

ElectricAccelerator Visual Studio Integration Guide

Version 4.2.4

Electric Cloud, Inc.
35 South Market Street, Suite 100
San Jose, CA 95113
www.electric-cloud.com

ElectricAccelerator Visual Studio Integration 4.2.4

Copyright © 2002–2016 Electric Cloud, Inc. All rights reserved.

Published 1/15/2016

Electric Cloud® believes the information in this publication is accurate as of its publication date. The information is subject to change without notice and does not represent a commitment from the vendor.

THE INFORMATION IN THIS PUBLICATION IS PROVIDED “AS IS.” ELECTRIC CLOUD, INCORPORATED MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WITH RESPECT TO THE INFORMATION IN THIS PUBLICATION, AND SPECIFICALLY DISCLAIMS IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Use, copying, and distribution of any ELECTRIC CLOUD software described in this publication requires an applicable software license.

Copyright protection includes all forms and matters of copyrightable material and information now allowed by statutory or judicial law or hereinafter granted, including without limitation, material generated from software programs displayed on the screen such as icons, screen display appearance, and so on.

The software and/or databases described in this document are furnished under a license agreement or nondisclosure agreement. The software and/or databases may be used or copied only in accordance with terms of the agreement. It is against the law to copy the software on any medium except as specifically allowed in the license or nondisclosure agreement.

Trademarks

Electric Cloud, ElectricAccelerator, ElectricCommander, ElectricDeploy, ElectricInsight, and Electric Make are registered trademarks or trademarks of Electric Cloud, Incorporated.

Electric Cloud products—ElectricAccelerator, ElectricCommander, ElectricDeploy, ElectricInsight, and Electric Make—are commonly referred to by their “short names”—Accelerator, Commander, Deploy, Insight, and eMake—throughout various types of Electric Cloud product-specific documentation.

Other product names mentioned in this guide might be trademarks or registered trademarks of their respective owners and are hereby acknowledged.

Contents

Chapter 1: Overview	1-1
About the ElectricAccelerator® Visual Studio Integration	1-1
Chapter 2: What's New	2-1
New Features and Functionality in Version 4.2.4	2-1
New Features and Functionality in Version 4.2.3	2-1
New Features and Functionality in Version 4.2.2	2-1
New Features and Functionality in Version 4.2	2-1
Resolved Issues in Version 4.2.3	2-3
Resolved Issues in Version 4.2.2	2-3
Resolved Issues in Version 4.2	2-3
Chapter 3: Known Issues	3-1
Chapter 4: System Requirements	4-1
Supported Visual Studio Versions	4-1
Prerequisites	4-2
MSBuild Prerequisites	4-2
VS IDE Add-In Prerequisites	4-2
VS Converter Add-In Prerequisites	4-2
ecdevenv Utility Prerequisites	4-2
Upgrade Notes	4-4
Chapter 5: Installing the IDE Add-In	5-1
Installing the Add-In Using the GUI	5-1
Installing the Add-In Silently	5-2
Choosing Installation Options	5-2
Chapter 6: Using the VS IDE Add-In Interface	6-1
Main Menu and Toolbar	6-2
Build Solution Locally	6-3
Output Pane	6-4
Context-Sensitive Menus	6-5
Solution Settings	6-7
General Solution Settings	6-7
Options Solution Settings	6-13
Debug Solution Settings	6-14
Advanced Solution Settings	6-17
Command Line Solution Settings	6-19
Chapter 7: Using the ecdevenv Utility	7-1
About ecdevenv	7-1

No Makefiles are Required	7-1
Miscellaneous Files that ecdevenv Generates	7-2
Virtualizing the Visual Studio Toolchain	7-2
Building Multiple Solutions and Projects	7-2
Enabling Debugging Output	7-3
Using ecdevenv Command Options	7-3
Using ecdevenv Environmental Variables	7-4
Using Fast Solution Conversion	7-5
Chapter 8: Using the VS Converter Add-In	8-1
Using ecdevenv	8-1
Creating a Makefile	8-1
Setting the Path for 64-Bit or Xbox Builds	8-2
Chapter 9: Setting VS Converter Add-In Environment Variables	9-1
Setting the Converter Add-In Environment Variables	9-1
Relocating Makefiles Generated by the Add-In	9-8
Setting the Environment Variable to Enable Local Solution Builds	9-9
Setting the Environment Variable for ecdevenv Startup Checks	9-9
Chapter 10: Tuning Performance	10-1
Improving Build Time for /Zi + PCH Builds	10-1
Improving Build Time for Solutions with Many Projects	10-1
Improving Final Link Time	10-2
Improving Incremental Build Time	10-2
Improving Incremental Linking Time	10-2
Improving Incremental Linking Time with the Add-In	10-2
Improving Incremental Linking Time without the Add-In	10-2
Optimizing Parallelization Using PDB Splitting	10-2
Optimizing Parallelization with the Add-In	10-2
Optimizing Parallelization without the Add-In	10-3
Chapter 11: Using MSBuild	11-1
Building MSBuild Projects in Parallel	11-1
Using MSBuild to Build a Project Directly in Its Solution Context	11-1
Chapter 12: Debugging a Failed Build	12-1
Chapter 13: Troubleshooting Problems	13-1
Initializing Visual Studio	13-1
Common Issues	13-1
Visual Studio is missing the Electric Cloud menu	13-2
Application Data folder could not be created. make: *** [all]	13-2
For VS2005 SP1 builds, the build is not broken up and runs as one large job	13-3
Build terminated with "not making progress" error	13-3
Visual Studio quits immediately at the start of the build	13-3
Error "'devenv' not found" is displayed	13-3
Error "Unable to build specified project" or missing file errors	13-4
Error "msbuild not found"	13-4
Missing DLL errors or Visual Studio installation is corrupt	13-4

Error “command line too long”	13-4
The build is slow (not parallelized) and/or each line of the build output is prefixed with 1>, 2>, etc ...	13-4
Error: ‘ ’ not recognized	13-5
When virtualizing the Visual Studio toolchain, regsvr32 fails trying to register a DLL that uses debug CRT DLLs	13-5
Particular projects do not build under eMake	13-5
Electric Cloud menu in Visual Studio is grayed out (disabled)	13-5
Invalid macro invocation '\$' build error	13-6
Using Visual Studio 2010, a project fails at link when using the add-in but succeeds when using Visual Studio alone	13-6
Upgrading only cluster agents to Accelerator v7.0 might cause an error	13-6
Index	14-1

Chapter 1: Overview

About the ElectricAccelerator® Visual Studio Integration

The ElectricAccelerator® Visual Studio integration is composed of two distinct add-ins and the ecdevenv.exe utility:

- Visual Studio IDE Add-In (VS IDE Add-In)

This add-in integrates with the Microsoft Visual Studio IDE and lets you build Visual Studio solutions and projects using Electric Make® (eMake) from within the IDE. The add-in provides an Electric Cloud build menu and toolbar. The existing build menu remains intact for local (non-eMake) builds.

- Visual Studio Converter Add-In (VS Converter Add-In)

This is a command line add-in used by eMake to convert Visual Studio projects into NMAKE makefiles.

- ecdevenv.exe Utility

ecdevenv is a drop-in replacement for devenv.exe that builds Visual Studio solutions and projects using eMake. It provides a number of important features. See [Using the ecdevenv Utility](#) for information.

The ElectricAccelerator Visual Studio integration is installed automatically during the installation of ElectricAccelerator or ElectricAccelerator Huddle™.

Chapter 2: What's New

New Features and Functionality in Version 4.2.4

- Allow the add-in to be switched off for certain solutions. (VSP-938)
- Fixed “fatal error C1041” when the PDB file is in a non-standard location. (VSP-950)
- Fix “The system cannot find the path specified” when building with `ECADDIN_UP_TO_DATE_CHECK=true`. (VSP-958)

New Features and Functionality in Version 4.2.3

- When using `ecdevenv /quick`, add switch `ECDEVENV_USE_SOLUTION_CONTEXT` to use the solution context for all projects and switch `ECDEVENV_USE_SOLUTION_CONTEXT_FOR_PROJECTS` to build specific projects in the solution context. (VSP-869)
- Enable the *basic* annotation detail in the IDE add-in by default. (VSP-897)
- New solutions do not need to be closed and re-opened prior to building in the IDE add-in. (VSP-925)
- Allow `ecdevenv /quick` builds to skip duplicate projects by setting `ECDEVENV_SKIP_DUPLICATE_PROJECTS`. (VSP-936)
- Improve the reliability of first-time builds by setting `--emake-readdir-conflicts=1` by default in `ecdevenv` builds. (VSP-937)

New Features and Functionality in Version 4.2.2

- A pop-up message is introduced that warns if a directory that you manually added to the eMake root does not exist at build time. (VSP-901)
- (Visual Studio 2005) A message that appears at the Visual Studio prompt at build time is introduced to warn if a critical Microsoft hotfix for Visual Studio 2005 Service Pack 1 is not installed. (Without the hotfix, the VS Converter Add-In will not work correctly, which means that the build will not be accelerated.) For more information and download instructions for the hotfix, see the “[Visual Studio 2005 behaves as if the Visual Studio Add-in is not installed](#)” article on the Electric Cloud “Ask” website. (VSP-900)

New Features and Functionality in Version 4.2

Version 4.2 of the Visual Studio Integration fixes several issues and introduces a number of improvements. The most important improvements are

- Functionality that adds locations of all project inputs and outputs (C++ only) as well as project and solution locations (for all project types) to the eMake root automatically
- Changes to default settings in the IDE Add-In and the ecdevenv utility to increase speed and ensure correctness of incremental builds

The following list provides details about the improvements that are included in version 4.2.

- In the IDE add-in, support is added for the 64-bit version of ElectricInsight, which is introduced in ElectricInsight 5.0. On 64-bit systems, the **Electric Cloud > Run ElectricInsight** menu option invokes the 64-bit version. (VSP-875)
- In the IDE add-in, the **Add Emake Roots Automatically** checkbox is added to the **General > Basic** tab of the Electric Cloud **Solution Settings** dialog box. When checked, this checkbox adds the locations of all inputs (such as .cpp and .h files) and outputs (such as .obj, .exe, and .dll files) in a project to the eMake root automatically. For details and limitations, see the [Basic Solution Settings](#) section in the [Using the VS IDE Add-In Interface](#) chapter. Project locations and each location for the solution(s) are now added automatically to the eMake root. (VSP-834 and VSP-759)
- To ensure correct incremental builds, the eMake eDepend feature (also called Auto Depend) is now enabled by default in ecdevenv when ecdevenv is invoked from the Visual Studio command prompt. (When Auto Depend is enabled in the IDE Add-In GUI, this default is overridden when ecdevenv is called from the GUI.) (VSP-818)
- In the IDE add-in, to increase the speed of incremental builds, the default values of the `ECADDIN_ENABLE_INCREMENTAL_LINK` and `ECADDIN_UP_TO_DATE_CHECK` environment variables are changed from `false` to `true`. The **Enable Incremental Link** checkbox and the **Up To Date Check** checkbox in the **General > Performance** tab of the Electric Cloud **Solution Settings** dialog box are now checked by default. (VSP-815)
- (Available in Visual Studio 2005 and newer) You can now convert solutions to NMAKE more quickly by parallelizing down to the project level only. To do so, you can use the `/quick` option in ecdevenv or the **Coarse Grain Parallelization** checkbox in the **General > Performance** tab in the Electric Cloud **Solution Settings** dialog box. This feature does not require Visual Studio to be installed. For details, see the [Using Fast Solution Conversion](#) section in the [Using the ecdevenv Utility](#) chapter. (VSP-881 and VSP-820)
- In the converter add-in, the copying of project references now occurs in parallel to improve incremental build performance. (VSP-816)
- When invoking ecdevenv from the Visual Studio command prompt, if you try to build a solution that was created in another Visual Studio version, a warning appears at the prompt. (VSP-804)
- (Visual Studio 2010 and newer) In the converter add-in, to reduce the number of build failures because of files locked by multiple MSBuild processes, the `nodeReuse` parameter is now automatically set to `false` in all calls to invoke MSBuild. (VSP-801)
- The `ECADDIN_MAKEFILE_CACHE` environment variable is introduced. This lets you specify an alternate folder for the cached (temporary) makefiles that the converter add-in generates. For details, see the [Setting VS Converter Add-In Environment Variables](#) chapter. (VSP-791)

- The `ECADDIN_BUILD_PROJECTS_IN_SOLUTION_CONTEXT` environment variable and the `BuildProjectsInSolutionContext` configuration file variable are introduced for the converter add-in. When set to `true`, they let you use MSBuild to build projects directly, but in the solution context. The **Use Solution in MsBuild Projects** checkbox, which also enables this behavior, is added to the **General > Options** tab of the Electric Cloud **Solution Settings** dialog box.

Also, the `ECADDIN_BUILD_SPECIFIC_PROJECTS_IN_SOLUTION_CONTEXT` environment variable and the `BuildSpecificProjectsInSolutionContext` configuration file variable are introduced for the converter add-in. These variables let you enter a specific list of projects to build directly in MSBuild, but in the solution context. The **Use Solution For Specified MsBuild Projects** checkbox, which also enables this behavior, is added to the **General > Options** tab of the Electric Cloud **Solution Settings** dialog box. (VSP-872 and VSP-790)

- (Visual Studio 2010 and newer) The default value of the `ECADDIN_REMOVE_DEPENDENCIES` environment variable is changed from `true` to `false`. In the IDE add-in, the **Remove Dependencies** checkbox in the **General > Performance** tab of the Electric Cloud **Solution Settings** dialog box is now unchecked by default. (VSP-788)
- `ecdevenv` “diff” reporting in the Visual Studio console and in the Visual Studio **Output** window is improved. If the makefile is regenerated, the reason for the regeneration is described more clearly. (VSP-676)

Resolved Issues in Version 4.2.3

- Fixed the fatal error U1071: cycle in dependency tree for target. (VSP-928)
- The `ecdevenv /quick` option should not check for Visual Studio to be in the PATH environment variable. (VSP-934)
- Fixed issues building `ecdevenv` configuration files containing solutions with relative paths. (VSP-935)

Resolved Issues in Version 4.2.2

- (Visual Studio 2010 and newer) Fixed an issue where commas used in a definition in the **Preprocessor Definitions** field in the **Property Pages** dialog box were incorrectly parsed. (VSP-910)
- Fixed the issue that produced build errors such as `Invalid macro invocation '$'`. This was caused by an NMAKE limitation that treated the dollar sign (\$) as a special character that preceded a macro name. You can now use \$ in a preprocessor definition with an odd number of \$ symbols. (VSP-874)

Resolved Issues in Version 4.2

- (Visual Studio 2002 and 2003) Fixed an issue where the converter add-in attempted to invoke MSBuild when used with Visual Studio 2002 and 2003. The add-in now invokes `devenv` when used on non-parsed (non-C++) projects such as C# projects. (VSP-888)

- (Visual Studio 2013) Fixed the issue where MSBuild.exe was launched from the .NET directory (rather than from C:\Program Files (x86)\MSBuild\12.0\Bin). This caused the following error message:

```
C:\Program Files (x86)
\MSBuild\Microsoft\VisualStudio\v12.0\CodeAnalysis\Microsoft.CodeAnalysis.targets
(214,5): error MSB4175: The task factory "CodeTaskFactory" could not be loaded from
the assembly
"C:\Windows\Microsoft.NET\Framework\v4.0.30319\Microsoft.Build.Tasks.v12.0.dll".
Could not load file or assembly
'file:///C:\Windows\Microsoft.NET\Framework\v4.0.30319\Microsoft.Build.Tasks.v12.0.
dll' or one of its dependencies. The system cannot find the file specified. (VSP-
847)
```

- Fixed an issue where the converter add-in tried to create a link command for an EXE project that had additional dependencies but no objects. (VSP-846)
- (Visual Studio 2010) Removed the additional /AI and /clr:nostdlib eMake command lines that were not present in the Visual Studio build. (The fix for VSP-474 added these command lines to the converter add-in to work around a Visual Studio 2010 issue, which Microsoft has now fixed.) (VSP-845)
- Fixed an issue where the converter add-in did not support the Streaming SIMD Extensions 2 instruction set on 64-bit builds, which caused the following compiler error message when the **Enable Enhanced Instruction Set** configuration property was set to anything other than **Not Set** for a 64-bit build: `cl : Command line warning D9002 : ignoring unknown option '/arch:SSE2'.` (VSP-844)
- Fixed an issue where unchecking the **Use Local Agents** checkbox in the **General > Cluster** tab of the Electric Cloud **Solution Settings** dialog box of the IDE add-in did not turn off usage of local agents. (VSP-769)
- Fixed the issue where the **Cluster Manager** and **Emulation/Platform** debug logging level checkboxes were missing from the **Debug > Emake** tab of the Electric Cloud **Solution Settings** dialog box of the IDE add-in. (VSP-743)

Chapter 3: Known Issues

- If the cluster upgrade option of the VS IDE Add-In installer fails, re-run the installer to ensure the VS Converter Add-In is installed correctly.
- Because of an issue with previous versions' uninstallers, an upgrade might cause the following error message: `Cannot find script file: C:\ECloud\i686_win32\bin\unregaddin.vbs`. You can ignore this message.
- Make sure you finish all Visual Studio installations *before* installing the VS IDE Add-In. Adding a new language to an existing Visual Studio installation with the VS IDE Add-In already installed causes Visual Studio to display an empty Electric Cloud menu. The workaround is to reinstall the add-in.
- For Visual Studio 2012 or later, the project build order under eMake might be different to Visual Studio if project dependencies are not fully defined.

Workaround: If a build fails because a prerequisite project has not been built, add an explicit project dependency in the solution.

- Visual Studio 2008 builds might break or might not be optimized after upgrading from an earlier version.

Workaround: See knowledge base article [KBEA-00065](#), *The build breaks after upgrading to Visual Studio 2008*.

- Microsoft Visual C++ 2010 projects that contain "custom build rules" will not be parallelized at the project item level.
- Lightswitch projects are not supported.
- You might encounter a build error such as `Invalid macro invocation '$'`. This is caused by an NMAKE limitation that treats the dollar sign (\$) as a special character that precedes a macro name. You cannot use (\$) in a preprocessor definition unless there is an even number of (\$) symbols.

Workaround: Either do not use the single dollar sign (\$), or specify it by using a double dollar sign (\$\$).

- For Visual Studio 2010 and later, the MSBuild task batching syntax is not supported for C++ build events (pre-build, pre-link, and post-build events).

Workaround: Substitute the variables with actual values.

Chapter 4: System Requirements

Supported Visual Studio Versions

The Visual Studio IDE Add-In (VS IDE Add-In) supports the following versions of Visual Studio:

- Visual Studio 2013
- Visual Studio 2012
- Visual Studio 2010
- Visual Studio 2008
- Visual Studio 2005

Note: The VS IDE Add-In for Visual Studio 2010 or later does not support Xbox builds, Windows Mobile configurations, or custom build rules.

The VS Converter Add-In supports all .NET versions of Visual Studio:

- Visual Studio 2013
- Visual Studio 2012
- Visual Studio 2010
- Visual Studio 2008
- Visual Studio 2005
- Visual Studio .NET 2003
- Visual Studio .NET 2002

The ecdevenv utility supports all .NET versions of Visual Studio:

- Visual Studio 2013
- Visual Studio 2012
- Visual Studio 2010
- Visual Studio 2008
- Visual Studio 2005

- Visual Studio .NET 2003
- Visual Studio .NET 2002

Prerequisites

MSBuild Prerequisites

The IDE add-in cannot virtualize the MSBuild utility. If you are virtualizing your toolchain, you must install the following packages for your version of Visual Studio on every agent host in your cluster.

Visual Studio Version	Package(s)
2013	<ul style="list-style-type: none">• .NET Framework 4.5• Microsoft Build Tools 2013
2012	.NET Framework 4.0.30319
2010	.NET Framework 4.0.30319
2008	.NET Framework 3.5
2005	.NET Framework 2.0.50727

If you do not virtualize your toolchain, you must install Visual Studio on each agent host in your cluster.

VS IDE Add-In Prerequisites

The VS IDE Add-In requires:

- eMake installed on the build machine
- ElectricAccelerator v7.0.2 or later
- .NET Framework v2.0

VS Converter Add-In Prerequisites

The VS Converter Add-In requires:

- Microsoft Visual C++ 2005 SP1 Redistributable Package (all Visual Studio versions)
- .NET Framework v2.0
- (If using Visual Studio 2005 Service Pack 1) Microsoft hotfix that is described in Microsoft Knowledge Base articles 933054 and 934517. Without the hotfix, the VS Converter Add-In will not function correctly, which means that `ecdevenv` is not called, so builds will not be accelerated.

For download information, see the [“Visual Studio 2005 behaves as if the Visual Studio Add-in is not installed”](#) article on the Electric Cloud “Ask” website.

ecdevenv Utility Prerequisites

The `ecdevenv` utility requires:

- VS Converter Add-In
- .NET Framework v2.0

Upgrade Notes

- When you upgrade from VS Converter Add-In version 3.0, “debug” builds using PDB files will generate many conflicts on the first build. This occurs because the build order might have changed since version 3.0. (Subsequent builds will be fine.)
- Electric Cloud recommends regenerating the history file after upgrading the VS Converter Add-In.

Chapter 5: Installing the IDE Add-In

Installing the Add-In Using the GUI

To install the VS IDE Add-In locally, run the installer provided.

1. Right-click the VSAddIn-<version>-Install.exe file and choose **Run as administrator**.
2. When the **Install ElectricAccelerator VS IDE Add-In** popup appears, click **Yes**.
3. When the welcome screen appears, click **Next**.
4. When the **Choose Destination Location** screen appears, click **Next** to accept the default installation location (C:\ECloud) or click **Browse** to change the location.
5. When **Setup Type** screen appears, click to choose a setup type:
 - **ElectricAccelerator VS IDE Add-in Local Install**
 - **ElectricAccelerator VS Converter Add-in Local Install**
 - **ElectricAccelerator VS Converter Add-in Cluster Upgrade**—Upgrades the VS Converter Add-In (Solution Support Add-In) on all Windows cluster agents that are registered to the Cluster Manager that you specify.

Note: For the VS Converter Add-In cluster upgrade to proceed, the installation directory on all agents must be C:\ECloud, and you must have installed eRunner on the Cluster Manager and agent machines.

This setup type is not required when you are virtualizing the Visual Studio toolchain, because in this case, the converter add-in is not actually installed on the agent machines.

- **Custom**—Lets you select multiple setup types (components) from this list.

Click **Next**.

5. When the **Start Copying Files** screen appears, review your settings before continuing the installation. Click **Next** to continue or **Back** to make changes.
6. When the **InstallJammer Wizard Complete** screen appears (it displays “Install finished”), your installation is complete. Click **Finish** to close the installer.

The installation log file is in the installation directory’s root, C:\ECloud by default.

Note: Installing just the Converter Add-In uninstalls the existing IDE Add-In if present.

Installing the Add-In Silently

To perform a silent install, follow these steps:

1. Run an installation with the `/save-response-file <filename>` option and your desired settings.

This creates the response file in the directory where you ran the installer.

2. Use the resulting response file for silent identical installs by using the `/response-file <filename>` and `/mode silent` options.

Choosing Installation Options

Use this structure for options: `<Install filename> [options]`

The following options are available to customize your installation:

Option	Description
<code>/help</code>	Displays help information.
<code>/mode [ARG]</code>	Sets the installation mode. Available values: <code>standard</code> or <code>silent</code> .
<code>/prefix [ARG]</code>	Sets the installation directory.
<code>/response-file [ARG]</code>	The file from which to read installer responses.
<code>/save-response-file [ARG]</code>	The file to which installer responses are written when the installer exits.
<code>/temp [ARG]</code>	Sets the temporary directory used by the program.
<code>/type [ARG]</code>	Performs the selected type of installation. Available values: <code>addin</code> , <code>uiaddin</code> or <code>cluster</code> .
<code>/version</code>	Displays installer version information.

Chapter 6: Using the VS IDE Add-In Interface

When the VS IDE Add-In starts, it checks if the following are present:

- eMake— If the add-in cannot find eMake, the add-in's Build/Rebuild/Clean functions are disabled.

When eMake is run from Visual Studio, it must be run through an intermediate application named `ecspawn.exe`. This program ensures that eMake responds correctly to Ctrl-C and that child processes are grouped together. This application is displayed in the Task Manager. Do not terminate the application; it stops when the build finishes or when the build is canceled.

Do not run local (non-eMake) builds while running eMake builds and vice-versa.

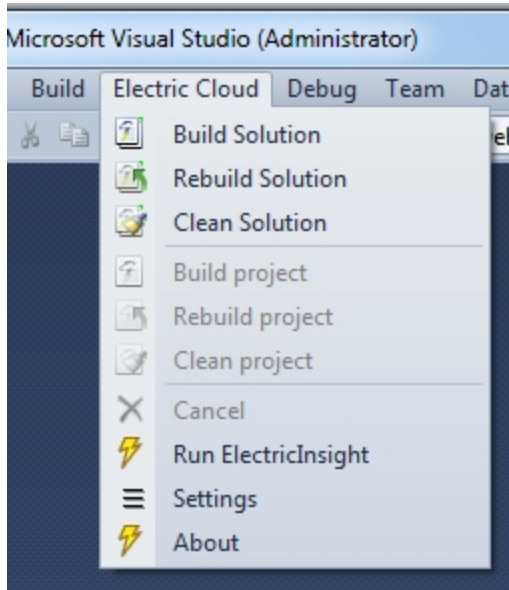
- ElectricInsight®—If the add-in cannot find Insight, the add-in's Run ElectricInsight function is disabled.

Topics:

- [Main Menu and Toolbar](#)
- [Solution Settings](#)

Main Menu and Toolbar

When you run Visual Studio, you are presented with the Electric Cloud main menu (displayed in the following screenshot) and toolbar:



The menu has the following functions:

- Build Solution—Builds the current solution
- Build Solution Locally—Builds the current solution locally with eMake but **without using** remote agents or local agents (this function is equivalent to turning off the Cluster Manager and local agents). This function is hidden by default. See [Build Solution Locally](#) for additional information about this function.
- Rebuild Solution—Rebuilds the current solution
- Clean Solution—Cleans the current solution
- Build <project>—Builds the current project or selection
- Rebuild <project>—Rebuilds the current project or selection
- Clean <project>—Cleans the project or selection
- Cancel—Cancels a running eMake build. When a build is running, you can cancel it by selecting Cancel. Cancel is available only during a running build, rebuild, or clean.

- **Run ElectricInsight**—Runs Insight with the current annotation file (if it exists). At any time, you might run ElectricInsight (Insight) to view the annotation file. Insight loads the specified annotation file or defaults to `emake.xml`.

When you run Insight from Visual Studio, Visual Studio looks for the currently running instance of `einsight`. If `einsight` is currently running, the annotation file is not loaded (or reloaded). Manually open the annotation file from Insight, or close Insight and select Run ElectricInsight again.

On 64-bit systems, this menu option invokes the 64-bit version of ElectricInsight. For details about the 64-bit and 32-bit versions of ElectricInsight, see Chapter 1, *Installing ElectricInsight* in the *ElectricInsight User Guide* at http://docs.electric-cloud.com/accelerator_doc/AcceleratorIndex.html.

- **Settings**—Opens the solution settings dialog
- **About**—Displays add-in information

When selecting one of the build commands, the add-in calls `ecdevenv.exe` to perform the build.

The project and configuration are taken from the current context. The command is dependent on the menu item.

Note: A new unsaved C++ solution cannot be built until the solution .ecd file has been named. Before building with eMake, first close the solution and then reopen it.

The toolbar provides the same functionality as the Electric Cloud main menu and is customizable.

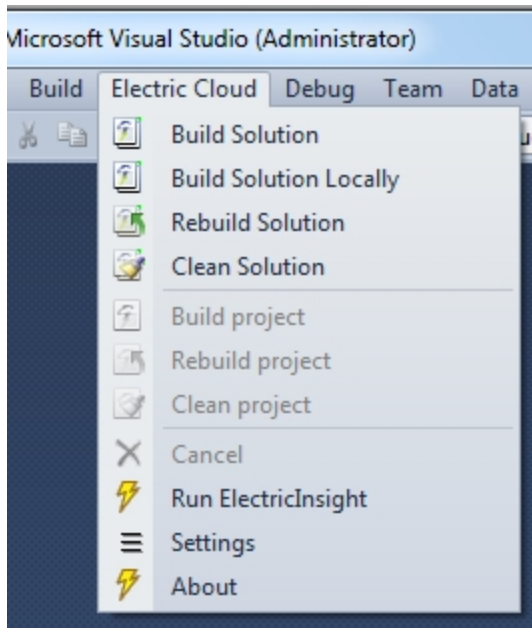


Build Solution Locally

You can choose to build a solution locally with eMake but ***without using*** remote agents or local agents. You might want to use this function if a distributed incremental build is slow, or if a local Visual Studio incremental build causes unnecessary rebuilding of objects.

To make this function visible in the menu, set the environment variable `ECUIADDIN LOCAL BUILD=true`.

The following screenshot illustrates the menu with the Build Solution Locally function.

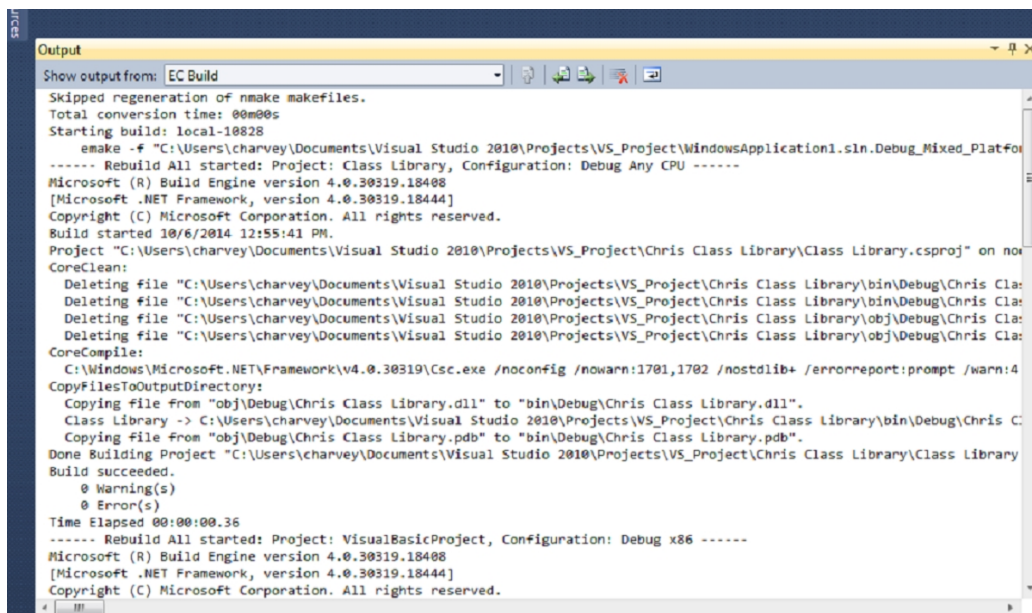


Advisories for Build Solution Locally

- The eMake local build does **not** support the eMake eDepend feature (also called Auto Depend). This means changes in header files might not cause dependent source files to be recompiled.
- The eMake local build does **not** produce an annotation.
- Because history is not generated, unexpected conflicts might occur on subsequent eMake cluster builds.

Output Pane

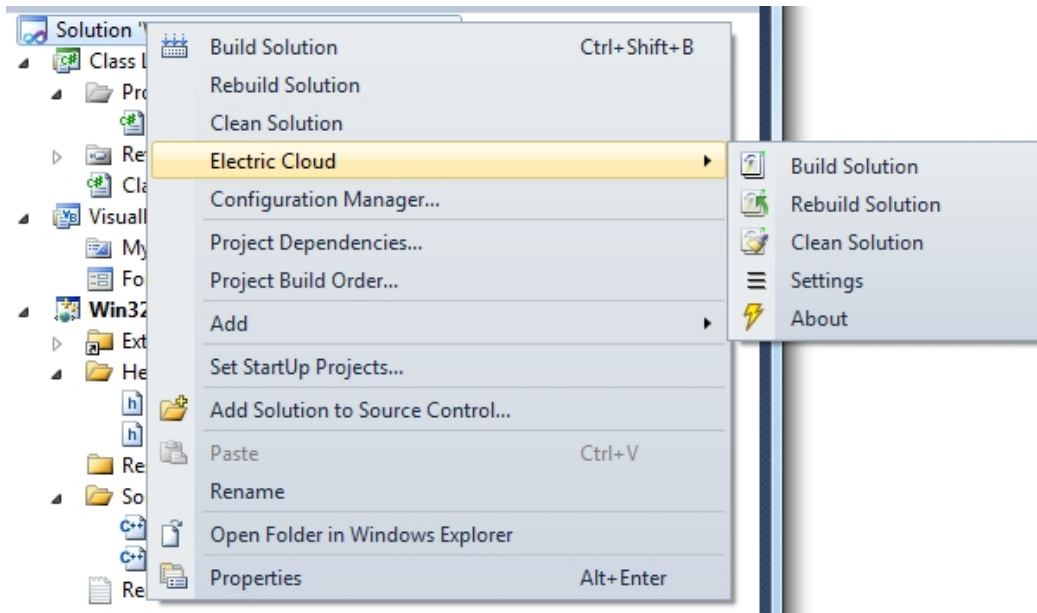
Output from an eMake build is displayed in the EC Build output pane (displayed in the following screenshot).



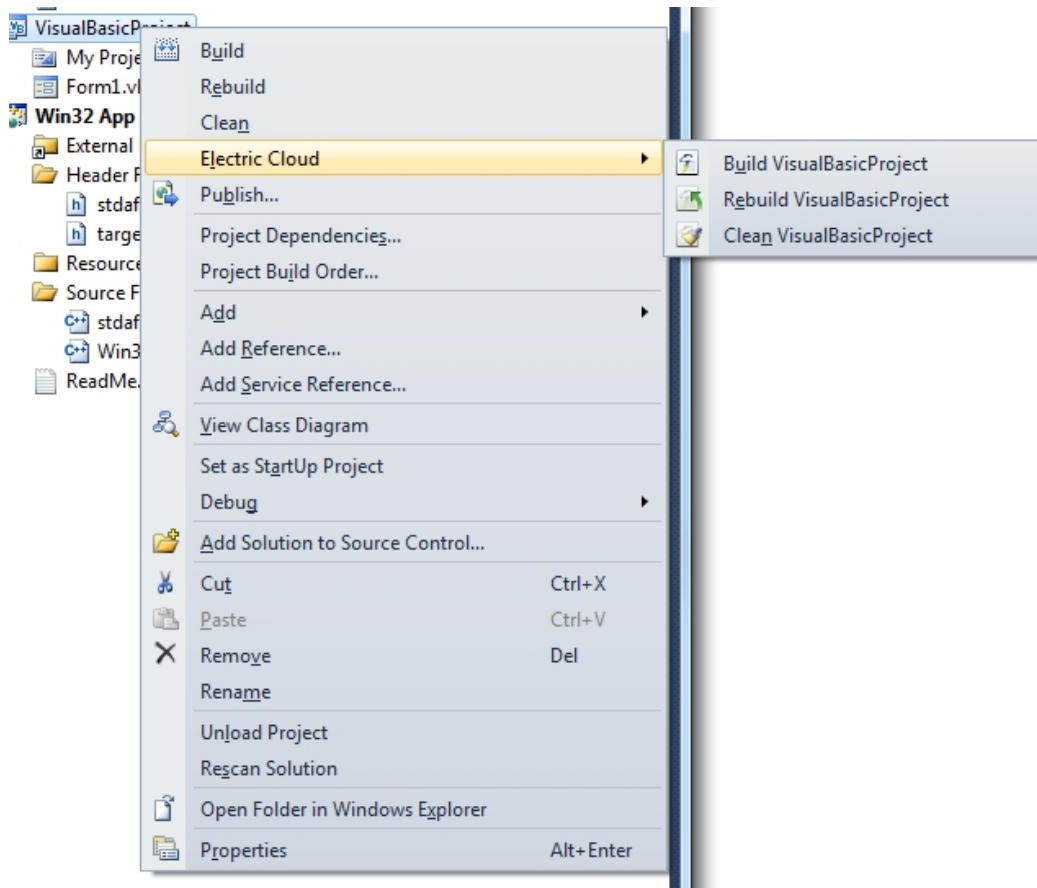
Context-Sensitive Menus

The add-in provides additional context-sensitive menus.

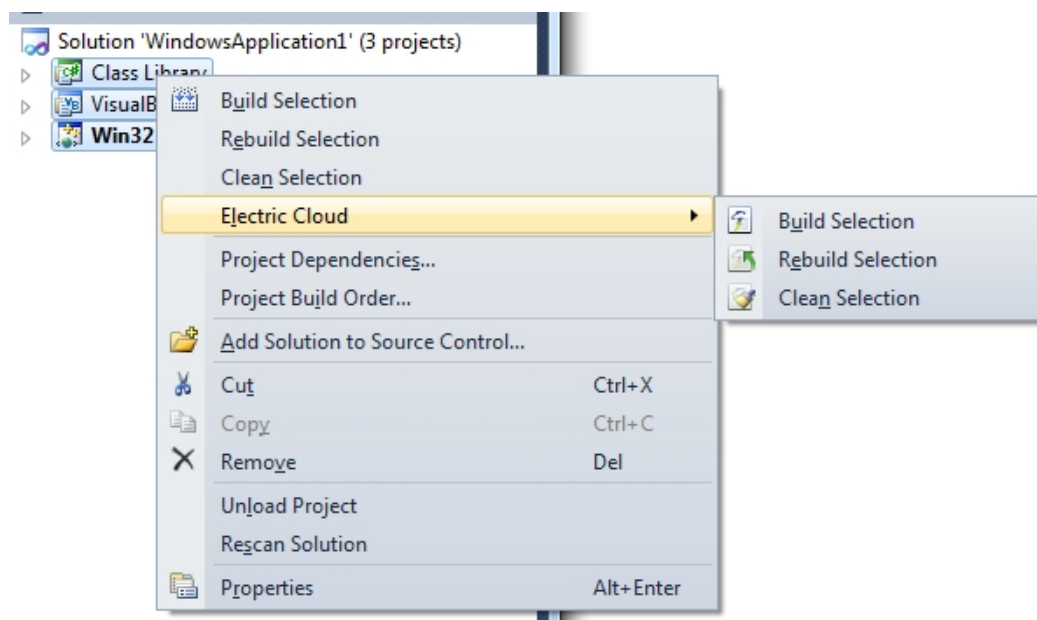
The following screenshot illustrates the Solution menu.



The following screenshot illustrates the Project menu.



The following screenshot illustrates the Selection menu.



Solution Settings

The add-in contains the following Solution Setting categories:

- [General Solution Settings](#)
- [Options Solution Settings](#)
- [Debug Solution Settings](#)
- [Advanced Solution Settings](#)
- [Command Line Solution Settings](#)

Notes:

- The add-in no longer supports global options. Consequently, solution settings do not inherit from global settings. If you have been using an earlier version of the add-in, the previous global settings are migrated to solution settings when you first open a solution.
- See [Setting VS Converter Add-In Environment Variables](#) for environment variable descriptions.

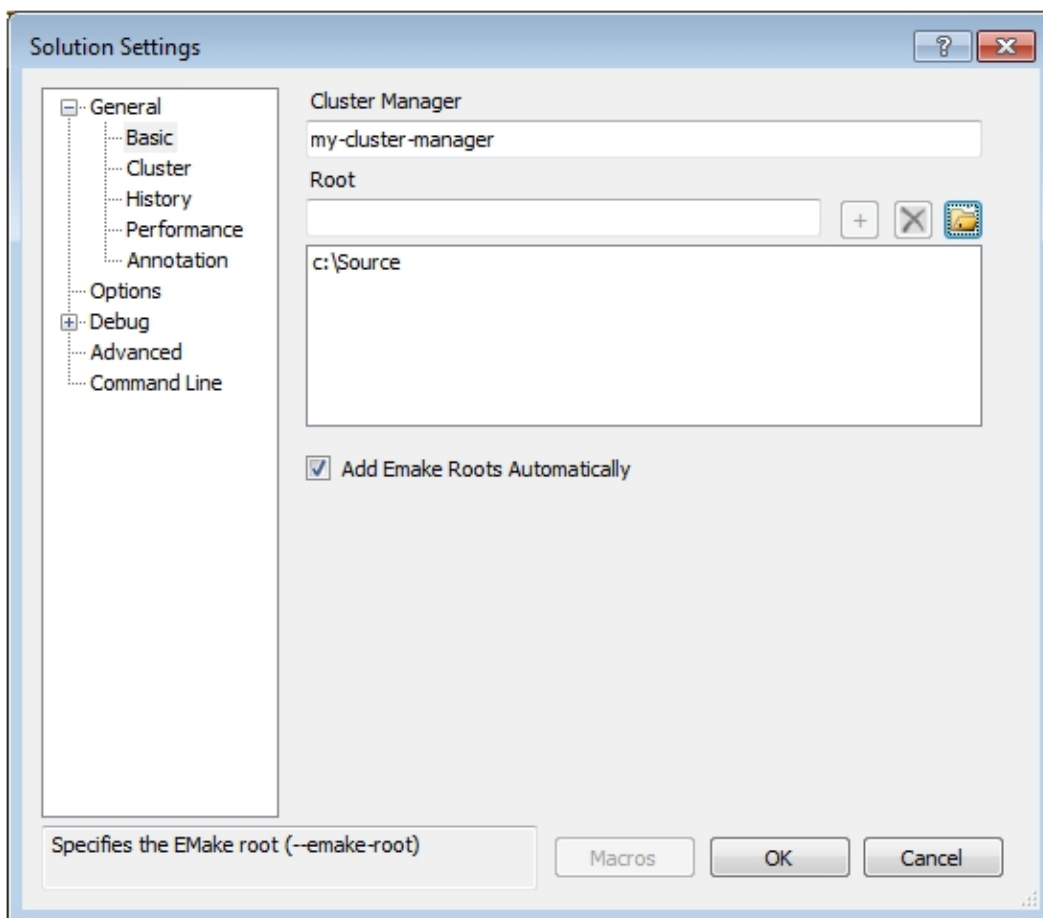
General Solution Settings

This category contains most frequently used settings:

- [Basic Solution Settings](#)
- [Cluster Solution Settings](#)
- [History Solution Settings](#)
- [Performance Solution Settings](#)
- [Annotation Solution Settings](#)

Basic Solution Settings

The following screenshot illustrates the Basic sub-category.



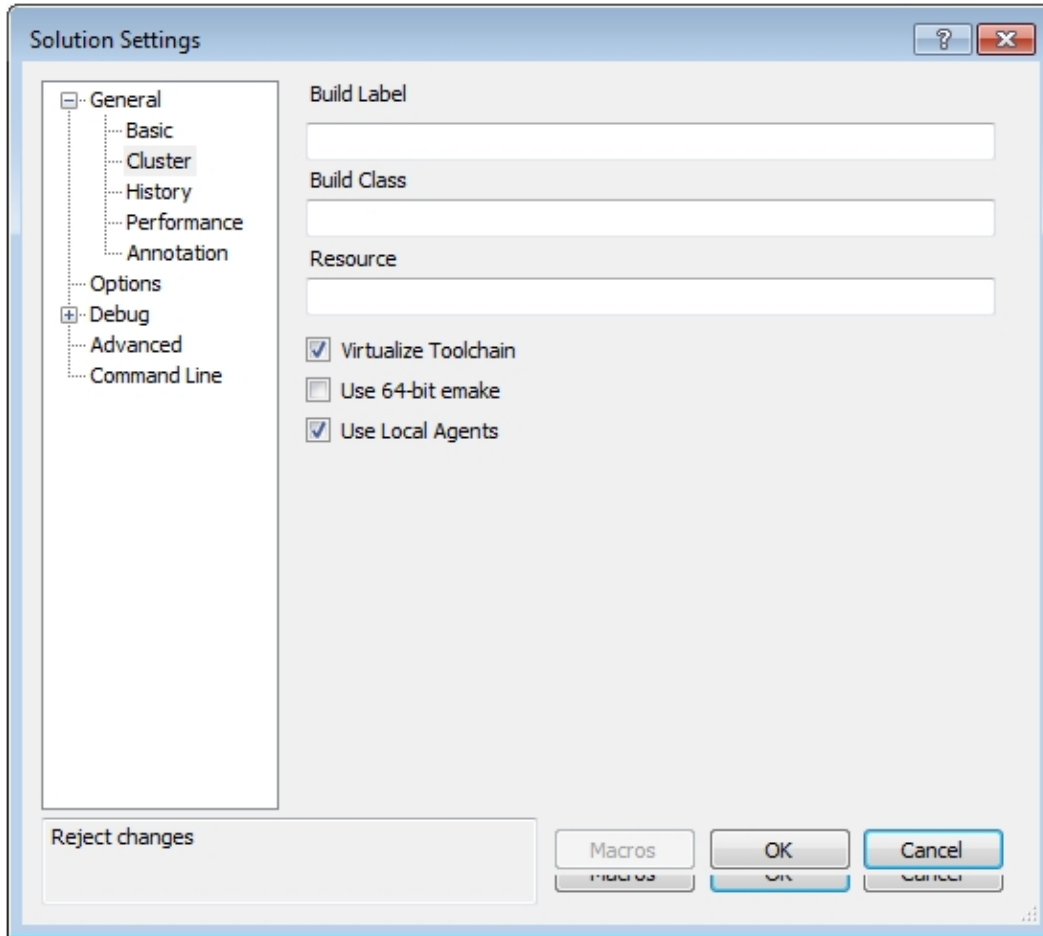
- **Cluster Manager**—Indicates the eMake Cluster Manager (`--emake-cm`). If this field is empty, an eMake build is performed with local agents when **Use Local Agents** is selected. When **Use Local Agents** is not selected, a local eMake build (without remote or local agents) is performed.
- **Root**—Specifies the eMake root (`--emake-root`). To add a path to the eMake root manually, enter a path or click the folder button to browse and click the plus button to add it to the list. To delete a path from the eMake root manually, select it in the list and click the x button.
- **Add Emake Roots Automatically**—Adds the locations of all inputs (such as .cpp and .h files) and outputs (such as .obj, .exe, and .dll files) to the eMake root automatically.

This checkbox applies only to C++ projects. If you have a C# project or a project with inputs and outputs not in the solution or project locations, you must add their locations to the eMake root manually.

This checkbox does not apply to third-party tools, because they cannot be virtualized. This checkbox does not apply to project locations and each location for the solution(s), because they are always added automatically.

Cluster Solution Settings

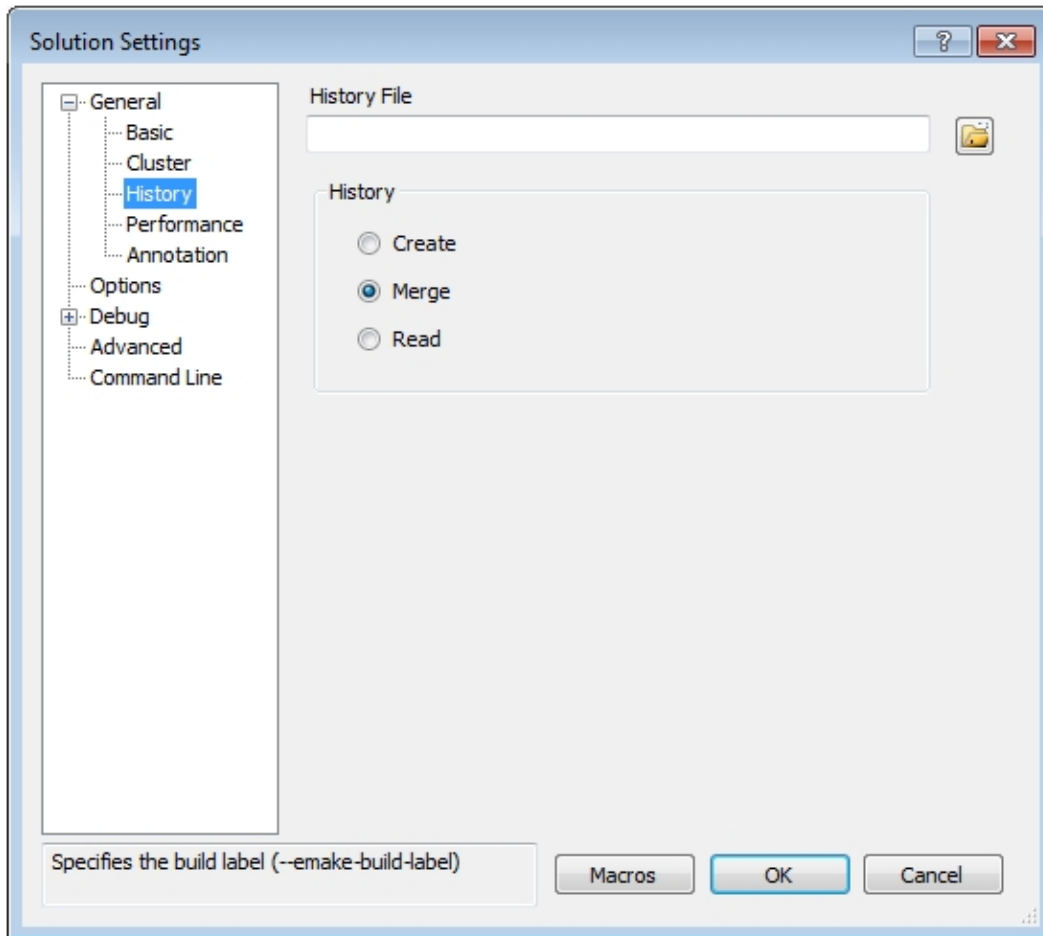
The following screenshot illustrates the Cluster sub-category.



- Build Label—Specifies the build label (`--emake-build-label`).
- Build Class—Specifies the build class (`--emake-class`).
- Resource—Specifies the build resource (`--emake-resource`).
- Virtualize Toolchain—Determines whether to virtualize the Visual Studio toolchain. This checkbox is checked by default.
- Use 64-bit eMake—Determines whether to use the 64-bit version of eMake. This checkbox is unchecked by default.
- Use Local Agents—Determines whether to use local agents (`--emake-localagents`). This checkbox is checked by default.

History Solution Settings

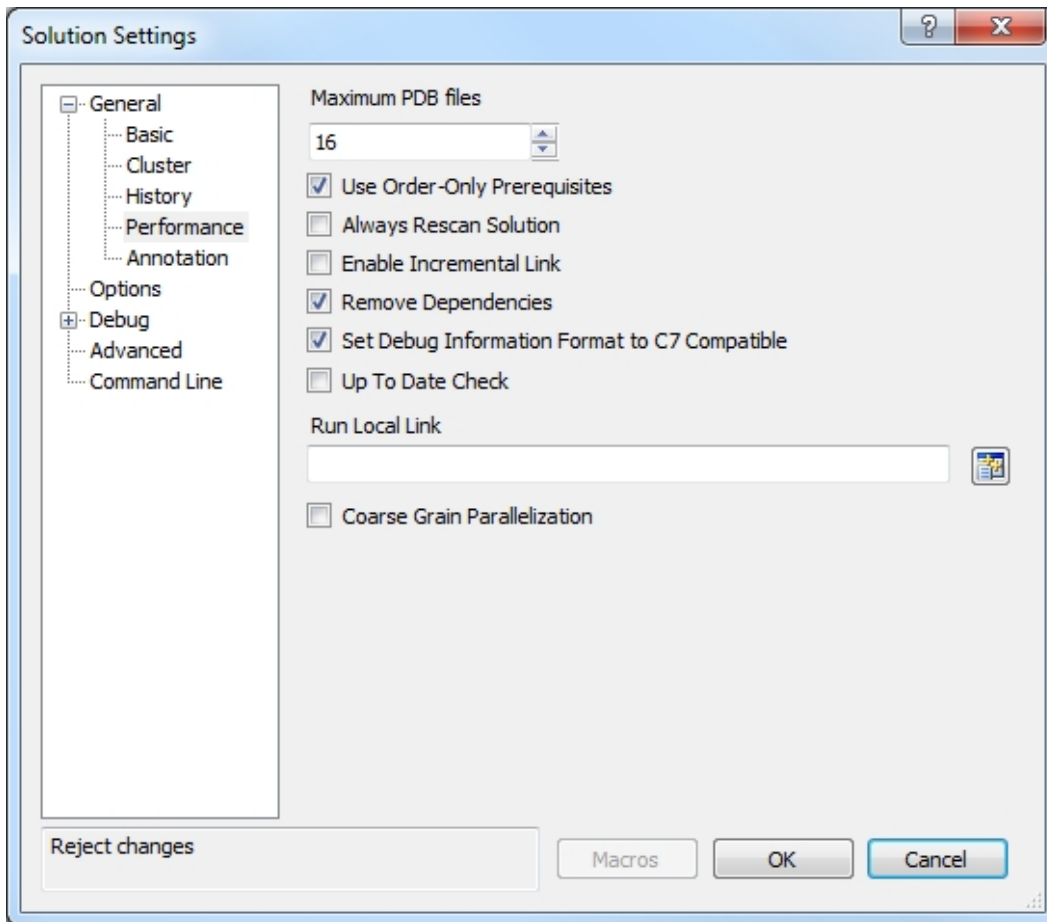
The following screenshot illustrates the History sub-category.



- History File—Specifies the history file to use (`--emake-historyfile`). The default is `eMake.data`.
- History—Specifies the eMake history option (`--emake-history`). Available values: create, merge, or read.

Performance Solution Settings

The following screenshot illustrates the Performance sub-category.

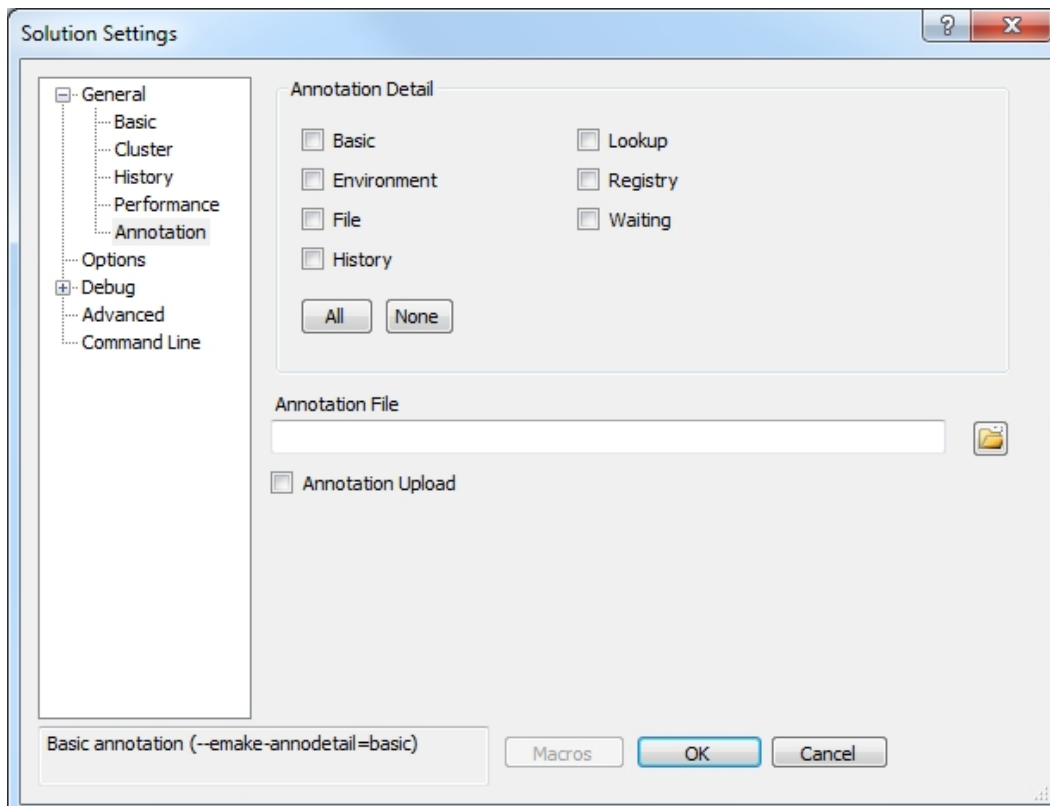


- **Maximum PDB Files**—Sets the maximum number of PDB files used when splitting (sets `ECADDIN_MAX_PDB_FILES`). Default is 16. If you have fewer than 16 agents, you can decrease this value to be equal to or less than the number of agents.
- **Use Order-Only Prerequisites**—Determines whether to use order-only prerequisites (sets `ECADDIN_USE_ORDER_ONLY_PREREQS=true` if enabled). Using order-only prerequisites can improve first-time build speed.
- **Always Rescan Solution**—Determines whether to always recreate temporary makefiles even if the solution has not changed.
- **Enable Incremental Link**—Enables/disables incremental linking (sets `ECADDIN_ENABLE_INCREMENTAL_LINK=true` if enabled).
- **Remove Dependencies**—Determines whether to remove dependencies and references (sets `ECADDIN_REMOVE_DEPENDENCIES=true` if enabled). Removing dependencies prevents Visual Studio from building dependent projects.
- **Set Debug Information to C7 Compatible**—Determines whether to force compiler option `/Z7` (sets `ECADDIN_FORCE_Z7`).
- **Up To Date Check**—Determines whether to check if anything requires building (set `ECADDIN_UP_TO_DATE_CHECK=true` if enabled).

- **Run Local Link**—Links specified projects locally using `#pragma runlocal (sets ECADDIN_RUN_LOCAL_LINK)`. Use the projects button to select projects.
- **Coarse Grain Parallelization**—Converts the solution to NMAKE more quickly by parallelizing down to the project level only. It does not require Visual Studio to be installed. Using this option is equivalent to using the `ecdevenv /quick` option. This feature works only with Visual Studio 2005 and newer versions.

Annotation Solution Settings

The following screenshot illustrates the Annotation sub-category.



- **Annotation Detail**—Specifies the level of annotation detail (`--emake-annodetail`) from the following selections:
 - **Basic**—Collects information about every command run by the build. Detailed information about each “job” in the build is recorded, including command arguments, output, exit code, timing, and source location. In addition, the build structure is represented as a tree where each recursive make level is represented in the XML output.
 - **Environment**—Adds information about environment variable modifications.
 - **File**—Adds information about files read or written by each job.
 - **History**—Adds information about missing serializations discovered by eMake. This includes information about which file caused two jobs to become serialized by the eMake history mechanism.
 - **Lookup**—Adds information about files that were looked up by each job. **Note:** *This mode can cause the annotation file to become quite large.*

- Registry—Adds information about registry operations.
- Waiting—Adds information about the complete dependency graph for the build.
- Annotation File—Specifies the annotation file (`--emake-annofile`). Required if annotation detail is set. Use folder button to select a file.
- Annotation Upload—Determines whether to upload the annotation file to the Cluster Manager (`--emake-annoupload`).

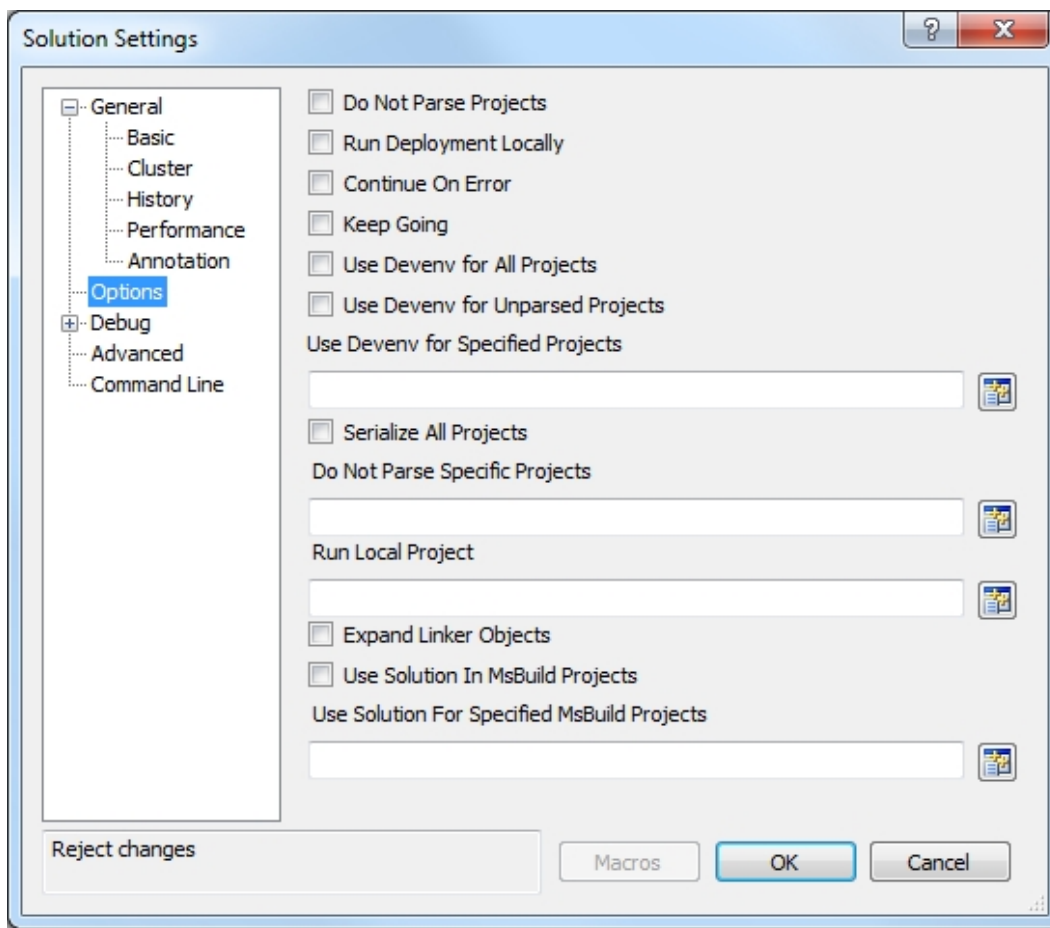
Note: If your local eMake version is 8.1 or newer, and if you select no Annotation Detail checkboxes, the add-in sets the `--emake-annodetail` eMake option to `none`. In this case, if the version of eMake is older than 8.1 on the *agent* machines, the following error appears at build time:

```
Starting build: <build number>
ERROR EC1007: Unknown annotation detail token: "none"
Valid tokens are: 'basic', 'env', 'file', 'history', 'lookup', 'md5', 'registry', 'waiting'
```

To avoid this error, you must upgrade the agents to EA 8.1 or newer or check at least one Annotation Detail checkbox.

Options Solution Settings

This category contains most frequently used optional settings.



- **Do Not Parse Projects**—Determines whether to prevent the VS Converter Add-In from breaking up C++ projects (sets `ECADDIN_DONT_PARSE_PROJECTS=true` if enabled).
- **Run Deployment Locally**—Determines whether to run deployment projects locally using `#pragma runlocal` (sets `ECADDIN_RUN_DEPLOYMENT_PROJECTS_LOCALLY=true` if enabled).
- **Continue On Error**—Determines whether to ignore errors that occur during the build (adds `/I` to the call to `eMake`).
- **Keep Going**—Determines whether to keep going when an error occurs during the build (adds `/k` to the call to `eMake`).
- **Use Devenv for All Projects**—Determines whether to run all projects using `devenv`, not `MSBuild` (sets `ECADDIN_USE_DEVENV=true` if enabled).
- **Use Devenv for Unparsed Projects**—Determines whether to use `devenv` for unparsed projects (sets `ECADDIN_USE_MSBUILD=true`).
- **Use Devenv for Specified Projects**—Run specified projects using `devenv`, not `MSBuild` (sets `ECADDIN_USE_DEVENV_FOR_PROJECTS`). You can either type in the project names (separated by semicolons and without quotes or white spaces) or click the corresponding button to browse for projects.
- **Serialize All Projects**—Determines whether to serialize all projects using `#serialize` (sets `ECADDIN_SERIALIZE=true` if enabled).
- **Do Not Parse Specific Projects**—Prevents the VS Converter Add-In from breaking up specified C++ projects (sets `ECADDIN_DONT_PARSE_PROJECT`). You can either type in the project names (separated by semicolons and without quotes or white spaces) or click the corresponding button to browse for projects.

This variable is useful for deploying the add-in. If, for any reason, the add-in cannot build some of your projects, this variable lets you work around the problem.

- **Run Local Project**—Runs specified projects locally using `#pragma runlocal` (sets `ECADDIN_RUN_LOCAL_PROJECT`). You can either type in the project names (separated by semicolons and without quotes or white spaces) or click the corresponding button to browse for projects.

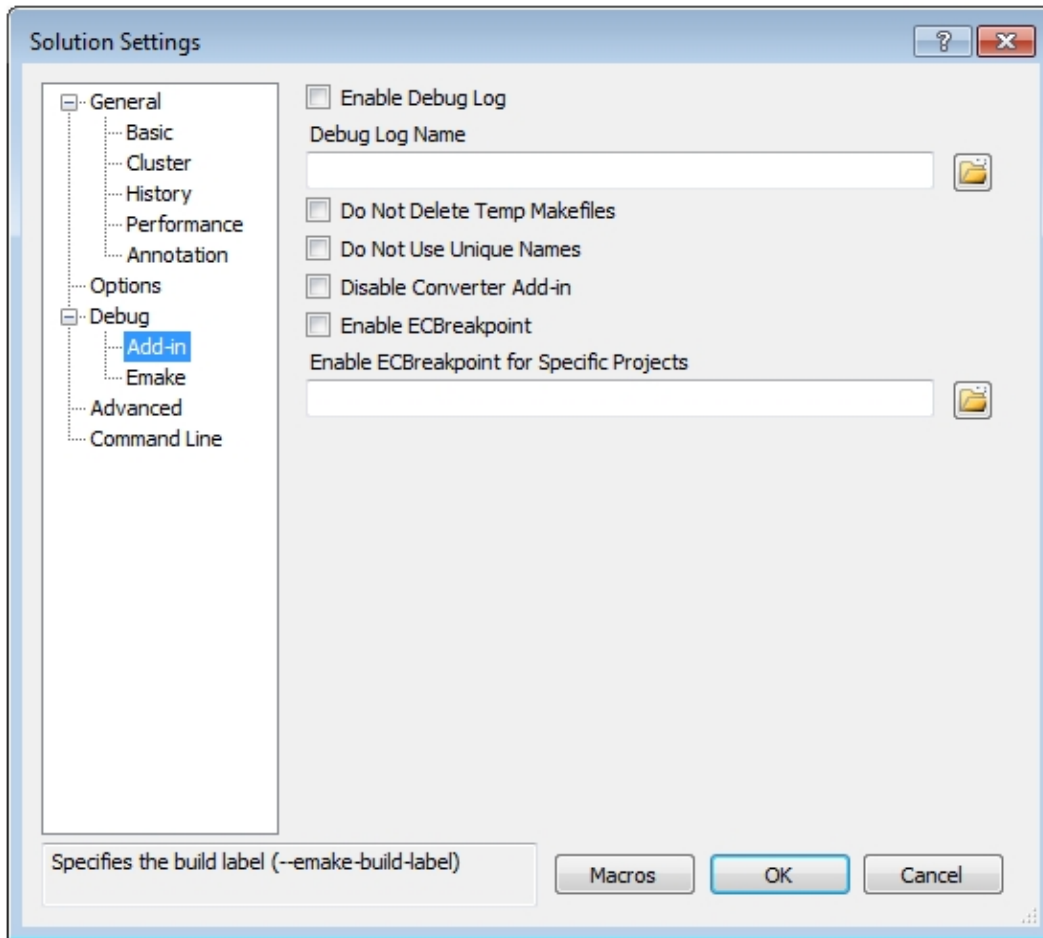
Use this variable if your build uses a local resource (for example, a resource only on the `eMake` host (for example, a database). You do not need to set this variable if your project build includes web deployment; this is handled by the add-in. Each project name must be the unique Visual Studio identifier for the project (for example, `solution1/project1.vcproj`).

- **Expand Linker Objects**—Determines whether to expand linker objects to full pathnames (sets `ECADDIN_EXPAND_LINKER_OBJECTS=true` if enabled).
- **Use Solution In MsBuild Projects**—Modifies the call to `MSBuild` to build a project within the solution context (sets `ECADDIN_BUILD_PROJECTS_IN_SOLUTION_CONTEXT=true` if enabled). This option overrides the **Use Solution For Specified MsBuild Projects** option. This option is needed for unparsed projects (such as C# projects or unparsed C++ projects).
- **Use Solution For Specified MsBuild Projects**—Modifies the call to `MSBuild` to build specific projects within the solution context (sets `ECADDIN_BUILD_SPECIFIC_PROJECTS_IN_SOLUTION_CONTEXT` accordingly). This option is needed for unparsed projects (such as C# projects or unparsed C++ projects). You can either type in the project names (separated by semicolons and without quotes or white spaces) or click the corresponding button to browse for projects.

Debug Solution Settings

This category contains debug options for the [Add-In Solution Settings](#) and [eMake Solution Settings](#).

Add-In Solution Settings



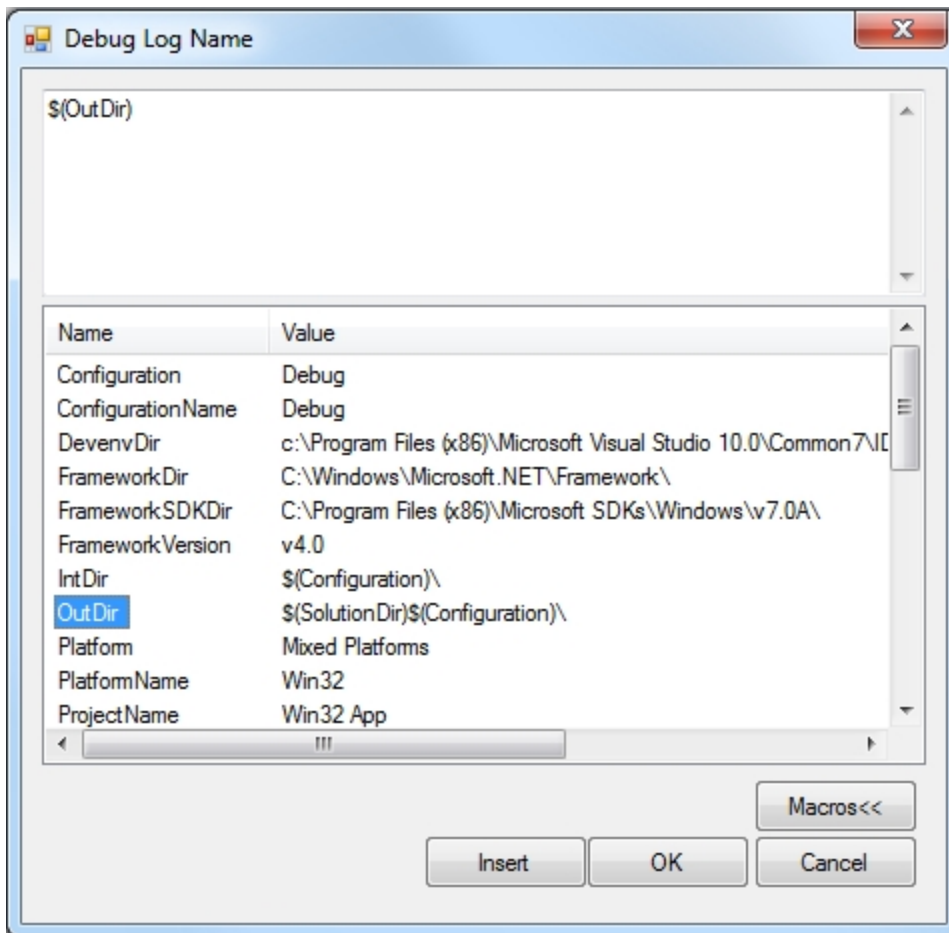
- **Enable Debug Log**—Determines whether to enable debug logging (sets `ECADDIN_DEBUG=true` if enabled).
- **Debug Log Name**—Specifies the name of the debug log (sets `ECADDIN_DEBUG_LOG_FILENAME`). The default location is `%TMP%\ecdebug<unique>.log` (where `%TMP%` is usually `C:\Users\<username>\AppData\Local\Temp`). Use the folder button to select a log file.
- **Do Not Delete Temp Makefiles**—Determines whether to delete temporary makefiles when the build completes (sets `ECADDIN_DONT_RM_TMP_MAKEFILES=true` if enabled).
- **Do Not Use Unique Names**—Determines whether to use unique names for temporary files (sets `ECADDIN_DONT_USE_UNIQUE=true` if enabled).
- **Enable ECBreakpoint**—Determines whether to invoke `ecbreakpoint` in failed jobs.
- **Enable ECBreakpoint for Specific Projects**—Invokes `ecbreakpoint` for specified projects. Use the folder button to select projects and delimit projects with a semi-colon.

Using Macros in the Solution Settings String-Based Fields

In any string-based field (such as the path name you enter into the **Debug Log Name** field) in the Electric Cloud solution settings, you can use a custom build command instead of a hardcoded string. For example, you can

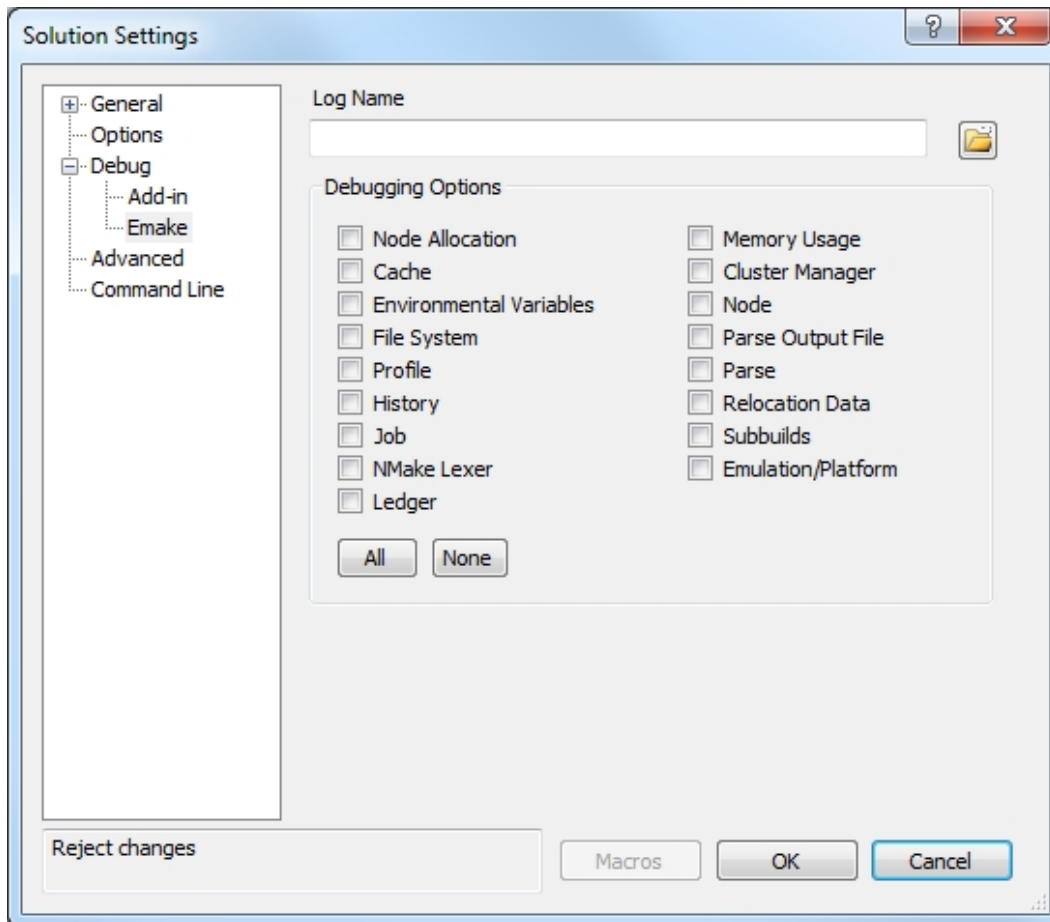
create a command that generates separate debug logs if you are running separate builds where each build has a different configuration.

Following is an example of a dialog box for entering macros:



To edit a set of macros for a field, click in the field, then click the **Macros** button to invoke the macro editor for that field, then click the **Macros<<** button to view the list of available macros.

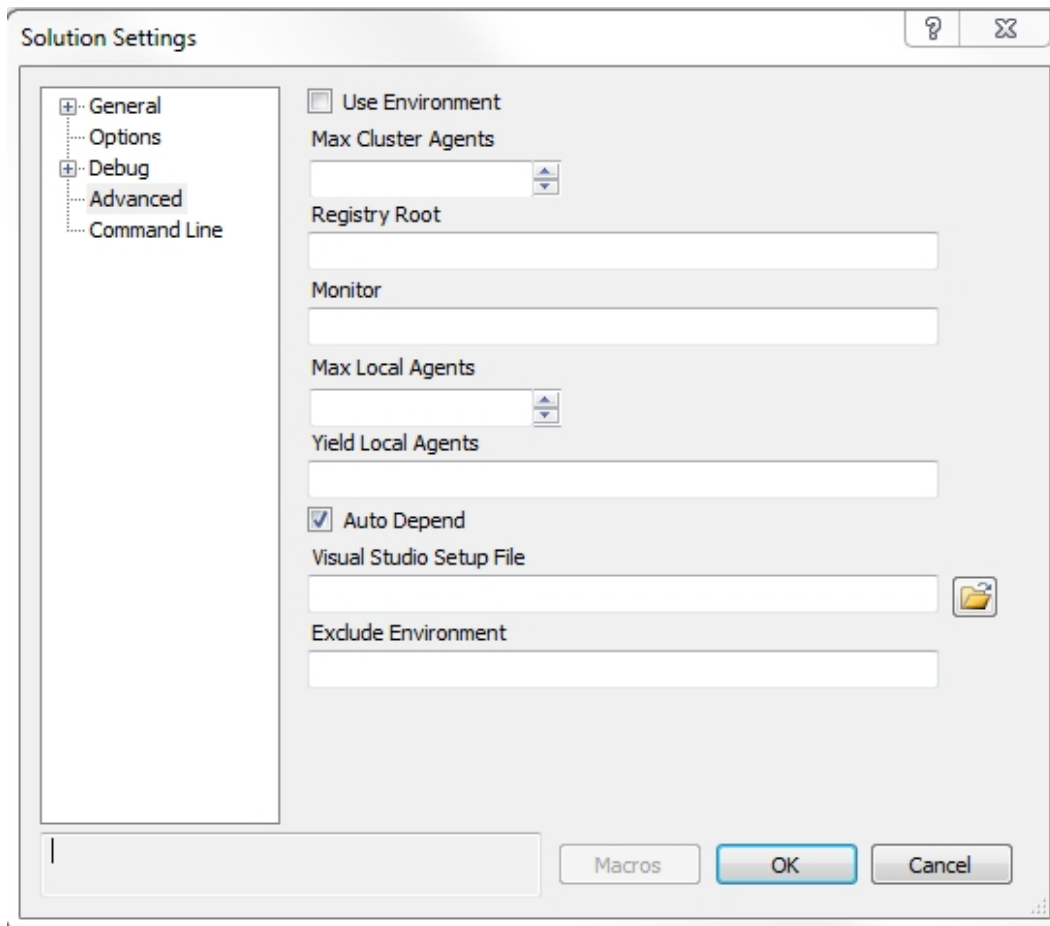
eMake Solution Settings



- **Log Name**—Specifies the name of the debug log (sets `--emake-logfile`). Use the folder button to select a log file.
- **Debugging Options**—Select the log levels for the eMake debug log file. For details about the eMake debug log file and log levels, see the “Electric Make Debug Log Levels” section in the *Troubleshooting* chapter of the *ElectricAccelerator Electric Make User Guide* at http://docs.electric-cloud.com/accelerator_doc/AcceleratorIndex.html.

Advanced Solution Settings

This category contains advanced options.



- **Use Environment**—Determines whether to add `/useenv` to the `devenv` call.
- **Max Cluster Agents**—Specifies the maximum number of agents to use during the build (`--emake-maxagents`).
- **Registry Root**—Specifies the registry root (`--emake-reg-roots`). You can specify multiple roots separated by ':' (a colon).
- **Monitor**—Allows the build to be monitored by ElectricInsight (`--emake-monitor`).
- **Max Local Agents**—Sets the maximum number of local agents to use (`--emake-localagents`).
- **Yield Local Agents**—If using more than N local agents, then eMake releases the number agents over N every T seconds so they can be used by another eMake that is looking for local agents (`--emake-yield-localagents=N, T`). Two values are required in this format: *release agents over this number, every this number of seconds*.

- Auto Depend—Enables or disables allowing eMake to determine dependencies automatically. Default is enabled.

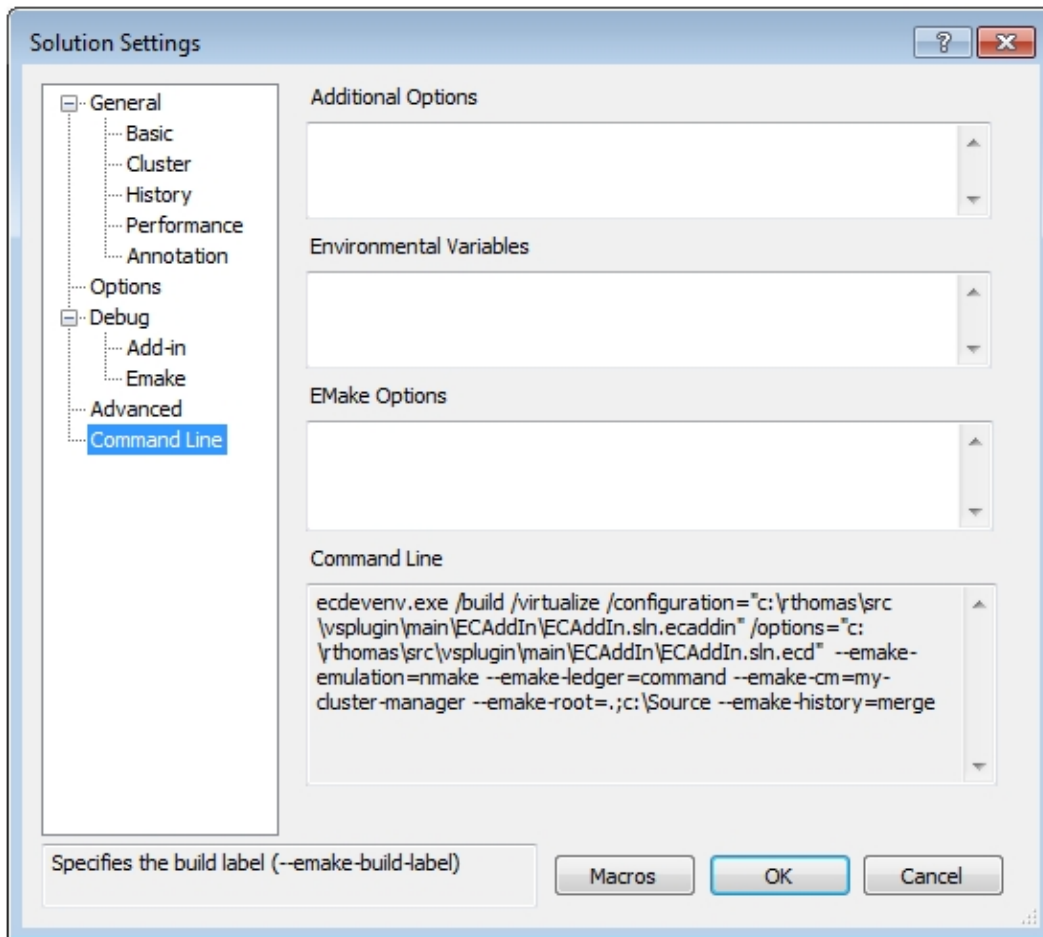
Note that Auto Depend is now enabled by default in ecdevenv when ecdevenv is invoked from the Visual Studio command prompt. (When Auto Depend is disabled in the IDE Add-In GUI, this default is overridden when ecdevenv is called from the GUI.)

For details about Auto Depend (also called eDepend), see the *Dependency Management* chapter in the *ElectricAccelerator Electric Make User Guide* at http://docs.electric-cloud.com/accelerator_doc/AcceleratorIndex.html.

- Visual Studio Setup file—Specifies the Visual Studio setup file for command line builds. Default is `vsvars32.bat`.
- Exclude Environment—Specifies a list of environment variables to exclude from eMake (`--make-exclude-env`), separated by ':' [a colon].

Command Line Solution Settings

This category lets you add additional options not explicitly specified elsewhere.



- **Additional Options**—Specifies a list of add-in options in this format: `<Name=Value>`. These are the same options specified on [Setting VS Converter Add-In Environment Variables](#)
- **Environmental Variables**—Specifies a list of environment variables in this format: `<variable>=<value>`, separated by a carriage return. Do not use “set”.
- **EMake Options**—Specifies a list of eMake options in this format: `--emake-<option>=<value>`, separated by a carriage return.
- **Command Line**—A non-editable field that shows the `ecdevenv` command that is executed by the add-in. VS Converter Add-In options are stored in the `/options` file.

Chapter 7: Using the ecdevenv Utility

ecdevenv.exe is a command-line drop-in replacement for devenv.exe that builds Visual Studio solutions and projects using eMake. Following are the key ecdevenv features:

- Skip generation of NMAKE makefiles
- No makefiles are required
- Visual Studio toolchain virtualization
- Ability to build multiple solutions and projects in one command
- Ability to specify global add-in options in an options file
- Forced regeneration of makefiles if required
- Ability to generate makefiles without running eMake

The pre-4.0 behavior of ecdevenv is still supported in versions 4.0 and later.

About ecdevenv

No Makefiles are Required

Without ecdevenv, you must create a makefile to build using eMake:

```
all:
    devenv.exe solution.sln /build Debug
```

Then call this makefile with eMake:

```
emake.exe --emake-cm=<your cm> --emake-emulation=nmake -f makefile
```

With ecdevenv, you can simply use the following command:

```
ecdevenv.exe solution.sln /build Debug --emake-cm=<your cm>
```

ecdevenv converts the solution into NMAKE makefiles and runs eMake on them using the eMake parameters specified. Devenv and eMake arguments can be in any order. ecdevenv automatically uses NMAKE emulation.

ecdevenv always converts the solution locally, so any differences between the eMake machine and agent machines can be ignored.

Note: When invoking ecdevenv, if you build a solution that was created in another version of Visual Studio, a warning appears at the command prompt. For example: Warning: 'MySolution.sln' version (2012) does not match version in use (2005).

Miscellaneous Files that ecdevenv Generates

ecdevenv creates several files in the build directory. You do **not** need to check these files in to your revision control system, unless you want to skip regeneration of NMAKE makefiles.

- *.eccache

"*.eccache" files are created in the same directory as the solution files. One file is created for each invocation of ecdevenv. ecdevenv uses this file to determine whether it needs to regenerate the temporary makefiles. Delete these only if you want to force ecdevenv to regenerate makefiles.

- metrics.sln.Release_Win32.ecmak and metrics.vcxproj.Release_Win32.ecmak

These are temporary makefiles generated by ecdevenv. Do not delete them unless you are regenerating the temporary makefiles.

- metrics_Release_Win32.sln

This is a copy of the original solution used to remove project dependencies. This was generated by ecdevenv (note the configuration name in the file name) Do not delete it unless you are regenerating the temporary makefiles.

- metrics.sln, metrics.vcxproj, metrics.vcxproj.filters, metrics.vcxproj.user

These are Visual Studio files. Do not delete them.

- ecdevenv.mak

This is the top-level makefile created by ecdevenv. Do not delete it.

- .roots

This contains the list of generated eMake roots. Do not delete it.

You can relocate these files by using the `ECADDIN_MAKEFILE_CACHE` environment variable. For details, see the [Relocating Makefiles Generated by the Add-In](#) section in the [Setting VS Converter Add-In Environment Variables](#) chapter.

Virtualizing the Visual Studio Toolchain

Use the `/virtualize` option to virtualize the Visual Studio toolchain. This virtualizes the Visual Studio installation directory, SDK directory, and relevant registry entries. This negates the need to install Visual Studio on the agent machines. Make sure that you have installed the following software before using this option:

- Relevant .NET version that you are using
- Relevant redistributable for your version of Visual Studio

Building Multiple Solutions and Projects

ecdevenv can build multiple solutions. To use this capability, create a makefile in the following format and (optionally) specify the projects and project configurations:

```
<BuildSpecification>
  <Solution>
    <Name>Solution1.sln</Name>
    <Platform>Mixed Platforms</Platform>
    <Configuration>Debug</Configuration>
    <Project>
      <Name>Project1\Project1.vbproj</Name>
      <Platform>Any CPU</Platform>
```

```

        <Configuration>Debug</Configuration>
    </Project>
    <Project>
        <Name>Project2\Project2.vcproj</Name>
        <Platform>Win32</Platform>
        <Configuration>Debug</Configuration>
    </Project>
</Solution>
<Solution>
    <Name>Solution2.sln</Name>
    <Platform>Mixed Platforms</Platform>
    <Configuration>Debug</Configuration>
</Solution>
</BuildSpecification>

```

Then pass the name of the file to ecdevenv using `/configuration=<filename>`.

Note: If you use the `/configuration` option, you cannot specify a solution, project, or project configuration on the command line.

Enabling Debugging Output

- Set `ECDEVENV_DEBUG=true` to turn on debugging (output is sent to stdout by default).
- Set `ECDEVENV_DEBUG_LOG=<filename>` to redirect output to a file.

Using ecdevenv Command Options

Option	Description
<code>/configuration</code>	Specifies the configuration file that allows multiple solutions and projects to be built. See the Building Multiple Solutions and Projects section above.
<code>/force</code>	Forces the regeneration of makefiles.
<code>/generate</code>	Generates the NMAKE makefiles without running eMake.
<code>/help</code>	Displays a list of ecdevenv options and sample options and configuration files.
<code>/makefile</code>	Specifies an alternative default makefile (rather than <code>ecdevenv.mak</code>).
<code>/options=<file></code>	Specifies the add-in options. The file uses the format shown in the Setting VS Converter Add-In Environment Variables chapter.

Option	Description
<code>/quick</code>	Converts the solution to NMAKE more quickly by parallelizing down to the project level only. For details, see the Using Fast Solution Conversion section below.
<code>/skip</code>	Never regenerates makefiles. You use this when you have cached makefiles. You must manually regenerate makefiles using <code>/force</code> when required.
<code>/use64bit</code>	Specifies the 64-bit version of eMake.
<code>/version</code>	Displays the ecdevenv version number.
<code>/virtualize</code>	Virtualize the Visual Studio toolchain (off by default). See the Virtualizing the Visual Studio Toolchain section above.

Using ecdevenv Environmental Variables

Variable	Description
ECDEVENV_DEBUG	Turns on debugging to stdout.
ECDEVENV_DEBUG_LOG	Redirects the debug output to a specified file.
ECDEVENV_NO_EXPLICIT_DEPS	Does not use project-to-project dependencies.
ECDEVENV_RUN_LOCAL	Specifies a list of projects to locally build <code>#pragma runlocal</code> . All project lists are delimited with semi-colons. See the add-in option <code>ECADDIN_RUN_LOCAL_LINK</code> .
ECDEVENV_SKIP_CHECKS	Skips checking for <code>devenv.com</code> in the <code>PATH</code> environment variable.
ECDEVENV_SKIP_DUPLICATE_PROJECTS	Skips projects if they have been previously built in a different solution.
ECDEVENV_USE_SOLUTION_CONTEXT	Builds all projects in the solution context. See the Debugging a Failed Build section.
ECDEVENV_USE_SOLUTION_CONTEXT_FOR_PROJECTS	Specifies a list of projects to build in the solution context. All project lists are delimited with semi-colons. See the Debugging a Failed Build section.

Using Fast Solution Conversion

Use `/quick` for fast solution conversion. This option converts the solution to NMAKE more quickly by parallelizing down to the project level only. The option is best suited to solutions containing no C++ projects.

When using the `/quick` option, you must also specify the full platform and configuration. For example, `ecdevenv solution.sln /build "Release|Win32" /quick`.

The option is available only with Visual Studio 2005 and newer versions. If you try to use the option when building a solution in Visual Studio 2002 or Visual Studio 2003, the following error message appears: `Error: /quick can only be used with VS2005 and higher`.

Note: You can also enable fast solution conversion in the IDE add-in by checking the **Coarse Grain Parallelization** checkbox in the **General > Performance** tab of the Electric Cloud **Solution Settings** dialog box.

Visual Studio does not need to be installed for you to use fast solution conversion. However, MSBuild must be installed.

Chapter 8: Using the VS Converter Add-In

ElectricAccelerator can build Visual Studio solutions in two different modes:

- Use `ecdevenv` as a drop-in replacement for your command-line build (recommended)
- Create a makefile containing the `devenv` calls

Using `ecdevenv`

If you choose to use `ecdevenv`, replace:

```
devenv.com Solution.sln /build Debug
```

with:

```
ecdevenv Solution.sln /build Debug --emake-cm=<yourcm> /virtualize
```

`ecdevenv` does the following:

1. Converts `Solution.sln` to NMAKE format.
2. Calls `eMake` on the generated files.

The `/virtualize` flag virtualizes the Visual Studio toolchain, negating the need to install Visual Studio on the agents. You must, however, ensure the relevant versions of .NET and redistributables are installed.

`devenv` must be in the `PATH` environment variable **before** executing `ecdevenv`.

Creating a Makefile

Before you can use Accelerator to build your Visual Studio project, make sure you have already installed and run Visual Studio on each agent host for each user (all *ECloudInternalUsers*).

Note: Virtualization of the toolchain is not possible when using this method.

If you currently invoke Visual Studio from inside a makefile, you are ready. If you invoke Visual Studio directly from the command line or through a batch file, you must create a makefile for `eMake` to run. For example:

```
all:
    devenv /build Release foo.sln

-- or --

all:
    devenv /build Release foo.sln /project bar.vcproj
```

The makefile must invoke `devenv` with the options you currently use. Ensure the correct version of `devenv` is in your path:

```
devenv /?
```

and ensure that the usual Visual Studio environment variables are set.

Setting the Path for 64-Bit or Xbox Builds

To run 64-bit or Xbox builds, you must set the path manually.

Chapter 9: Setting VS Converter Add-In Environment Variables

Setting the Converter Add-In Environment Variables

You can control the way the VS Converter Add-In works by setting these environment variables on the eMake machine.

Alternatively, you can set the variables in a configuration file and set `ECADDIN_CONFIGURATION_FILE` to the full or relative pathname.

```
all:
    set ECADDIN_CONFIGURATION_FILE=addin.cfg
    devenv.com Solution.sln /build Debug
```

Sample `addin.cfg`:

```
<SolutionSettings>
  <DisableAddin>>false</DisableAddin>
  <DoNotParseProjects>>false</DoNotParseProjects>
  <ECBreakpoint>>false</ECBreakpoint>
  <DoNotParseSpecificProjects />
  <ECBreakpointProjects />
  <RunLocalLink />
  <UseDevenvForProject />
  <EnableDebugLog>>false</EnableDebugLog>
  <DebugLogName>ecdebug.log</DebugLogName>
  <MaxPDBFiles>16</MaxPDBFiles>
  <UseOrderOnlyPrereqs>true</UseOrderOnlyPrereqs>
  <EnableIncrementalLink>>false</EnableIncrementalLink>
  <RemoveDependencies>true</RemoveDependencies>
  <ForceZ7>true</ForceZ7>
  <UpToDateCheck>>false</UpToDateCheck>
  <DoNotDeleteTempMakefiles>>false</DoNotDeleteTempMakefiles>
  <DoNotUseUniqueNames>>false</DoNotUseUniqueNames>
  <ExpandLinkerObjects>>false</ExpandLinkerObjects>
  <RunDeploymentLocally>>false</RunDeploymentLocally>
  <RunLocalProject />
  <SerializeAllProjects>>false</SerializeAllProjects>
  <UseDevenv>>false</UseDevenv>
  <UseMSBuild>true</UseMSBuild>
</SolutionSettings>
```

All environment variables override variables in the configuration file, so you can set them outside of the configuration file as needed.

Note: Environment variables that take Boolean values can accept: “0”, “no”, “false”, “off”, “1”, “yes”, “true”, or “on”. Case is not significant.

Environment Variable	Configuration File Variable	Description	Default
-	AddEmakeRootsAutomatically	<p>Adds the locations of all inputs (such as .cpp and .h files) and outputs (such as .obj, .exe, and .dll files) to the eMake root automatically.</p> <p>This variable applies only to C++ projects. For a C# project or a project with inputs and outputs not in the solution or project locations, you must add their locations to the eMake root manually.</p> <p>This variable does not apply to third-party tools, because they cannot be virtualized. It also does not apply to project locations and each location for the solution(s), because they are always added automatically.</p>	<p>False (convert er add-in)</p> <p>True (ecdev env and IDE add-in)</p>
ECADDIN_BUILD_ORDER	BuildOrder	Specifies projects' build order. Use only if Accelerator is not building projects in the correct order.	-
ECADDIN_BUILD_PROJECTS_IN_SOLUTION_CONTEXT	BuildProjectsInSolutionContext	Modifies the call to MSBuild to build a project within the solution context. This overrides the ECADDIN_BUILD_SPECIFIC_PROJECTS_IN_SOLUTION_CONTEXT variable and the BuildSpecificProjectsInSolutionContext configuration file variable. This option is needed for unparsed projects (such as C# projects or unparsed C++ projects).	False

Environment Variable	Configuration File Variable	Description	Default
ECADDIN_BUILD_SPECIFIC_PROJECTS_IN_SOLUTION_CONTEXT	BuildSpecificProjectsInSolutionContext	Allows a list of projects to be built in the solution context in MSBuild. Supply a list of projects (separated by a semicolon but no spaces). This option is needed for specific unparsed projects (such as C# projects or unparsed C++ projects).	-
ECADDIN_CHECK_DLLEXPORT	CheckDLLExport	Prevents the linker from including libraries that do not contain any exports. This might be slow.	False
ECADDIN_CONFIGURATION_FILE	-	Sets the filename of the configuration file. Environment variables override the settings in this file.	-
ECADDIN_CREATE_MISSING_DEPENDENCIES	CreateMissingDependencies	Creates missing dependencies to avoid missing dependency warnings.	False
ECADDIN_DEBUG	EnableDebugLog	Setting this variable to any value causes debug log files to remain in %TMP%\ecdebug<ID>.log on the agent host. These files are used for troubleshooting by Electric Cloud engineers. Normally, you do not need to set this value.	False
ECADDIN_DEBUG_LOG_FILENAME	DebugLogName	Specifies the debug log name. Requires ECADDIN_DEBUG. Use '\$1' in the file specification to insert a unique ID. For example, C:\Users\Bill\ecdebug_\$1.log. Use a file location outside of eMake root. The log file is stored on the agent.	-
ECADDIN_DISABLE_MINIMAL_REBUILDS	DisableMinimalRebuilds	Disables minimal rebuilds.	False
ECADDIN_DISALLOW_BSC	DisallowBSC	Does not generate browse information files.	False

Environment Variable	Configuration File Variable	Description	Default
ECADDIN_DISABLE_PCH	DisallowPCH	Does not generate/use precompiled header files (implied by ECADDIN_MAX_PDB_FILES)	False
ECADDIN_DISABLE_PDB	DisallowPDB	Does not generate PDB files.	False
ECADDIN_DISABLE_SBR	DisallowSBR	Does not generate browse information files from sources.	False
ECADDIN_DONT_ADD_PCH_LOCATION	DoNotAddPCHLocation	Prevents the add-in from adding the location of the PCH file in all cases. This variable is relevant only if ECADDIN_MAX_PDB_FILES or ECADDIN_DISABLE_PCH is switched on.	False
ECADDIN_DONT_ALLOW_PCH_AND_PDB	DoNotAllowPCHAndPDB	Switches off PDB and PCH generation.	False
ECADDIN_DONT_PARSE_PROJECT	DoNotParseSpecificProjects	<p>Prevents the VS Converter Add-In from breaking up specified C++ projects. This variable takes a list of project names separated by semi-colons and without white spaces. This variable is useful for deploying the add-in. If, for any reason, the add-in cannot build some of your projects, this variable lets you work around the problem.</p> <p>When using this variable, you might experience an the warning MSB4098. You can ignore this warning, because any project references are now converted into additional dependencies. MSBuild, however, does not provide a mechanism to turn off this warning.</p>	-

Environment Variable	Configuration File Variable	Description	Default
ECADDIN_DONT_PARSE_PROJECTS	DoNotParseProjects	This variable takes any non-blank value and its behavior is similar to ECADDIN_SERIALIZE. It calls devenv on each project (the add-in does not convert each project into individual compile/link steps).	-
ECADDIN_DONT_RM_TMP_MAKEFILES	DoNotDeleteTempMakefiles	Retains makefiles created during the build but normally deleted when the build finishes. This environment variable can have any value; it just needs to be set.	False
ECADDIN_DONT_RUN	DoNotRun	TEST only. Convert makefiles without running eMake.	False
ECADDIN_DONT_USE	DisableAddin	Disables the add-in. This environment variable can have any value, it just needs to be set. Also, you can disable the add-in on each host by using the Visual Studio Add-In Manager (on the Tools menu). Note: This is a “light-weight” uninstall program that disables one individual machine at a time.	False
ECADDIN_DONT_USE_UNIQUE	DoNotUseUniqueNames	Does not use unique names for temporary makefiles. Use with ECADDIN_DONT_RM_TMP_MAKEFILES.	False
ECADDIN_ECBREAKPOINT	ECBreakpoint	Determines whether to invoke ecbreakpoint on failed jobs.	False
ECADDIN_ECBREAKPOINT_PROJECTS	ECBreakpointProjects	Determines whether to invoke ecbreakpoint for specified projects. Use a semi-colon to delimit projects.	-
ECADDIN_ENABLE_INCREMENTAL_LINK	EnableIncrementalLink	Inserts a call to <code>ectouch.exe</code> . See Tuning Performance .	True

Environment Variable	Configuration File Variable	Description	Default
ECADDIN_EXPAND_LINKER_OBJECTS	ExpandLinkerObjects	Expands linker objects to one line per object. Prevents errors when link line length is exceeded.	False
ECADDIN_FORCE_Z7	ForceZ7	Enables /Z7 compiler options for all C++ files.	False
ECADDIN_INCLUDE_CMAKELISTS	IncludeCMakeLists	Excludes any source file with the name CMakeLists.txt. Set this variable to True to execute the file.	False
ECADDIN_INJECT_PCH_REFERENCE	InjectPCHReference	Adds /YI<projectname> to the PCH creator (/Yc). Set it to <i>false</i> only if linker tools error LNK2005 (multiply-defined symbol error) appears.	True
ECADDIN_MAKEFILE_CACHE	MakefileCache	Path to an alternate location for the cached (temporary) makefiles that are generated by the add-in. For details, see the Relocating Makefiles Generated by the Add-In section below.	-
ECADDIN_MAX_PDB_FILES	MaxPDBFiles	Specifies the maximum number of PDB files produced. See Optimizing Parallelization Using PDB Splitting .	16
ECADDIN_MSBUILD_DIR	MSBuildDir	Path to the location of msbuild.exe if different from the default.	-
ECADDIN_MSBUILD_PARAMETERS	MSBuildParameters	Adds extra parameters to msbuild command line.	-
ECADDIN_NORMIMIZE_PATHS	NormalizePaths	Normalizes all paths in the makefile.	True
ECADDIN_REMOVE_DEPENDENCIES	RemoveDependencies	Removes project-to-project dependencies to improve parallelization.	False
ECADDIN_RUN_DEPLOYMENT_PROJECTS_LOCALLY	RunDeploymentLocally	Runs deployment projects locally using #pragma runlocal.	False

Environment Variable	Configuration File Variable	Description	Default
ECADDIN_RUN_LOCAL_LINK	RunLocalLink	A list of projects where the linker will be run locally (using <code>#pragma runlocal</code>).	-
ECADDIN_RUN_LOCAL_PROJECT	RunLocalProject	Use this variable if your build uses a local resource (for example, a resource only on the eMake host (for example, a database). You do not need to set this variable if your project build includes web deployment; this is handled by the add-in. The value of this variable is a list of project names separated by semi-colons. Each project name must be the unique Visual Studio identifier for the project (for example, <code>solution1/project1.vcproj</code>). Do not add quotation marks or white spaces.	-
ECADDIN_SERIALIZE	SerializeAllProjects	Causes each project to be built serially. It inserts <code>'#pragma allserial'</code> into each makefile. This variable is equivalent to setting <code>ECADDIN_DONT_PARSE_PROJECTS</code> .	False
ECADDIN_UP_TO_DATE_CHECK	UpToDatecheck	Pre-parses the projects to determine whether there is anything to build. Prevents unnecessary rebuilding of static build steps.	True
ECADDIN_USE_DEVENV	UseDevenv	Uses <code>devenv</code> (rather than <code>msbuild</code>) for all unparsed projects.	False
ECADDIN_USE_DEVENV_FOR_PROJECT	UseDevenvForProject	Uses <code>devenv</code> (rather than <code>msbuild</code>) to build specific projects. Supply a list of projects (separated by a semicolon) to be built with <code>devenv</code> .	-

Environment Variable	Configuration File Variable	Description	Default
ECADDIN_USE_LEGACY_CODE	UseLegacyCode	Use this variable to workaround a Visual Studio bug where AdditionalLibraryDirectories does not give the correct value.	False
ECADDIN_USE_MSBUILD	UseMSBuild	Lets you use msbuild internally for projects that the add-in cannot parse.	True
ECADDIN_USE_ORDER_ONLY_PREREQS	UseOrderOnlyPreReqs	Uses order-only prerequisites (available in Accelerator v7.0 and later). This allows for quicker first time (no history) builds.	True
ECADDIN_USE_RELATIVE_PATHS	UseRelativePaths	Uses relative paths in the makefile to reduce line lengths.	False
ECADDIN_USE_WCE_MACROS	UseWCEMacros	Loads platform macros.	False
ECADDIN_XBOX_INSTALL_DIR	XBoxInstallDir	Path to the location of Xbox SDK if different from the default.	-
ECADDIN_XBOX_VERSION	XBoxVersion	Xbox SDK version if different from the default.	-

Relocating Makefiles Generated by the Add-In

Generated (temporary) makefiles include .ecmak files, solutions and projects, RC files, and up-to-date check files. By default, these files are in the same location as their corresponding solution or project. For example:

- C:\Test\solution_001\solution_001.sln **generates** C:\Test\solution_001\solution_001.sln.ecmak
- C:\Test\solution_001\project1\project1.vcxproj **generates** C:\Test\solution_001\project1\project1.vcxproj.ecmak

You can specify a path to an alternate location for these files by setting the ECADDIN_MAKEFILE_CACHE environment variable. For example, if you enter `set ECADDIN_MAKEFILE_CACHE=C:\temp`, then

- C:\Test\solution_001\solution_001.sln **generates** C:\temp\c\Test\solution_001\solution_001.sln.ecmak
- C:\Test\solution_001\project1\project1.vcxproj **generates** C:\temp\c\Test\solution_001\project1\project1.vcxproj.ecmak

(Note that the drive colon is replaced with a directory separator in the cache.) This variable is useful when you want to keep eMake-generated files out of the source tree or when the source tree is read-only.

Note: If you use this environment variable, keep in mind that the paths for the generated files must not exceed the Windows character limit.

Setting the Environment Variable to Enable Local Solution Builds

You can build a solution locally with eMake but without using remote agents or local agents. You might want to use this function if a distributed incremental build is slow, or if a local Visual Studio incremental build causes unnecessary rebuilding of objects. To make this function visible in the **Electric Cloud** menu, set the environment variable `ECUIADDIN_LOCAL_BUILD=true`. For more information about this menu, see the [Main Menu and Toolbar](#) section.

Setting the Environment Variable for ecdevenv Startup Checks

By default, the ecdevenv utility perform several checks (such as the existence and proper version number of the converter add-in) when it starts. To disable the checks, set the environment variable `ECDEVENV_SKIP_CHECKS=true`.

Chapter 10: Tuning Performance

The add-in has several methods for improving performance. To determine which is best for your situation, generate an annotation file and open it in ElectricInsight.

To generate an annotation file, pass `--emake-annodetail=basic, file, lookup, env` to your eMake call. By default, the annotation file is named `emake.xml`.

Available methods:

- [Improving Build Time for /Zi + PCH Builds](#)
- [Improving Build Time for Solutions with Many Projects](#)
- [Improving Final Link Time](#)
- [Improving Incremental Build Time](#)
- [Improving Incremental Linking Time](#)
- [Optimizing Parallelization Using PDB Splitting](#)

Improving Build Time for /Zi + PCH Builds

The default configuration for VC++ projects is /Zi and using PCH. To parallelize this combination, the add-in splits PDB and duplicates PCH. However PCH files are usually very large and might negate any improvement parallelization offers.

To improve build time in these circumstances:

1. Set `ECADDIN_FORCE_Z7=true`

This is the single most effective way to improve build speed.

2. Set `ECADDIN_DISALLOW_PCH`

This turns off PCH but might result in build failures that can be fixed in code only.

3. Reduce `ECADDIN_MAX_PDB_FILES`

Reducing this setting reduces parallelism but decreases the time spent copying PCH files.

Improving Build Time for Solutions with Many Projects

Some very large solutions with few inter-project dependencies might benefit from not parsing the project down to the project item level. Follow these steps:

1. Set `ECADDIN_DONT_PARSE_PROJECTS=true`
2. Clear history.
3. Rebuild.

Although you lose fine-grain parallelism, the reduced overhead might reduce the overall build time.

Improving Final Link Time

Many typical solutions have a final link (or lib) that is very large and slow on the cluster. To perform this link locally, set `ECADDIN_RUN_LOCAL_LINK=<project>`.

IMPORTANT: Running projects locally with `#pragma runlocal` might cause other issues. When running with `#pragma runlocal`, only changes in the current working directory are recognized by EFS, so it is not advised if there are subsequent jobs that use files outside of the CWD.

Improving Incremental Build Time

By default, the add-in does not rebuild prebuild events. Instead, the add-in first checks whether there is anything out of date. If not, nothing will be built, including the prebuild event.

To always rebuild prebuild events, set `ECADDIN_UP_TO_DATE_CHECK=false`. Note that when you set this environment variable to `false`, if you have a prebuild event that touches files, it could potentially rebuild far more than Visual Studio would.

Improving Incremental Linking Time

Improving Incremental Linking Time with the Add-In

Visual Studio supports incremental linking with the `/INCREMENTAL` linker option. This does not function in eMake, because eMake updates the time stamp of the exe/dll when it copies it back to the build machine (from the agent) to prevent any problems because of clock skew.

To work around this problem, the add-in “touches” the exe after the link with its current time stamp. This explicit modification of the time stamp instructs eMake to preserve the time stamp, which keeps the validity of its incremental status. The add-in inserts a call to `ectouch.exe`, which performs the action stated above. `ectouch.exe` must be in `%PATH%`.

To disable this feature with the add-in, set `ECADDIN_ENABLE_INCREMENTAL_LINK=false`.

Improving Incremental Linking Time without the Add-In

If you are not using the add-in, you can still use this feature. You can rename `ectouch.exe` to `eclink.exe` and replace occurrences of `link.exe` with `eclink.exe`. `eclink.exe` should be in `%PATH%`. Alternatively, you can rename `link.exe` to `link_ec.exe` and copy `eclink.exe` to `link.exe`. (If you want something other than `link_ec.exe`, set `EC_ORIGINAL_LINK_PATH` to the location of the “real” `link.exe`.)

Optimizing Parallelization Using PDB Splitting

Optimizing Parallelization with the Add-In

By default, Visual Studio puts all debugging information in a centralized database (PDB) called `vc80.pdb` (this is Visual Studio version-specific). Because each compilation modifies this file, everything in the project is serialized. A workaround is to group debug information into multiple PDB files. You can accomplish this automatically if you use the add-in.

`ECADDIN_MAX_PDB_FILES` is set to 16 by default. You can change this value to be equal to or less than the number of agents, but you might need to increase or decrease this for optimal efficiency. `ECADDIN_MAX_PDB_`

`FILES` specifies the maximum number of PDB files produced. Each file is placed into a PDB determined by a hash of its filename. This method ensures that a particular file is always placed in the same PDB. This is necessary to ensure eMake's history file remains valid.

For example, if a project contains 4 files, `File1.cpp`, `File2.cpp`, and so on, and they are all serialized on PDB file `vc80.pdb`. Set `ECADDIN_MAX_PDB_FILES=2` will create (at most) 2 PDB files:

```
File1.cpp --' <ProjectName>_0.pdb
File2.cpp --' <ProjectName>_1.pdb
File3.cpp --' <ProjectName>_0.pdb
File4.cpp --' <ProjectName>_1.pdb
```

In this example, `File1` and `File3` will be serialized against each other but will build in parallel from `File2` and `File4` (which will be serialized against each other).

You can change this variable in the Visual Studio IDE Add-In solution settings. Go to the Performance section of the Add-in pane.

The history file must be deleted when adding or changing the value of `ECADDIN_MAX_PDB_FILES`. You can also set `--emake-history=create`.

Optimizing Parallelization without the Add-In

This technique can be used without using the add-in. This distribution contains the application `hashstr.exe`, which hashes the filename and returns the bucket number. You can use this in your makefile to set the PDB filename (using `/fd`) in the same manner as above. Precompiled headers must be switched off for this to work.

Usage: `hashstr "mystring" [modulus]`

Where `mystring` is the string from which to generate the hash value, and `modulus` is the number of hash bins you want to use.

You can add this to a pattern rule for builds that suffer from performance degradation due to PDB serialization, with something similar to the following:

```
%.o: %.c
$(CC) /c $(cflags) $(PCH_USE_FLAGS) $(cvars) $(cplus_flags) $(LOCAL_INCLUDE) $(P
CB_INCLUDE) $< /Fo$@ /Fd$(shell ${path-to-hashstr}/hashstr.exe "$@" ${hashstr-mo
dulus}) .pdb
```


Chapter 11: Using MSBuild

Building MSBuild Projects in Parallel

ElectricAccelerator **cannot** parallelize MSBuild project files. If you have multiple MSBuild projects, however, you can create a makefile to build them in parallel.

For example:

```
all: project1 project2

project1
    msbuild myproject.csproj /t:build

project2:
    msbuild myproject.csproj /t:build
```

Then run:

```
emake -f makefile --emake-emulation=nmake --emake-cm<your cm>
```

For C++ projects, call devenv (or ecdevenv) to parallelize those projects down to the project item level.

If you use a top-level MSBuild script that builds separate projects, convert that to NMAKE in the format above to achieve parallelization under eMake.

Using MSBuild to Build a Project Directly in Its Solution Context

When set to true, the ECADDIN_USE_SOLUTION_IN_MSBUILD environment variable lets you use MSBuild to build a project directly, but in the context of its solution, by generating an appropriate call to be sent to MSBuild. For example, when you build a C# project named WindowsApplication2 with project configuration Debug|AnyCPU, the generated makefile contains the following call to MSBuild:

```
"msbuild.exe" "WindowsApplication2.csproj" /t:build /p:Configuration="Debug",Platform="AnyCPU"
```

(Note that MSBuild expects there to be no space between Any and CPU.)

When you set the environment variable to true, the command is changed to build the WindowsApplication2 project in solution configuration Debug|Mixed Platforms:

```
"msbuild.exe" "solution_0012.sln" /t:WindowsApplication2 /p:Configuration="Debug",Platform="Mixed Platforms"
```


Chapter 12: Debugging a Failed Build

Perform the following tasks first when debugging a failed build:

- Double-check that the build works under Visual Studio.
- See the [Troubleshooting Problems](#) chapter or ask.electric-cloud.com.

If the previous tasks do not help you debug your build, do the following steps:

1. Set `ECADDIN_DEBUG=true` and `ECADDIN_DEBUG_LOG_FILENAME=<filename>`.
2. Rerun the build.

The *<filename>* will exist on the machine that performed the conversion. When using `ecdev`, this will be the local machine. When running `dev` or `ecdev` remotely, the file will exist on the remote machine.

Chapter 13: Troubleshooting Problems

Initializing Visual Studio

If you don't virtualize the toolchain, you must initialize Visual Studio on *every agent* host for *each ECloudInternalUser*. Each Accelerator agent runs as user ECloudInternalUser1, ECloudInternalUser2, and so on.

Log in to each user account and run Visual Studio and do the following:

1. Choose **Tools > Options** and browse to **Project and Solutions > Build and Run**.
2. Set the maximum number of parallel project builds to 1.
3. Choose **Help > Customer Feedback Options**.
4. Initialize the Customer Experience Improvement Program to either Yes or No.

If you still encounter issues, go to the [Electric Cloud ElectricAccelerator Knowledge Base](#) and search for "Visual Studio". Also refer to ask.electric-cloud.com for answers to common issues with eMake and the Visual Studio Integration.

Common Issues

Check this list of common issues after you verify that Visual Studio initialized properly:

- [Visual Studio is missing the Electric Cloud menu](#)
- [Application Data folder could not be created. make: *** \[all\]](#)
- [For VS2005 SP1 builds, the build is not broken up and runs as one large job](#)
- [Build terminated with "not making progress" error](#)
- [Visual Studio quits immediately at the start of the build](#)
- [Error "'devenv' not found" is displayed](#)
- [Error "Unable to build specified project" or missing file errors](#)
- [Error "msbuild not found"](#)
- [Missing DLL errors or Visual Studio installation is corrupt](#)
- [Error "command line too long"](#)
- [The build is slow \(not parallelized\) and/or each line of the build output is prefixed with 1>, 2>, etc](#)

- Error: '[' not recognized
- When virtualizing the Visual Studio toolchain, regsvr32 fails trying to register a DLL that uses debug CRT DLLs
- Particular projects do not build under eMake
- Electric Cloud menu in Visual Studio is grayed out (disabled)
- Invalid macro invocation '\$' build error
- Using Visual Studio 2010, a project fails at link when using the add-in but succeeds when using Visual Studio alone
- Upgrading only cluster agents to Accelerator v7.0 might cause an error

Visual Studio is missing the Electric Cloud menu

Description

The VS IDE Add-In is installed, but the Electric Cloud menu is missing and the Tools menu item is corrupted (shows "Electric Cloud").

The add-in might throw an exception similar to the following:

```
3:Error: Adding Build menu item: Could not load file or assembly 'stdole, Version=7.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a' or one of its dependencies. The system cannot find the file specified.
```

Cause

This occurs because the add-in requires stdole.dll to be installed and registered.

Solution

1. Close all instances of Visual Studio.
2. Uninstall the add-in from Control Panel.
3. Open a command prompt and register the DLL manually (Adjust the path to gacutil.exe accordingly.):

```
"%PROGRAM_FILES%\Microsoft SDKs\Windows\v7.0A\Bin\gacutil.exe" -i "%PROGRAM_FILES%\Common Files\Microsoft Shared\MSEnv\PublicAssemblies\stdole.dll"
```

Run Visual Studio and check if the Electric Cloud menu is present.

4. If Step 3 doesn't work, install the Office 2003 Update: Redistributable Primary Interop Assemblies from <http://www.microsoft.com/download/en/details.aspx?DisplayLang=en&id=20923>.
5. Run:

```
devenv.exe /ResetSettings
```

This resets the menus.

6. Re-install the add-in and run Visual Studio.

Application Data folder could not be created. make: *** [all]

Cause

The current user does not have an account on the agent that is running devenv.exe.

Solution

Virtualize the Visual Studio toolchain or set `--emake-exclude-env=USERPROFILE`.

See <http://ask.electric-cloud.com/questions/299/what-does-it-mean-when-visual-studio-reports-the-application-data-folder-for-visual-studio-could-not-be-created>.

For VS2005 SP1 builds, the build is not broken up and runs as one large job

Cause

The hotfix for VS2005 SP1 is not installed.

Solution

See <http://ask.electric-cloud.com/questions/439/visual-studio-2005-behaves-as-if-the-visual-studio-add-in-is-not-installed>.

Build terminated with “not making progress” error

Cause

There are many reasons for this error. It usually occurs when a build has shown a modal dialog (that is not visible to the build user) and is waiting for input.

Solution

See <http://ask.electric-cloud.com/questions/427/why-does-visual-studio-stall-and-display-a-modal-dialog>.

Visual Studio quits immediately at the start of the build

Cause

You are running the wrong version of Visual Studio for your build.

Solution

Ensure the environment is setup for the version of Visual Studio you are using.

Error “devenv’ not found” is displayed

Cause

Visual Studio is not installed on the agent or is not in the same location as the build machine.

Solution

Install Visual Studio on the agent or set the `PATH` environment variable to reflect the installation directory on the agent.

Error “Unable to build specified project” or missing file errors

Cause

The missing project or files are not in the eMake root.

Solution

Make sure all files are either present on the agent, or in the emake root.

Error “msbuild not found”

Cause

.NET is not installed on the agent.

Solution

Install the relevant version of .NET on all agents.

Missing DLL errors or Visual Studio installation is corrupt

Cause

C++ redistributable is not installed on the agent.

Solution

Install the relevant redistributable for the version of Visual Studio you’re using on the agents.

Error “command line too long”

Cause

The add-in has generated a command line that is too long.

Solution

If the error occurs during linking, set `ECADDIN_EXPAND_LINKER_OBJECTS=true`, otherwise set `ECADDIN_USE_RELATIVE_PATHS=true` in your environment.

The build is slow (not parallelized) and/or each line of the build output is prefixed with 1>, 2>, etc

Cause

The build is not using the add-in. The 1>, 2> is an indication that devenv is being used.

Solution

Check that the VS Converter Add-In is installed on the agents or build machine. (Go to **Tools > Add In Manager**.)

For VS2005 SP1, check if the hotfix is installed ([see above](#)).

Check for other third-party add-ins on the agents. The VS Converter Add-In might not be compatible with other build-related add-ins.

Error: '[' not recognized

Cause

You are using an older Accelerator version (pre 7.0) that doesn't recognize order-only prerequisites.

Solution

Turn off `ECADDIN_USE_ORDER_ONLY_PREREQS`.

When virtualizing the Visual Studio toolchain, regsvr32 fails trying to register a DLL that uses debug CRT DLLs

Cause

These SxS DLLs cannot be virtualized and are not part of the Visual Studio redistribution.

Solution

Do one of the following:

- See [http://msdn.microsoft.com/en-us/library/aa985618\(v=VS.90\).aspx](http://msdn.microsoft.com/en-us/library/aa985618(v=VS.90).aspx)
- Copy debug DLLs from `<VSINSTALLDIR>\VC\redist\Debug_NonRedist` to the target directory (the location of the DLL that is being registered)
- Copy `Microsoft.VC90.DebugCRT.manifest` and `msvcr90d.dll` from `<VSINSTALLDIR>\VC\redist\Debug_NonRedist\x86\Microsoft.VC90.DebugCRT`

Particular projects do not build under eMake

Solution

Use `ECADDIN_DONT_PARSE_PROJECT` to specify the offending projects. Use either the project name or the project path as shown in the solution file.

Electric Cloud menu in Visual Studio is grayed out (disabled)

Cause

This might occur if you install Visual Studio **after** installing the add-in. Visual Studio's setup routine has not initialized the add-in.

Also, the debug log will contain: `AddCommandControls failed for Build: The parameter is incorrect. (Exception from HRESULT: 0x80070057 (E_INVALIDARG))`

Solution

Open a Visual Studio 2010 or later command prompt as administrator and type:

```
devenv /setup
```

Invalid macro invocation '\$' build error

Cause

An NMAKE limitation treats the dollar sign (\$) as a special character that precedes a macro name. It is not possible to use '\$' in a preprocessor definition unless the number of '\$' is even.

Solution

Either avoid having to use the single dollar sign (\$), or specify it by using a double dollar sign (\$\$).

Using Visual Studio 2010, a project fails at link when using the add-in but succeeds when using Visual Studio alone

Description

You encounter this error: LINK : fatal error LNK1123: failure during conversion to COFF: file invalid or corrupt

Solution

Upgrade Visual Studio to 2010 SP1.

Upgrading *only cluster agents* to Accelerator v7.0 might cause an error

Cause

When upgrading the cluster agents only to Accelerator v7.0, be advised that an older eMake client will run the same version of eMake on the agent (if it is available). This might result in the following error:

NMAKE : fatal error U1073: don't know how to make '|'

Solution

Do one of the following:

- Upgrade the local eMake client to 7.0 or later (recommended).
- Set `ECADDIN_USE_ORDER_ONLY_PREREQS=false` in your environment.

Index

B

build

local builds 6-3

multiple solution and projects 7-2

C

common issues 13-1

Converter Add-In 8-1

known issues 3-1

upgrading 4-4

D

debugging 12-1

E

ecdevenv 7-1

ElectricInsight 6-1, 6-3

eMake 6-1

environment variables 9-1

I

IDE Add-In installation 5-1

initializing Visual Studio 13-1

Insight (ElectricInsight) 6-1, 6-3

installation

options 5-2

silent 5-2

installing the IDE Add-In 5-1

M

main menu 6-2

makefile creation 8-1

msbuild 11-1

P

performance optimization of Visual Studio 10-1

prerequisites 4-2

problems, common 13-1

S

settings

advanced 6-17

command line 6-19

debug 6-14

options 6-13

supported Visual Studio versions 4-1

T

toolbar 6-2

toolchain virtualization 7-2

troubleshooting 13-1

U

upgrading 4-4

V

virtualization of the Visual Studio toolchain 7-2

Visual Studio

performance optimization 10-1

supported versions 4-1

Virtualization of the toolchain 7-2

