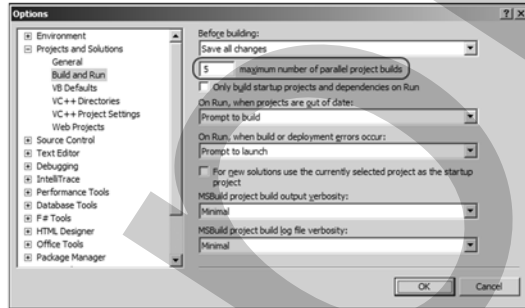
- Allows multiple projects to be built in parallel, while adhering to project dependency restrictions
  - MSBuild supports and controls project-level parallelism
- You set the max number of MSBuild Nodes that are used to execute the targets
  - This can be set through the Visual Studio IDE or the MSBuild command line
  - MSBuild nodes may be created or recycled (they generally linger for ~15 minutes after finishing a build assigned to it)

## Enabling Project-Level Parallelism

- Use the MSBuild.exe /m (or /maxcpucount) switch
  - Example: msbuild.exe /m:4 Foo.sln
  - If you omit this switch, a parallel project build is not performed (only one core will be used)
  - If you include the switch, but fail to specify a number, it will default to the number of cores on the machine
  - If there are any project dependencies, then you may get an effective concurrency less than what you specified
- Specify the setting in Visual Studio

## Project-Level Parallelism in Visual Studio

- Set the maximum number of parallel project builds under Tools > Options
  - You can specify an integer between 1 and 32 inclusive
  - It defaults to the # of cores on your computer
  - You can set it higher than the # of cores on your computer
  - This settings applies to all solutions and projects, not just the current sln/project



## File-Level Parallelism

- Allows groups of source files to be compiled in parallel in a project
  - It is up to the individual task authors to support this

## Enabling File-Level Parallelism

- Use the /MP switch on the CL (C++ compile) task
  - This causes the compiler to create one or more copies of itself, each in a separate process
  - These copies simultaneously compile the source files thus substantially reducing the total time to compile the source
- Experiment!
  - The improvement in build time depends on the number of cores, the number of files to compile, and the availability of system resources (such as I/O capacity)
  - Experiment with the /MP option to determine the best setting to build a particular project
  - See <http://bit.ly/jXj5Yx> for more tips

## File-Level Parallelism Examples

```
cl /MP7 a.cpp b.cpp c.cpp d.cpp e.cpp
```

- The compiler uses five processes because that is the lesser of five source files and a maximum of seven processes

```
cl /MP a.cpp b.cpp c.cpp
```

- The operating system reports two cores so the compiler uses two processes in its calculation
- As a result, the compiler will execute the build with two processes because that is the lesser of two processes and three source files

## File-Level Parallelism in MSBuild

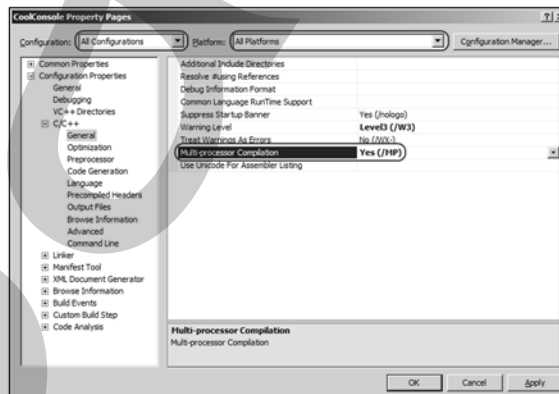
- Use the MultiProcessorCompilation and CL\_MPCount switches

```
Msbuild /p:MultiProcessorCompilation=true;  
CL_MPCount=3 MyProject.vcxproj
```

- Make sure to choose all Configurations and Platforms

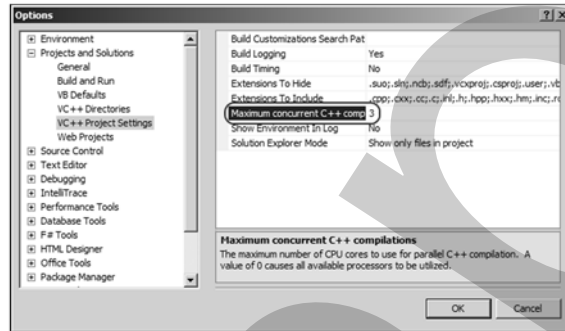
## File-Level Parallelism in Visual Studio

- First, enable multi-processor compilation support on the C++ project
  - Make sure to choose all Configurations and Platforms



## File-Level Parallelism in Visual Studio

- Second, set exactly how many simultaneous compilations should happen in Tools > Options.
  - Set any non-negative value
  - A value of 0 is equivalent to the # of cores on the machine



## Optimizing Build Parallelism

- Understand your project and file dependencies
- Analyze any bottlenecks that you discover
  - The /detailedsummary switch provides helpful information

## Incremental Builds

- An incremental build is one where you reuse the results of the preceding build while processing only the changes made to the project since that build
  - If the source files weren't changed since the last build, then no processing will occur
- This saves time!

## Incremental Builds: How Does it Work?

- MSBuild uses timestamps to only run those tasks whose inputs have changed since the last build

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

- MSBuild 4.0 introduces the File Tracker to track file accesses made by a process
  - It essentially eavesdrops on all file activity so that incremental builds are based on accurate information

## Build, Rebuild, and Clean



- **Build** compiles only those project files and components that have changed since the last build
  - A.k.a. an Incremental Build
- **Rebuild** builds all project files and components from scratch, regardless of whether they've changed or not
  - Rebuild simply calls "Clean" + "Build"
- **Clean** deletes any intermediate and output files
  - This forces the next build to be a full (non-incremental) build

## Trust MSBuild 4.0 Incremental Builds

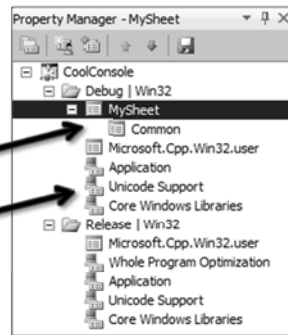
- Incremental builds in Visual C++ 2010 are reliable!
- The File Tracker infrastructure is much better than the dependency checker infrastructure of previous versions
  - File Tracker captures input and output information by automatically working behind the scenes
- Incremental builds in Visual C++ 2010 are faster too because MSBuild checks timestamps in parallel

## Property Sheets

- Property sheets allow you to share settings between projects
  - It is analogous to using header files to share type declarations among multiple class files
- Property sheets have a .props extension
  - They can contain any valid MSBuild elements, although they typically only contain settings (properties and items) and references to other property sheets
- Project files or property sheets that import a property sheet are said to “inherit” the settings

## Creating and Managing Property Sheets

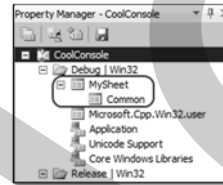
- Use the Property Manager window
  - View menu > Other Windows > Property Manager
- Sheets have different icons
  - User property sheets
  - System property sheets
- You can add, remove, and reorder user-added sheets
  - Note: the display order is the reverse of the textual order
- Double-click a property sheet to view/edit the pages
  - Note: System property sheets are read-only



## Examining the MSBuild Files

- The .vcxproj file imports MySheet.props

```
...
<ImportGroup Label="PropertySheets"
Condition="'$(Configuration)|$(Platform)'=='Debug|Win32'">
  <Import Project="$(UserRootDir)\Microsoft.Cpp.$(Platform).user.props"
Condition="exists('$(UserRootDir)\Microsoft.Cpp.$(Platform).user.props')"
Label="LocalAppDataPlatform" />
  <Import Project="MySheet.props" />
</ImportGroup>
...
```

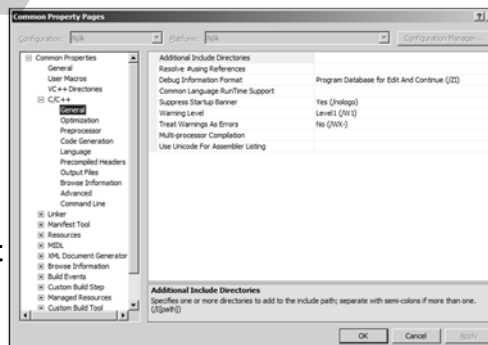


- The MySheet.props file imports Common.props

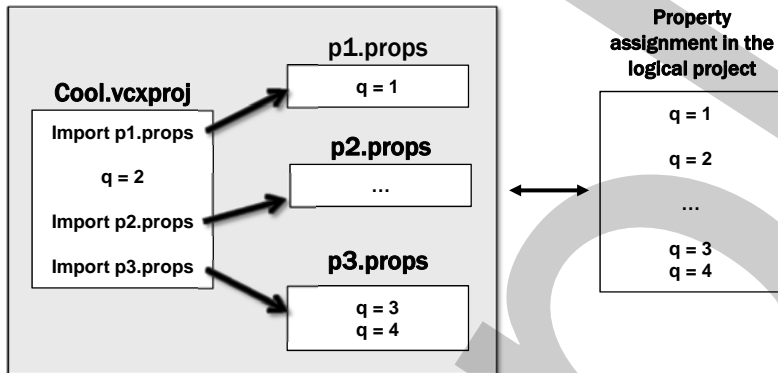
```
...
<ImportGroup Label="PropertySheets">
  <Import Project="Common.props" />
</ImportGroup>
...
```

## Property Pages

- Property pages are the primary way you change a project's properties
  - They are not new, but the underlying infrastructure is
- In Visual C++ 2010 you can easily add your own property pages
- Visual Studio allows you to view/edit the properties
  - In Solution Explorer, by right-clicking the project
  - In Property Manager



## Project Evaluation of Properties



- **Note:** if you specified `/p:q=42` on the command line, it would remain 42 throughout the evaluation

## Project Evaluation of Properties: Quiz

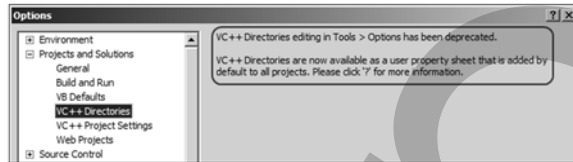
- What is the value of B?

```
<PropertyGroup>
  <A>1</A>
  <B>$(A)</B>
  <A>2</A>
</PropertyGroup>
```

- What would it have been in VCBUILD?

## Visual C++ Directories

- The IDE equivalents of the command-line variables:
  - PATH, INCLUDE, LIBPATH, etc.
  - Typically you point these to the various SDK components
- These were stored in Tools > Options in VC++ 2008



- Now they are stored in a property page

## Why did Microsoft Change This?

- You can now define different Visual C++ directory settings per project
  - Tools > Options applies to all projects
- Settings are now project files and can be checked-in
  - Your buddy (or your build server) can now have the same settings; builds shouldn't break or require additional edits
- Settings are stored consistently
  - Prior to Visual C++ 2010, VCComponents.dat was an INI-based file; now they are XML-based MSBuild files

## **Summary**

- Visual C++ 2010 projects are now MSBuild files
- Earlier project versions can be easily migrated
- The build process is very robust and is extensible
- Parallel builds can be configured at the project and file level
- Incremental builds save time
- Property sheets and page architecture has changed and now leverage the MSBuild file format

## **Lab**

In this lab you will setup the learning environment and get familiar with the basics of MSBuild

- Explore VC++ 2010 Project File
- Convert a VC++ 2008 Project File to VC++ 2010
- Configure and run an incremental build
- Configure and run parallel builds (optional)

# **Accentient**<sup>TM</sup>



## **Lab 2: Visual C++ Support**

### **Leveraging MSBuild 4.0**

## **Estimated time to complete this lab: 90 minutes**

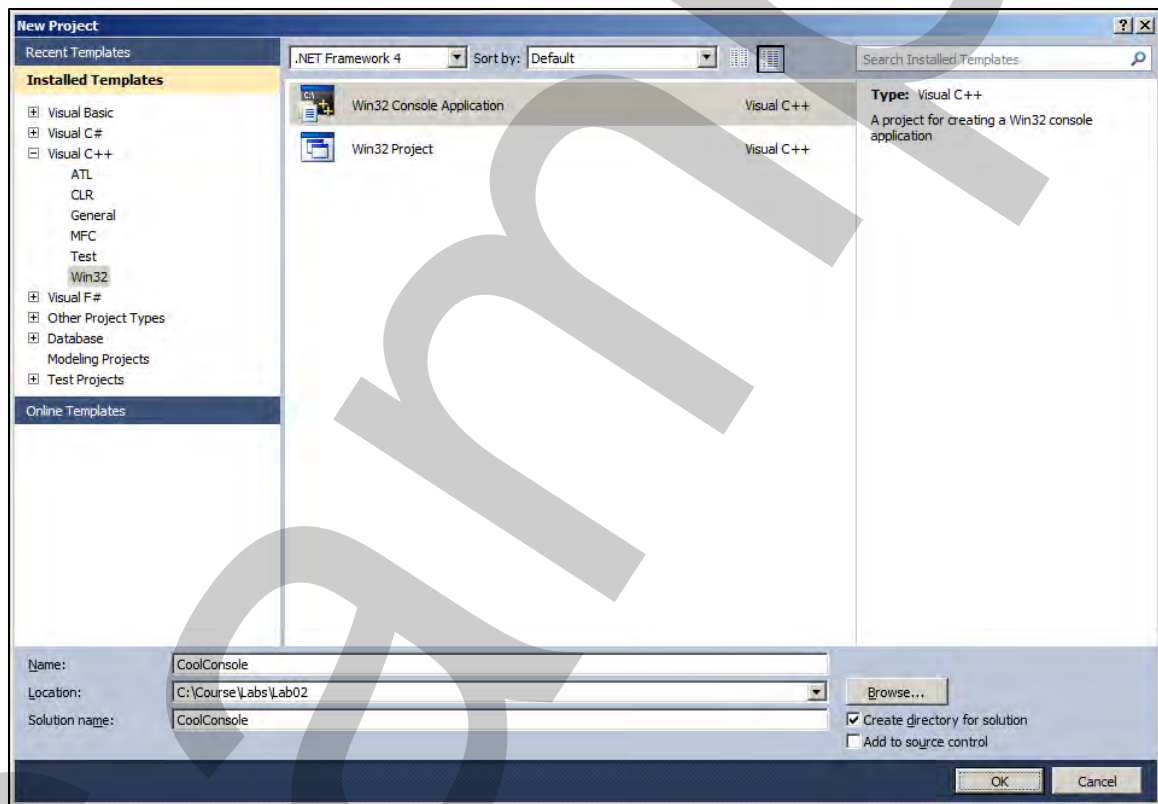
Beginning with Visual Studio 2010, MSBuild is the standard build system for Visual C++ projects. When you build a project in the Visual Studio integrated development environment (IDE), it uses the msbuild.exe tool, an XML-based project file, and optional settings files. Although you can use msbuild.exe and a project file on the command line, the IDE provides a user interface so that you can more easily configure settings and build a project.

# EXERCISE 1 – EXPLORE A .VCXPROJ PROJECT FILE

## TASK – EXPLORE A .VCXPROJ PROJECT FILE

In this task, you will log on to Windows as Student, create and explore a Visual C++ Console Application.

1. Log on to Windows as **Student** with the password **password**.
2. Launch **Visual Studio 2010**.
3. From the **File** menu, select **New > Project**.
4. Select a **Visual C++ > Win32 > Win32 Console Application** project template and name it **CoolConsole** choosing **C:\Course\Labs\Lab02** as the location.



5. Click **OK**.
6. Click **Finish** on the **Win32 Application Wizard**.
7. Close the code editor window.

- In **Solution Explorer**, right-click on the **CoolConsole** project and select **Unload Project**.
- Right-click on the **CoolConsole** project and select **Edit CoolConsole.vcxproj**.

Are you looking at an MSBuild file? \_\_\_\_\_

You can tell because of the XML namespace (xmlns) listed:

```
<?xml version="1.0" encoding="utf-8"?>
<Project DefaultTargets="Build" ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <ItemGroup Label="ProjectConfigurations">
    <ProjectConfiguration Include="Debug|Win32">
```

You can also see the Properties, ItemGroups, and Imports statements.

- Close the code editor window.

Don't save any changes, if you happened to make any.

- In **Solution Explorer**, right-click on the **CoolConsole** project and select **Reload Project**.
- Exit **Visual Studio**.

## TASK – USE MSBUILD TO BUILD COOLCONSOLE

In this task, you will use the MSBuild.exe command-line utility to build the CoolConsole Visual C++ project.

- Using **Windows Explorer**, navigate to **C:\Course\Labs\Lab02**.
- Right-click on **MSBuild-CoolConsoleSln.bat** and select **Edit**.

As you can see we are just passing in the path to the .sln (solution) file to MSBuild and asking it to build it.

- Close **Notepad**.

4. Double-click **MSBuild-CoolConsoleSln.bat** to execute it.

It should only take a couple of seconds to run. Please review the output in the command window. It should look something like this:

```
Build started 6/10/2011 11:27:56 AM.
Project "C:\Course\Labs\Lab02\CoolConsole\CoolConsole.sln" on node 1 (default targets).
ValidateSolutionConfiguration:
  Building solution configuration "Debug!Win32".
Project "C:\Course\Labs\Lab02\CoolConsole\CoolConsole.sln" (1) is building "C:\Course\Labs\Lab02\CoolConsole\CoolConsole\CoolConsole.vcxproj" (2) on node 1 (default targets).
PrepareForBuild:
  Creating directory "Debug\".
  Creating directory "C:\Course\Labs\Lab02\CoolConsole\Debug\".
InitializeBuildStatus:
  Creating "Debug\CoolConsole.unsuccessfulbuild" because "AlwaysCreate" was specified.
ClCompile:
  C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\bin\CL.exe /c /ZI /nologo /W3 /WX- /Od /Oy- /D WIN32 /D _DEBUG /D _CONSOLE /D UNICODE /D UNICODE /Gm /EHsc /RTC1 /MDd /GS /fp:precise /Zc:wchar_t /Zc:forScope /Yc"Stdafx.h" /Fp"Debug\CoolConsole.pch" /Fo"Debug\\" /Fd"Debug\vc100.pdb" /Gd /TP /analyze- /errorReport:queue stdafx.cpp
  C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\bin\CL.exe /c /ZI /nologo /W3 /WX- /Od /Oy- /D WIN32 /D _DEBUG /D _CONSOLE /D UNICODE /D UNICODE /Gm /EHsc /RTC1 /MDd /GS /fp:precise /Zc:wchar_t /Zc:forScope /Yu"Stdafx.h" /Fp"Debug\CoolConsole.pch" /Fo"Debug\\" /Fd"Debug\vc100.pdb" /Gd /TP /analyze- /errorReport:queue CoolConsole.cpp
ManifestResourceCompile:
  C:\Program Files (x86)\Microsoft SDKs\Windows\v7.0A\bin\rc.exe /nologo /fo"Debug\CoolConsole.exe.embed.manifest.res" Debug\CoolConsole_manifest.rc
Link:
  C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\bin\link.exe /ERRORREPORT:QUEUE /OUT:"C:\Course\Labs\Lab02\CoolConsole\Debug\CoolConsole.exe" /INCREMENTAL /NOLOGO kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib aadvapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib /MANIFEST /ManifestFile:"Debug\CoolConsole.exe.intermediate.manifest" /MANIFESTUAC:"level='asInvoker' uiAccess='false'" /DEBUG /PDB:"C:\Course\Labs\Lab02\CoolConsole\Debug\CoolConsole.pdb" /SUBSYSTEM:CONSOLE /TLBID:1 /DYNAMICBASE /NXCOMPAT /IMPLIB:"C:\Course\Labs\Lab02\CoolConsole\Debug\CoolConsole.lib" /MACHINE:X86 Debug\CoolConsole.exe.embed.manifest.res
  Debug\CoolConsole.obj
  Debug\stdafx.obj
Manifest:
  C:\Program Files (x86)\Microsoft SDKs\Windows\v7.0A\bin\mt.exe /nologo /verbose /out:"Debug\CoolConsole.exe.embed.manifest" /manifest Debug\CoolConsole.exe.intermediate.manifest
  C:\Program Files (x86)\Microsoft SDKs\Windows\v7.0A\bin\rc.exe /nologo /fo"Debug\CoolConsole.exe.embed.manifest.res" Debug\CoolConsole_manifest.rc
LinkEmbedManifest:
  C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\bin\link.exe /ERRORREPORT:QUEUE /OUT:"C:\Course\Labs\Lab02\CoolConsole\Debug\CoolConsole.exe" /INCREMENTAL /NOLOGO kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib aadvapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib /MANIFEST /ManifestFile:"Debug\CoolConsole.exe.intermediate.manifest" /MANIFESTUAC:"level='asInvoker' uiAccess='false'" /DEBUG /PDB:"C:\Course\Labs\Lab02\CoolConsole\Debug\CoolConsole.pdb" /SUBSYSTEM:CONSOLE /TLBID:1 /DYNAMICBASE /NXCOMPAT /IMPLIB:"C:\Course\Labs\Lab02\CoolConsole\Debug\CoolConsole.lib" /MACHINE:X86 Debug\CoolConsole.exe.embed.manifest.res
  Debug\CoolConsole.obj
  Debug\stdafx.obj
  CoolConsole.vcxproj -> C:\Course\Labs\Lab02\CoolConsole\Debug\CoolConsole.exe
FinalizeBuildStatus:
  Deleting file "Debug\CoolConsole.unsuccessfulbuild".
  Touching "Debug\CoolConsole.lastbuildstate".
Done Building Project "C:\Course\Labs\Lab02\CoolConsole\CoolConsole\CoolConsole.vcxproj" (default targets).
Done Building Project "C:\Course\Labs\Lab02\CoolConsole\CoolConsole.sln" (default targets).

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

Time Elapsed 00:00:03.24
Press any key to continue . . .
```

5. Press any key to close the command window.

- Using **Windows Explorer**, navigate to **C:\Course\Labs\Lab02\CoolConsole**.

Do you see the Debug subfolder? \_\_\_\_\_

This folder contains the incremental linker file, application, and program debug database files that were created as part of the build process.

- Navigate to **C:\Course\Labs\Lab02\CoolConsole\CoolConsole\Debug**.

This folder contains all of the working files that were used to compile the application, including manifest, object, header, resource, and .TLOG files.

- Navigate back to **C:\Course\Labs\Lab02**.

- Right-click on **MSBuild-CoolConsoleInClean.bat** and select **Edit**.

This will run the previous build but only the *clean* target. This will remove the files from the aforementioned Debug folders.

- Close **Notepad**.

- Double-click **MSBuild-CoolConsoleInClean.bat** to execute it.

It should only take a couple of seconds to run. Please review the output.

- Press any key to close the command window.

- Using **Windows Explorer**, navigate back to **C:\Course\Labs\Lab02\CoolConsole\CoolConsole\Debug**.

You should see that all the files are gone, with the exception of a log file that contains a list of files that were cleaned.

The Clean target does just that. It smartly removes the files that were generated from a previous Build. In fact, the difference between a Build and a Rebuild is that Rebuild runs a Clean first. That's all.

## EXERCISE 2 – CONVERT A VISUAL C++ 2008 PROJECT

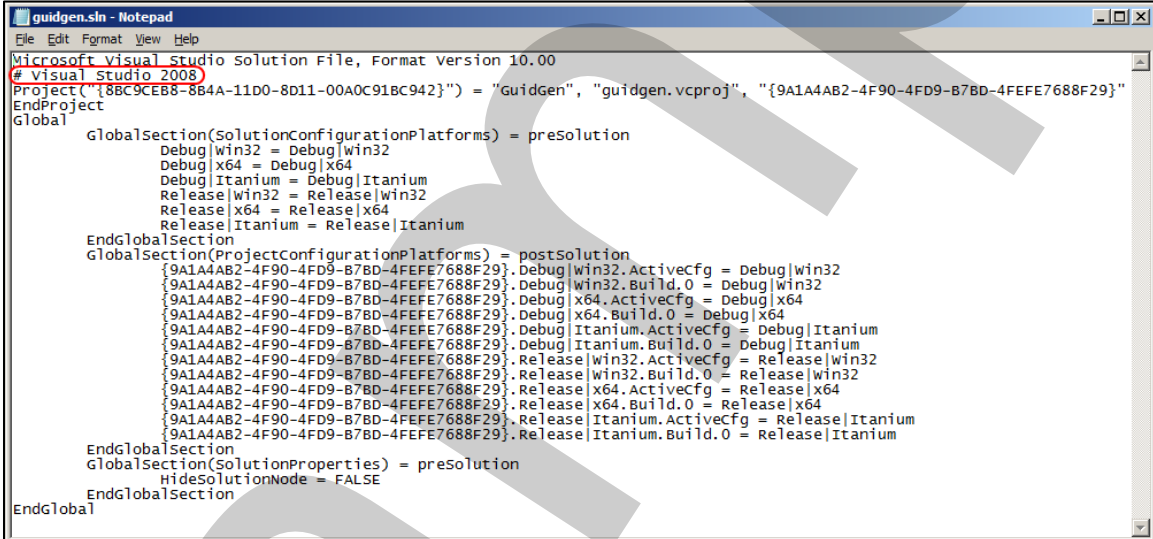
### TASK – CONVERT A VISUAL C++ 2008 PROJECT

In this task, you will open a Visual C++ 2008 project in Visual Studio 2010, launch the conversion wizard, and review the results of the conversion.

1. Using **Windows Explorer**, navigate to **C:\Course\Labs\Lab02\guidgen**.
2. Right-click on the **guidgen.sln** file and select **Open with > Notepad**.

If notepad is not an available option, you may have to select “choose default program” and select Notepad. Just make sure to clear the “Always use the selected program to open this kind of file” option or else double-clicking a .sln file won’t launch Visual Studio anymore!

As you can see the solution’s version is 2008:



```
guidgen.sln - Notepad
File Edit Format View Help
Microsoft Visual Studio solution File, Format version 10.00
# Visual Studio 2008
Project("{8BC9CEB8-8B4A-11D0-8D11-00A0C91BC942}") = "guidgen", "guidgen.vcproj", "{9A1A4AB2-4F90-4FD9-B7BD-4FEFE7688F29}"
EndProject
Global
GlobalSection(SolutionConfigurationPlatforms) = presolution
  Debug|win32 = Debug|win32
  Debug|x64 = Debug|x64
  Debug|Itanium = Debug|Itanium
  Release|win32 = Release|win32
  Release|x64 = Release|x64
  Release|Itanium = Release|Itanium
EndGlobalSection
GlobalSection(ProjectConfigurationPlatforms) = postSolution
  {9A1A4AB2-4F90-4FD9-B7BD-4FEFE7688F29}.Debug|win32.ActiveCfg = Debug|win32
  {9A1A4AB2-4F90-4FD9-B7BD-4FEFE7688F29}.Debug|win32.Build.0 = Debug|win32
  {9A1A4AB2-4F90-4FD9-B7BD-4FEFE7688F29}.Debug|x64.ActiveCfg = Debug|x64
  {9A1A4AB2-4F90-4FD9-B7BD-4FEFE7688F29}.Debug|x64.Build.0 = Debug|x64
  {9A1A4AB2-4F90-4FD9-B7BD-4FEFE7688F29}.Debug|Itanium.ActiveCfg = Debug|Itanium
  {9A1A4AB2-4F90-4FD9-B7BD-4FEFE7688F29}.Debug|Itanium.Build.0 = Debug|Itanium
  {9A1A4AB2-4F90-4FD9-B7BD-4FEFE7688F29}.Release|win32.ActiveCfg = Release|win32
  {9A1A4AB2-4F90-4FD9-B7BD-4FEFE7688F29}.Release|win32.Build.0 = Release|win32
  {9A1A4AB2-4F90-4FD9-B7BD-4FEFE7688F29}.Release|x64.ActiveCfg = Release|x64
  {9A1A4AB2-4F90-4FD9-B7BD-4FEFE7688F29}.Release|x64.Build.0 = Release|x64
  {9A1A4AB2-4F90-4FD9-B7BD-4FEFE7688F29}.Release|Itanium.ActiveCfg = Release|Itanium
  {9A1A4AB2-4F90-4FD9-B7BD-4FEFE7688F29}.Release|Itanium.Build.0 = Release|Itanium
EndGlobalSection
GlobalSection(SolutionProperties) = presolution
  HideSolutionNode = FALSE
EndGlobalSection
EndGlobal
```

You can also tell by the icon you see listed in Windows Explorer. It will have a small “9” in the icon () and opposed to a Visual Studio 2010 solution (.

3. Close **Notepad**.
4. Right-click on the **guidgen.vcproj** file and select **Open with > Notepad**.

Is this file in XML format? \_\_\_\_\_

Is this an MSBuild project file? \_\_\_\_\_

How can you tell? \_\_\_\_\_



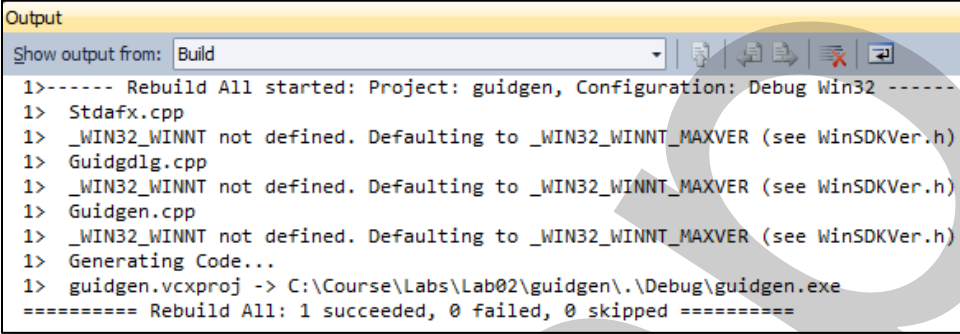
11. Close the conversion report window.

You can always reopen the UpgradeLog.XML file to return to this report.

12. From the **Build** menu select **Rebuild Solution**.

Were there any errors or warnings? \_\_\_\_\_

The small solution and project should have built without errors or warnings.



```
Output
Show output from: Build
1>----- Rebuild All started: Project: guidgen, Configuration: Debug Win32 -----
1> Stdafx.cpp
1> _WIN32_WINNT not defined. Defaulting to _WIN32_WINNT_MAXVER (see WinSDKVer.h)
1> Guiddlg.cpp
1> _WIN32_WINNT not defined. Defaulting to _WIN32_WINNT_MAXVER (see WinSDKVer.h)
1> Guidgen.cpp
1> _WIN32_WINNT not defined. Defaulting to _WIN32_WINNT_MAXVER (see WinSDKVer.h)
1> Generating Code...
1> guidgen.vcxproj -> C:\Course\Labs\Lab02\guidgen\.\Debug\guidgen.exe
===== Rebuild All: 1 succeeded, 0 failed, 0 skipped =====
```

13. Press **F5** to run the guidgen application.

You can see the GUID being generated at the bottom of the window. Feel free to play with the different formats and generate new GUIDs.

14. Click **Exit**.

15. In **Solution Explorer**, right-click on the **guidgen** project and select **Unload Project**.

16. Right-click on the **guidgen** project and select **Edit guidgen.vcxproj**.

As you can see, the conversion wizard created a proper MSBuild project file. Spend a few moments scrolling down and reviewing the contents of this file.

Do any of the properties have blue underlines? \_\_\_\_\_

What does this mean? (hint: hover over one) \_\_\_\_\_

17. Close the code editor window.

**Don't save any changes, if you happened to make any.**

18. Reload the project in **Solution Explorer**.

## TASK – USE PROPERTY MANAGER

In this task, you will use the Property Manager window to view system and user property sheets and make a change to a user property sheet.

1. From the **View** menu select **Other Windows > Property Manager**.
2. Expand the **guidgen** root node in the **Property Manager** window.

This solution has defined a “debug” and a “release” configuration, which is pretty standard.

What platforms have been defined? \_\_\_\_\_

3. Expand the **Debug | Win32** platform folder.

What system property sheets are defined? \_\_\_\_\_

\_\_\_\_\_

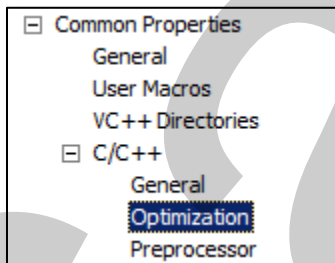
4. Double-click the **Application** system property sheet.

This opens the relevant property page.

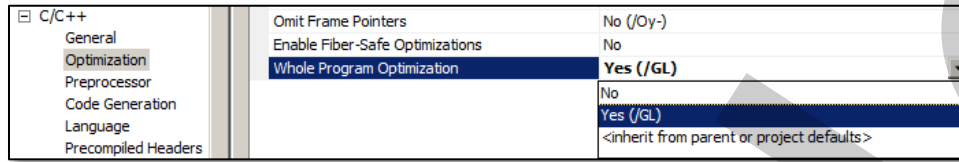
Are you able to make changes? \_\_\_\_\_

System property sheets are read-only.

5. Click **Cancel**.
6. Double-click the **Microsoft.Cpp.Win32.user** user property sheet.
7. Select the **Common Properties > C/C++ > Optimization** property page.



8. Change the **Whole Program Optimization** property to **Yes (/GL)**.



This is not a setting that is typically enabled in the Debug configuration.

9. Click **OK**.
10. From the **File** menu select **Save All**.

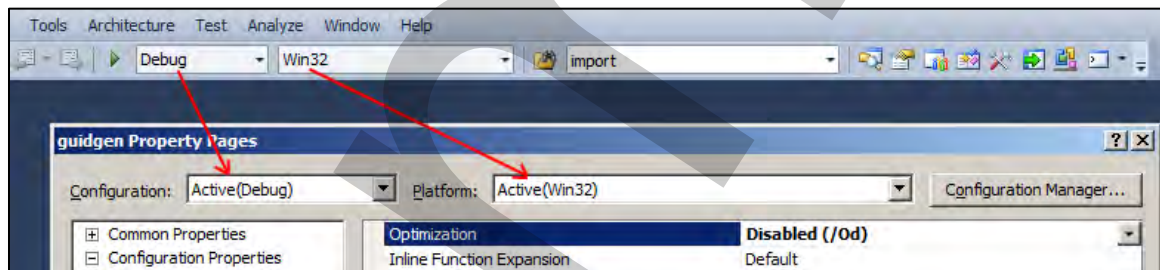
This ensures that all the changes to the .props files have been saved.

11. In **Solution Explorer**, right-click on the **guidgen** project and select **Properties**.

What Configuration is listed at the top of the dialog window? \_\_\_\_\_

What Platform is listed at the top of the dialog window? \_\_\_\_\_

These defaults are based on what you have selected in your Visual Studio IDE.

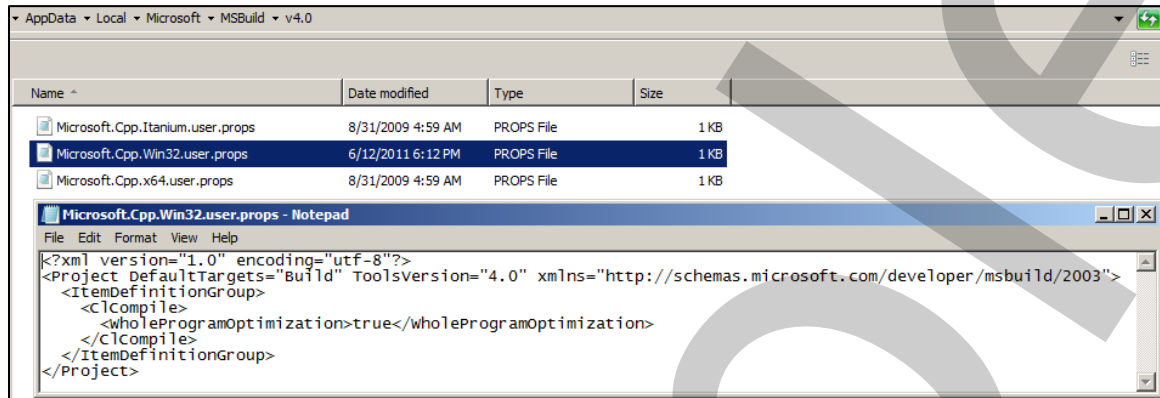


12. Select the **Configuration Properties > C/C++ > Optimization** property page.

Is it set to Yes (/GL)? \_\_\_\_\_

You might expect that if you reviewed the guidgen.vcxproj XML that you would find a `<WholeProgramOptimization>true</WholeProgramOptimization>` element in the `<CCompile>` section of Debug|Win32, but you'd be wrong.

What we did was change the Microsoft.Cpp.Win32.user.props file in your \AppData\Local\Microsoft\MSBuild\v4.0 folder:



This means that the next Visual C++ 2010 project you create will have the Whole Program Optimization already set for the Debug|Win32 configure.

13. Click **Cancel** twice.
14. Exit **Visual Studio**.

## EXERCISE 3 – LEVERAGE INCREMENTAL BUILDS

### TASK – CONFIGURE AND RUN AN INCREMENTAL BUILD

In this task, you will open a Visual C++ 2010 solution and perform build, rebuild, and clean operations on it.

1. Using **Windows Explorer**, navigate to **C:\Course\Labs\Lab02\Scribble**.
2. Double-click the **Scribble.sln** solution file to open it.

This is a simple MFC program that allows you to scribble on documents, much like a combination of notepad and paint. It illustrates a wide breadth of MFC features. It may take a few moments for the solution to load the first time. You can monitor the progress on the status bar at the bottom.

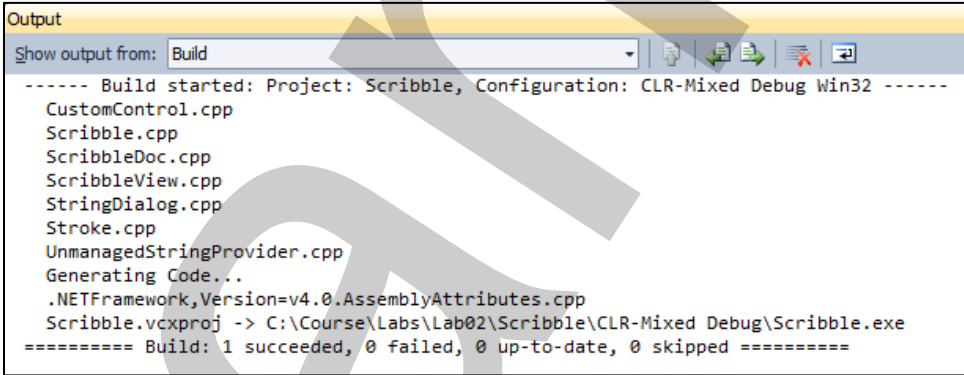
3. From the **View** menu select **Output**.

This ensures the Output window is visible to show us information about the build we are about to run.

4. From the **Build** menu select **Build Solution**.

It should only take a few seconds to build this project.

How many .cpp files are listed in the Output window? \_\_\_\_\_



```
Output
Show output from: Build
----- Build started: Project: Scribble, Configuration: CLR-Mixed Debug Win32 -----
CustomControl.cpp
Scribble.cpp
ScribbleDoc.cpp
ScribbleView.cpp
StringDialog.cpp
Stroke.cpp
UnmanagedStringProvider.cpp
Generating Code...
.NETFramework,Version=v4.0.AssemblyAttributes.cpp
Scribble.vcxproj -> C:\Course\Labs\Lab02\Scribble\CLR-Mixed Debug\Scribble.exe
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
```

The first time the project is built, all files must be compiled.

5. Press **F5** to run Scribble.

It's a very simple app allowing you to scribble on one or more documents.

6. Exit **Scribble**.

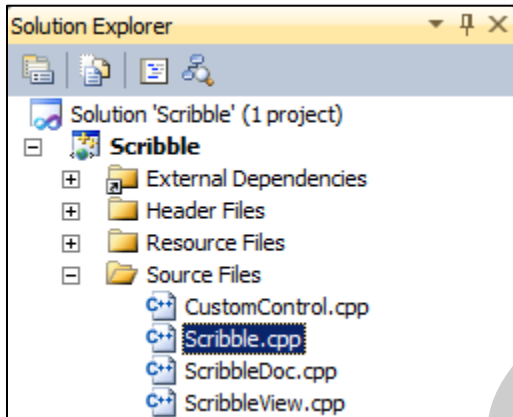
7. From the **Build** menu select **Build Solution**.

What is the message in the Output window? \_\_\_\_\_

No files were touched since the last build, so no compilation was necessary.

8. In **Solution Explorer**, locate and double-click on **Scribble.cpp**.

It should be found under the Source Files folder.



9. In the void **MainWindow::InitializeComponent()** method, make this change:

Change the Text to "Scrubble". This simply changes the title of the window when it runs.

```
void MainWindow::InitializeComponent()
{
    components = gcnew System::ComponentModel::Container;
    printDoc = gcnew System::Drawing::Printing::PrintDocument;
    printDoc->PrintPage += gcnew PrintPageEventHandler( this, &MainWindow::ScribblePrintPage );
    this->AutoScaleBaseSize = System::Drawing::Size( 5, 13 );
    this->Text = "Scrubble";
    this->IsMdiContainer = true;
    this->Menu = mainMenu;
    this->ClientSize = System::Drawing::Size( 600, 400 );
}
```

10. Save your changes.

11. Close the code editor window.

12. From the **Build** menu select **Build Solution**.

What source code file was listed in the Output window? \_\_\_\_\_

Only the one file was touched, so it was the only one that needed to be compiled.

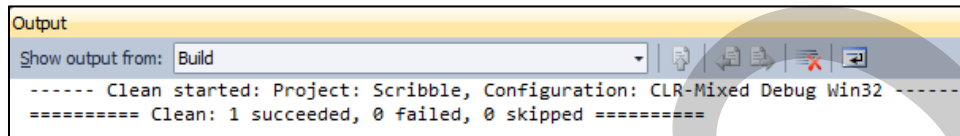
13. Using **Windows Explorer**, navigate to **C:\Course\Labs\Lab02\Scribble\CLR-Mixed Debug**.

How many files are listed in this folder? \_\_\_\_\_

14. Minimize **Windows Explorer**.

15. Return to **Visual Studio**.

16. From the **Build** menu select **Clean Solution**.



17. Return to **Windows Explorer**.

Are most of the files gone? \_\_\_\_\_

What file(s) remain? \_\_\_\_\_

18. Return to **Visual Studio**.

19. From the **Build** menu select **Build Solution**.

This build should be like the first one you did in this exercise. Since no output files exist in the \CLR-Mixed Debug folder, Visual Studio had to do a full build.

20. From the **Build** menu select **Rebuild Solution**.

A rebuild performs a *Clean* and a *Build*, so the net effect is just like the previous few steps.

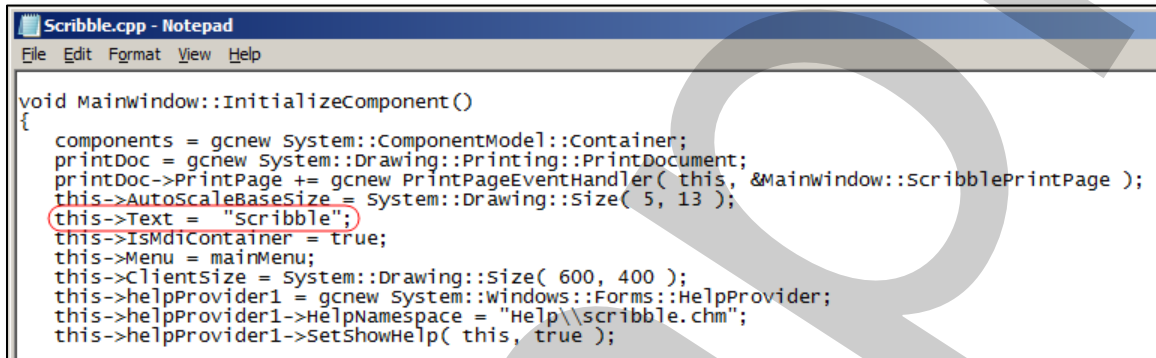
21. Close all code editor windows.

22. Minimize **Visual Studio**.

## TASK – SEE THE FILE TRACKER IN ACTION

In this task, you will make a change to a file outside of Visual Studio and see that incremental builds still occur.

1. Return to **Windows Explorer** and navigate to **C:\Course\Labs\Lab02\Scribble**.
2. Right-click on **Scribble.cpp** and select **Open With > Notepad**.
3. Change the text back to **Scribble**.

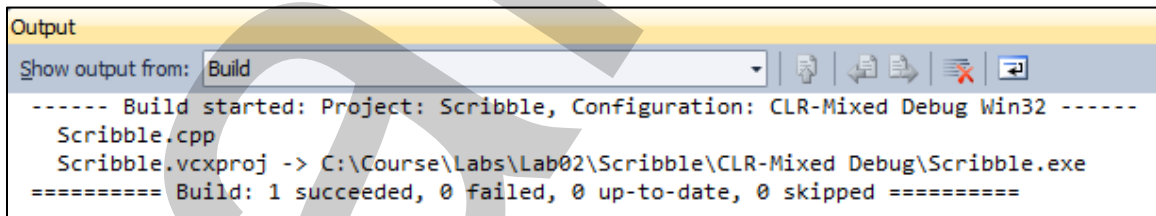


```
Scribble.cpp - Notepad
File Edit Format View Help

void MainWindow::InitializeComponent()
{
    components = gcnew System::ComponentModel::Container;
    printDoc = gcnew System::Drawing::Printing::PrintDocument;
    printDoc->PrintPage += gcnew PrintPageEventHandler( this, &MainWindow::ScribblePrintPage );
    this->AutoScaleBaseSize = System::Drawing::Size( 5, 13 );
    this->Text = "Scribble";
    this->IsMdiContainer = true;
    this->Menu = mainMenu;
    this->ClientSize = System::Drawing::Size( 600, 400 );
    this->helpProvider1 = gcnew System::Windows::Forms::HelpProvider;
    this->helpProvider1->HelpNamespace = "Help\\scribble.chm";
    this->helpProvider1->SetShowHelp( this, true );
}
```

4. Save your changes.
5. Exit **Notepad**.
6. Return to **Visual Studio**.
7. From the **Build** menu select **Build Solution**.

What source code file was listed in the Output window? \_\_\_\_\_



```
Output
Show output from: Build
----- Build started: Project: Scribble, Configuration: CLR-Mixed Debug Win32 -----
Scribble.cpp
Scribble.vcxproj -> C:\Course\Labs\Lab02\Scribble\CLR-Mixed Debug\Scribble.exe
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
```

You can't outsmart the File Tracker!

8. Exit **Visual Studio**.

## EXERCISE 4 – PROJECT-LEVEL BUILD PARALLELISM (OPTIONAL)

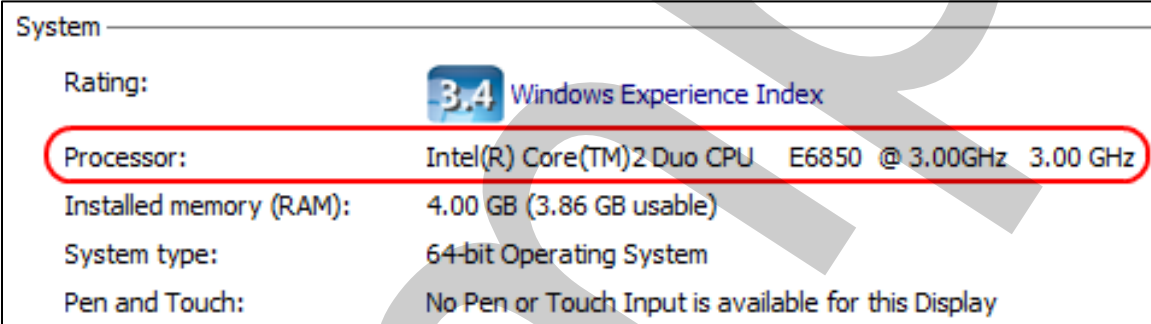
Note: If you are running these labs in a virtual machine, on a hosted environment, or on older hardware, you may not have multiple cores and these next two exercises won't be that interesting.

### TASK – CONFIGURE AND RUN A PROJECT-LEVEL PARALLEL BUILD

In this task, you will open a Visual C++ 2010 solution containing several projects, review it, and then build it using project-level build parallelism.

1. From the **Start** menu, right click on **Computer** and select **Properties**.

This page shows you, among other things, the type and number of cores/processors your computer has. This information is in the System section.



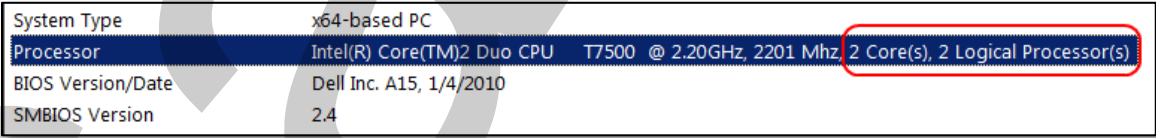
The screenshot shows the Windows System window with the following information:

Rating:	3.4 Windows Experience Index
Processor:	Intel(R) Core(TM)2 Duo CPU E6850 @ 3.00GHz 3.00 GHz
Installed memory (RAM):	4.00 GB (3.86 GB usable)
System type:	64-bit Operating System
Pen and Touch:	No Pen or Touch Input is available for this Display

What is your Processor? \_\_\_\_\_

How many cores/processors do you have? \_\_\_\_\_

Note: If you are running in a virtual machine, the processor information posted may not match your physical hardware. You can also run msinfo32.exe for more definitive information:



The screenshot shows the msinfo32.exe window with the following information:

System Type	x64-based PC
Processor	Intel(R) Core(TM)2 Duo CPU T7500 @ 2.20GHz, 2201 Mhz 2 Core(s), 2 Logical Processor(s)
BIOS Version/Date	Dell Inc. A15, 1/4/2010
SMBIOS Version	2.4

2. Close the **System** window.
3. Using **Windows Explorer**, navigate to **C:\Course\Labs\Lab02\BigSolution**.

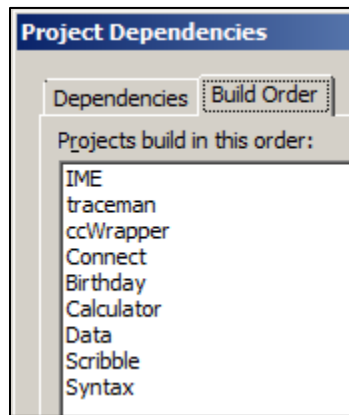
4. Double-click the **BigSolution.sln** solution file to open it.

This solution contains many unrelated projects. Each of these projects were found on MSDN's Visual C++ Samples for Visual Studio 2010 page: <http://bit.ly/lqJ84>.

It may take a few moments for the solution to load the first time. You can monitor the progress on the status bar at the bottom.

5. In **Solution Explorer**, right-click on **BigSolution** and select **Project Build Order**.

Notice the build order isn't necessarily alphabetically like they are listed in Solution Explorer:



6. Click the **Dependencies** tab.

Notice that there are no dependencies between any of the projects. Each of these projects could be built in parallel if we had enough processors!

7. Click **Cancel**.

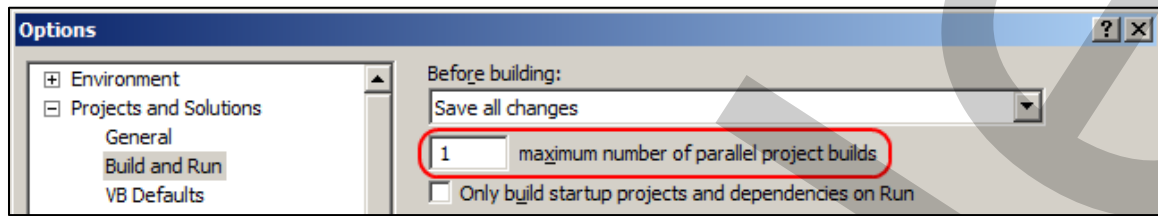
8. From the **Tools** menu select **Options**.

9. Expand the **Projects and Solutions > Build and Run** page.

What is your *maximum number of parallel project builds* set to? \_\_\_\_\_

Does this number match the number of cores/processors you have? \_\_\_\_\_

- Change the **maximum number of parallel project builds** value to **1**.



We are essentially turning *off* parallel project builds for the time being.

- Click **OK**.
- From the **View** menu select **Output**.

This ensures the Output window is visible to show us information about the build we are about to run.

- From the **Build** menu select **Rebuild Solution**.

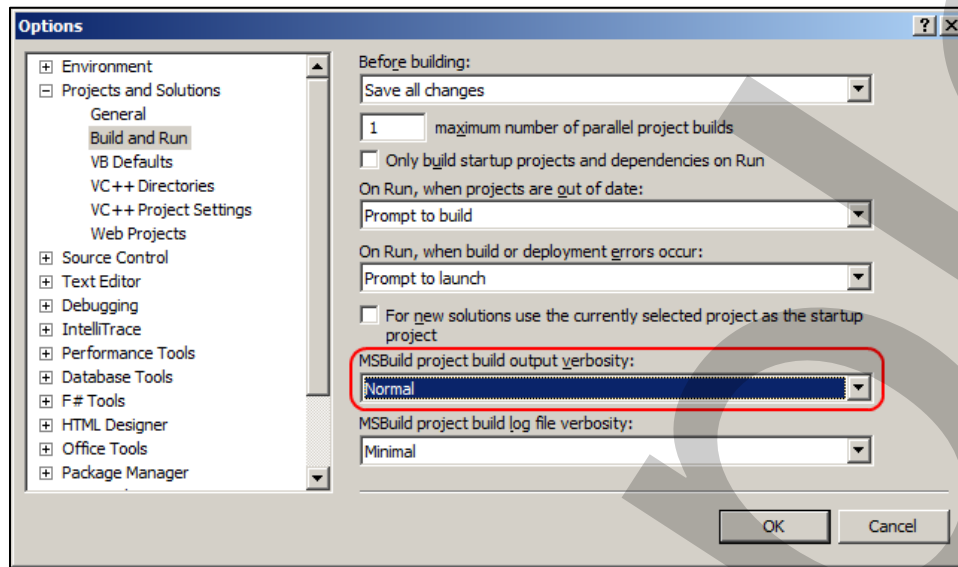
It will take a few moments for Visual Studio and MSBuild to execute a complete build of all the projects in the solution.

What was the overall elapsed time it took to build the solution? \_\_\_\_\_

Unfortunately the default level of MSBuild verbosity is *minimal* which doesn't include build times. We will need to change it to *normal* or higher and note the individual project build times.

- From the **Tools** menu select **Options**.
- Expand the **Projects and Solutions** > **Build and Run** page.

16. Change the **MSBuild project build output verbosity** value to **Normal**.



Notice that you can also set the verbosity level of the log file. This way you can see minimal information in the Output window, but detailed or diagnostic information in the log file for troubleshooting.

17. Click **OK**.

18. From the **Build** menu select **Rebuild Solution**.

Visual Studio doesn't have a way to show you the elapsed time to build the entire solution. This was discussed on stackoverflow (<http://bit.ly/ijWdCQ>), including a few possible workarounds, such as installing VSCommands (<http://bit.ly/hq6ZVk>).

I don't expect you to sum up all of the individual project times and, as it turns out, doing so won't give you an accurate measure of the length of time it took. For our next few tests, we'll adjourn to the command line where MSBuild.exe can help us gather this information a little easier.

19. From the **Tools** menu select **Options**.

20. Change the **maximum number of parallel project builds** value to **2** and click **OK**.

21. From the **Build** menu select **Rebuild Solution**.

Notice that as this build runs, the Output window precedes each line with a number. This shows which project build is echoing back to the Output window. As you can see below all the 1's are the IME project and all the 2's are the Traceman project. With multiple builds it can get really confusing in there.

```
1>----- Rebuild All started: Project: IME, Configuration: Debug Win32 -----
2>----- Rebuild All started: Project: traceman, Configuration: Debug Win32 -----
1>Build started 6/12/2011 8:44:20 PM.
1>_PrepareForClean:
1>  Deleting file ".\Debug\IME.lastbuildstate".
1>InitializeBuildStatus:
1>  Creating ".\Debug\IME.unsuccessfulbuild" because "AlwaysCreate" was specified.
1>ClCompile:
1>  StdAfx.cpp
2>Build started 6/12/2011 8:44:20 PM.
2>_PrepareForClean:
2>  Deleting file "Debug\traceman.lastbuildstate".
2>InitializeBuildStatus:
2>  Creating "Debug\traceman.unsuccessfulbuild" because "AlwaysCreate" was specified.
2>ClCompile:
2>  stdafx.cpp
1>  IME.cpp
2>  XMLReader.cpp
```

Here's a screenshot of running 9 parallel builds (one for each project):

```
1>----- Rebuild All started: Project: IME, Configuration: Debug Win32 -----
2>----- Rebuild All started: Project: traceman, Configuration: Debug Win32 -----
3>----- Rebuild All started: Project: ccWrapper, Configuration: Debug Win32 -----
4>----- Rebuild All started: Project: Connect, Configuration: Debug Win32 -----
5>----- Rebuild All started: Project: Birthday, Configuration: Debug Win32 -----
6>----- Rebuild All started: Project: Calculator, Configuration: Debug Win32 -----
7>----- Rebuild All started: Project: Data, Configuration: Debug Win32 -----
8>----- Rebuild All started: Project: Scribble, Configuration: Debug Win32 -----
9>----- Rebuild All started: Project: Syntax, Configuration: Debug Win32 -----
3>Build started 6/12/2011 8:58:51 PM.
1>Build started 6/12/2011 8:58:51 PM.
3>_PrepareForClean:
3>  Deleting file "Debug\ccWrapper.lastbuildstate".
1>_PrepareForClean:
1>  Deleting file ".\Debug\IME.lastbuildstate".
2>Build started 6/12/2011 8:58:51 PM.
2>_PrepareForClean:
2>  Deleting file "Debug\traceman.lastbuildstate".
4>Build started 6/12/2011 8:58:51 PM.
```

22. Exit **Visual Studio**.

### TASK – BENCHMARK PROJECT-LEVEL PARALLEL BUILD EXECUTION

In this task, you will use the MSBuild.exe command-line utility to build the BigSolution in various ways, seeing if parallelism actually results in faster builds.

1. Using **Windows Explorer**, navigate to **C:\Course\Labs\Lab02**.
2. Right-click on **MSBuild-BigSolution1Node.bat** and select **Edit**.

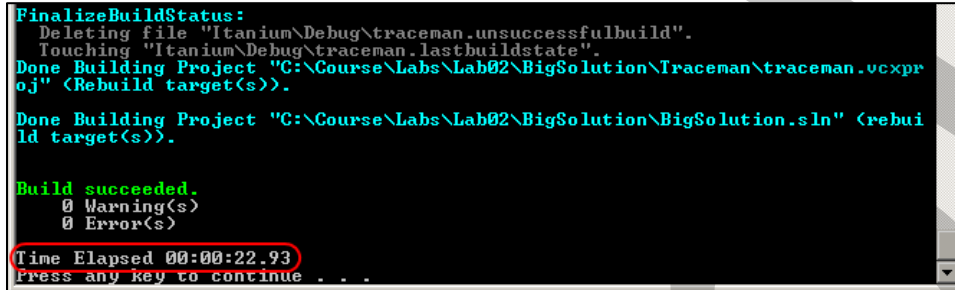
This MSBuild file simply does a rebuild of the BigSolution.sln without using any project-level parallelism. Remember the setting we made in Visual Studio was in Tools > **Options** so it didn't "stick" to this solution. We will have to tell MSBuild to use parallelism explicitly.

3. Close **Notepad**.

4. Double-click **MSBuild-BigSolution1Node.bat** to execute it.

What was the total elapsed time to build the solution? \_\_\_\_\_

This will be listed at the bottom of the command widow.



```
FinalizeBuildStatus:
  Deleting file "Itanium\Debug\traceman.unsuccessfulbuild".
  Touching "Itanium\Debug\traceman.lastbuildstate".
Done Building Project "C:\Course\Labs\Lab02\BigSolution\Traceman\traceman.vcxproj" (Rebuild target(s)).

Done Building Project "C:\Course\Labs\Lab02\BigSolution\BigSolution.sln" (rebuild target(s)).

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

Time Elapsed 00:00:22.93
Press any key to continue . . .
```

5. Press any key to close the command window.
6. Right-click on **MSBuild-BigSolution2Nodes.bat** and select **Edit**.

This version includes the `/m:2` switch telling MSBuild to utilize two processors.

7. Close **Notepad**.
8. Double-click **MSBuild-BigSolution2Nodes.bat** to execute it.

You may get a couple of warnings. Not all of the projects in the solution are compatible with the parallelism settings.

What was the total elapsed time to build the solution? \_\_\_\_\_

9. Press any key to close the command window.

10. Review and run **MSBuild-BigSolution3Nodes.bat**.

What was the total elapsed time to build the solution? \_\_\_\_\_

11. Review and run **MSBuild-BigSolution5Nodes.bat**.

What was the total elapsed time to build the solution? \_\_\_\_\_

12. Review and run **MSBuild-BigSolution9Nodes.bat**.

What was the total elapsed time to build the solution? \_\_\_\_\_

**Did the builds get any faster? If not, then it could be that you don't have hardware that supports parallel builds. Also, feel free to edit any of the .bat files and substitute in your own values to run your own benchmarks.**

13. Close all instances of **Notepad** and the command windows.

## EXERCISE 5 – FILE-LEVEL BUILD PARALLELISM (OPTIONAL)

### TASK – CONFIGURE AND RUN A FILE-LEVEL PARALLEL BUILD

In this task, you will open a larger Visual C++ 2010 project, review it, and then build it using file-level parallelism.

1. Using **Windows Explorer**, navigate to **C:\Course\Labs\Lab02\drawcli**.
2. Double-click the **drawcli.sln** solution file to open it.

It may take a few moments for the solution to load the first time. You can monitor the progress on the status bar at the bottom. The drawcli application is a full-featured object-oriented drawing application that is also an ActiveX Visual Editing container.

3. From the **Tools** menu select **Options**.
4. Expand the **Projects and Solutions > Build and Run** page.
5. Ensure that the **maximum number of parallel project builds** value is set to **2**.
6. Ensure that the **MSBuild project build output verbosity** value is set to **Normal**.
7. Click **OK**.
8. From the **View** menu select **Output**.

This ensures the Output window is visible to show us the information for the build we are about to execute.

9. From the **Build** menu select **Rebuild Solution**.

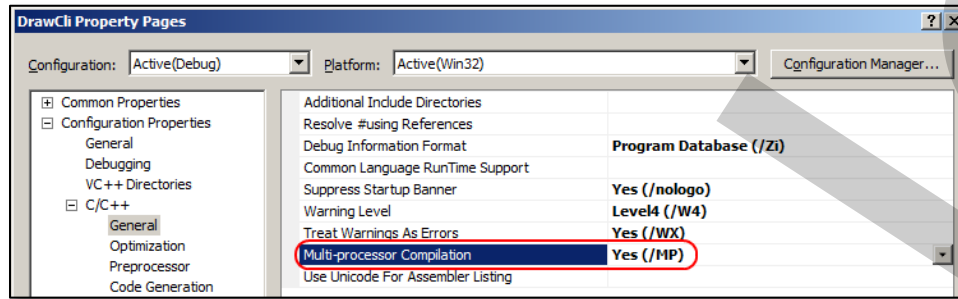
How long did it take to the project to build? \_\_\_\_\_

10. Perform two more rebuilds of the solution.

Write the average elapsed time here \_\_\_\_\_

11. In **Solution Explorer**, right-click on the **DrawCli** project and select **Properties**.
12. Expand the **Configuration Properties > C/C++ > General** page.

13. Change the **Multi-processor Compilation** property to **Yes (/MP)**.



14. Click **OK**.

15. From the **Tools** menu select **Options**.

16. Expand the **Projects and Solutions > VC++ Project Settings** page.

What is the value of your Maximum concurrent C++ compilations? \_\_\_\_\_

Note: a value of 0 causes all available processors to be utilized.

17. Click **Cancel**.

18. Perform three more rebuilds of the solution.

Record the elapsed times here \_\_\_\_\_

Did the builds get any faster? If not, then it could be that you don't have hardware that supports parallel builds.

19. Exit **Visual Studio**.

## Summary

Microsoft's decision for Visual C++ 2010 to leverage MSBuild is a huge step forward for Visual C++ developers. At first, the radically different file format may cause reactions ranging from indifference to concern over disruptions and breakages to any existing customizations to the build process. However, by allowing Visual C++ builds to take advantage of the wide variety of extensibility points in the MSBuild engine, the limited extensibility of the .VCPROJ format and VCBuild looks primitive by comparison.