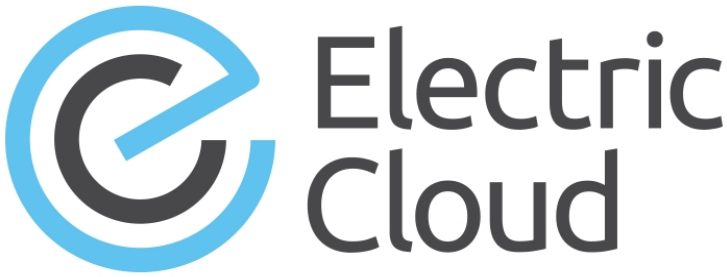


ElectricAccelerator

Visual Studio Integration Guide

Version 5.0

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

ElectricAccelerator Visual Studio Integration 5.0

Published 6/28/2016

Trademarks

Contents

Chapter 1: Overview

About the ElectricAccelerator® Visual Studio Integration

The ElectricAccelerator® Visual Studio Integration is composed of two distinct converters and the ecdevenv.exe utility:

- Visual Studio IDE Extension

  This extension 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 extension provides an Electric Cloud build menu and toolbar. The existing build menu remains intact for local (non-eMake) builds.

  This is implemented as a VSPackage in Visual Studio 2015 and an add-in in earlier versions of Visual Studio.

- Visual Studio Converter Extension

  This extension converts solutions and projects into NMAKE makefiles for eMake compatibility. It is invoked for command-line builds and ecdevenv.exe.

  This is implemented as a VSPackage in Visual Studio 2015 and an add-in in earlier versions of Visual Studio.

- 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. ecdevenv.exe is invoked by the Visual Studio IDE extension to perform eMake builds.

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 5.0

- Support Visual Studio 2015 (VSP-817, VSP-914, VSP-922, VSP-926, and VSP-1002)
- ElectricAccelerator no longer supports Visual Studio .NET, Visual Studio .Net 2003 and Visual Studio 2005. (VSP-983)
- Support for Chinese characters in path names. (VSP-991 and VSP-1018)
- Install Visual Studio 2010 32-bit redistributables for the new VSPackage extension. (VSP-1006)
- Support relative pathnames to enable relocation of generated makefiles. (VSP-1024)

Resolved Issues in Version 5.0

Visual Studio Converter Extensions

- Fixed the “Unable to open xxx_000.pch for read” error when switching between Visual Studio and ElectricAccelerator builds. (VSP-1008)
- ElectricAccelerator builds now have the same project build order as Visual Studio 2012 and later. (VSP-563)

Visual Studio IDE Extension

- Fixed the intermittent Visual Studio crash when building with ElectricAccelerator. (VSP-1020)

Chapter 3: Known Issues

- If up-to-date check is enabled, custom build steps will not rebuild if the output is missing.
- Order-only prerequisites are automatically turned off when up-to-date check is enabled.
- If the cluster upgrade option of the VS IDE Converter installer fails, re-run the installer to ensure the VS Converter 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 Integration. Adding a new language to an existing Visual Studio installation with the VS IDE Integration already installed causes Visual Studio to display an empty Electric Cloud menu. The workaround is to reinstall the integration.
- Visual Studio 2008 builds might break or might not be optimized after upgrading from an earlier version. The workaround is to go to the knowledge base article KBEA-00065, *The build breaks after upgrading to Visual Studio 2008*.
- In Microsoft Visual C++ 2010 and later, projects that contain "custom build rules" will not be parallelized at the project item level.
- Lightswitch projects are not supported.
- UWP projects are not supported.
- For Visual Studio 2010 and later, the MSBuild item metadata syntax is not supported for C++ build events (pre-build, pre-link, and post-build events). The workaround is to substitute the variables with actual values or use `ECADDIN_DONT_PARSE_PROJECT` to build the project with MSBuild.

Chapter 4: System Requirements

Supported Visual Studio Versions

The Visual Studio IDE Integration supports the following versions of Visual Studio:

- Visual Studio 2015
- Visual Studio 2013
- Visual Studio 2012
- Visual Studio 2010
- Visual Studio 2008

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

Prerequisites

MSBuild Prerequisites

The IDE converter 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	Packages
2015	• .NET Framework 4.6 • Microsoft Build Tools 2015
2013	• .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

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

VS IDE Extension Prerequisites

The VS IDE Extension requires:

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

VS Converter Extension Prerequisites

The VS Converter Extension requires:

- Microsoft Visual C++ 2005 SP1 Redistributable Package (Visual Studio 2013 and earlier)
- Microsoft Visual C++ 2010 Redistributable Package (Visual Studio 2015)
- .NET Framework v2.0

ecdevenv Utility Prerequisites

The ecdevenv utility requires:

- VS Converter Extension
- Microsoft Visual C++ 2005 SP1 Redistributable Package
- .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 Integration.

Chapter 5: Installing the IDE Integration

Installing the Integration Using the GUI

To install the VS IDE Integration locally, run the installer provided.

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

     Note: For the VS Integration VSPackage 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.

   Click **Next**.
5. When the **Select Visual Studio Version** screen appears, click **Next** to install the converter in all of the Visual Studio versions installed on your system, or select specific Visual Studio versions where you want the converter to be installed and click **Next.** By default, all the Visual Studio versions are selected.
6. When the **Installing** screen appears, view the progress of the installation.
7. When the **ElectricAccelerator VS Integration Setup** screen displays “ElectricAccelerator VS Integration Wizard Complete", 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.

Installing the Integration 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
`/help`	Displays usage information.
`/clustermanagerhost`	Specifies the cluster manager to be used for the cluster upgrade.
`/mode [ARG]`	Sets the installation mode. Available values: `standard` or `silent`.
`/prefix [ARG]`	Sets the installation directory.
`/response-file [ARG]`	The file from which to read installer responses.
`/save-response-file [ARG]`	The file to which installer responses are written when the installer exits.
`/temp [ARG]`	Sets the temporary directory used by the program.
`/type [ARG]`	Performs the selected type of installation. Available values are `uiaddin` and `cluster`.
`/version`	Displays installer version information.
`/vsversion`	Installs the converter for the specified list of Visual Studio versions. Available values are `2008`, `2010`, `2012`, `2013`, and `2015`.

Chapter 6: Using the VS IDE Extension Interface

When the VS IDE Extension starts, it checks if the following are present:

- eMake— If the converter cannot find eMake, the converter'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.

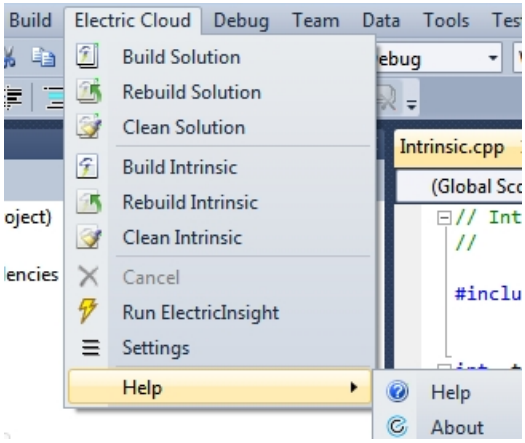
  Do not run local (non-eMake) builds while running eMake builds and vice-versa.

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

Topics:

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.

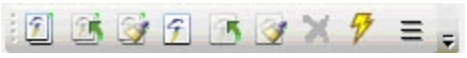
  On 64-bit systems, this menu option invokes the 64-bit version of ElectricInsight. For details about the 64-bit and 32-bit versions of ElectricInsight, see Chapter 1, *Installing ElectricInsight* in the *ElectricInsight User Guide* at http://docs.electric-cloud.com/accelerator_doc/AcceleratorIndex.html.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

- Help sub-menu
    - Help – Display help resources.
    - About—Displays information about the current extension installed.

When selecting one of the build commands, the converter 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.

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

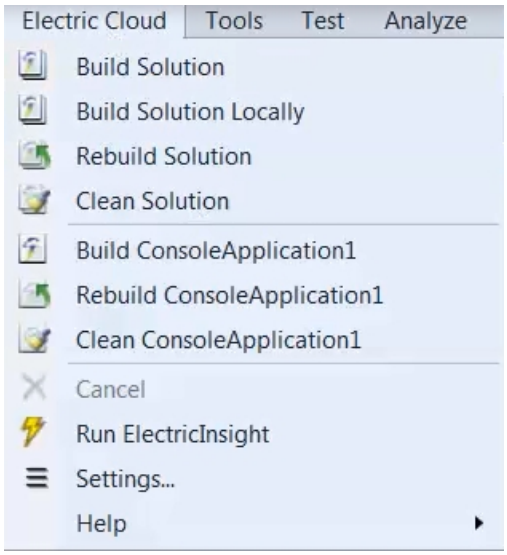


Build Solution Locally

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.

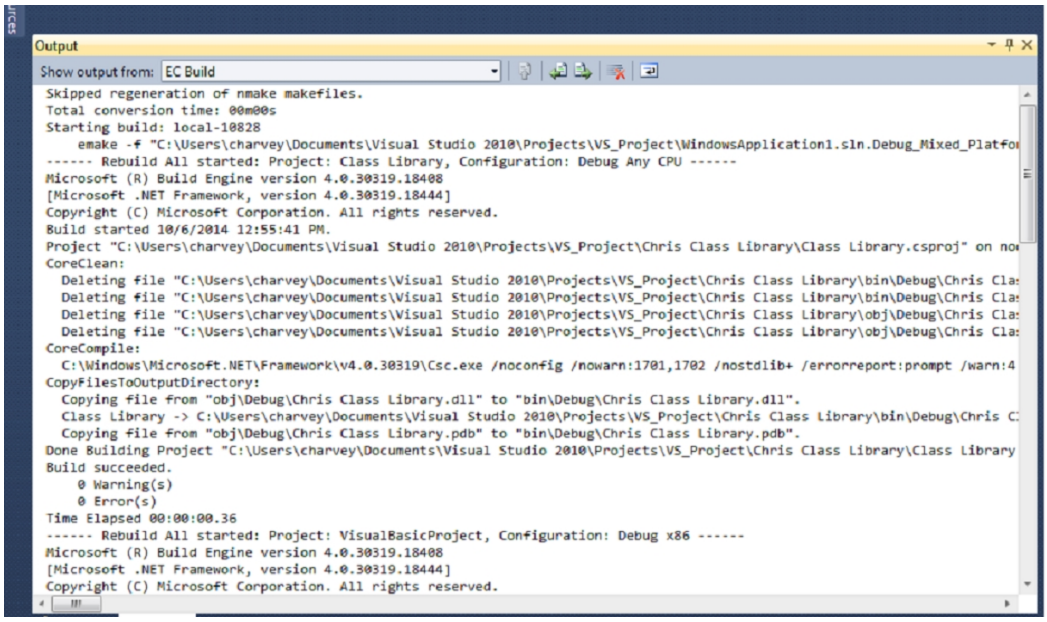


Advisories for Build Solution Locally

- 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

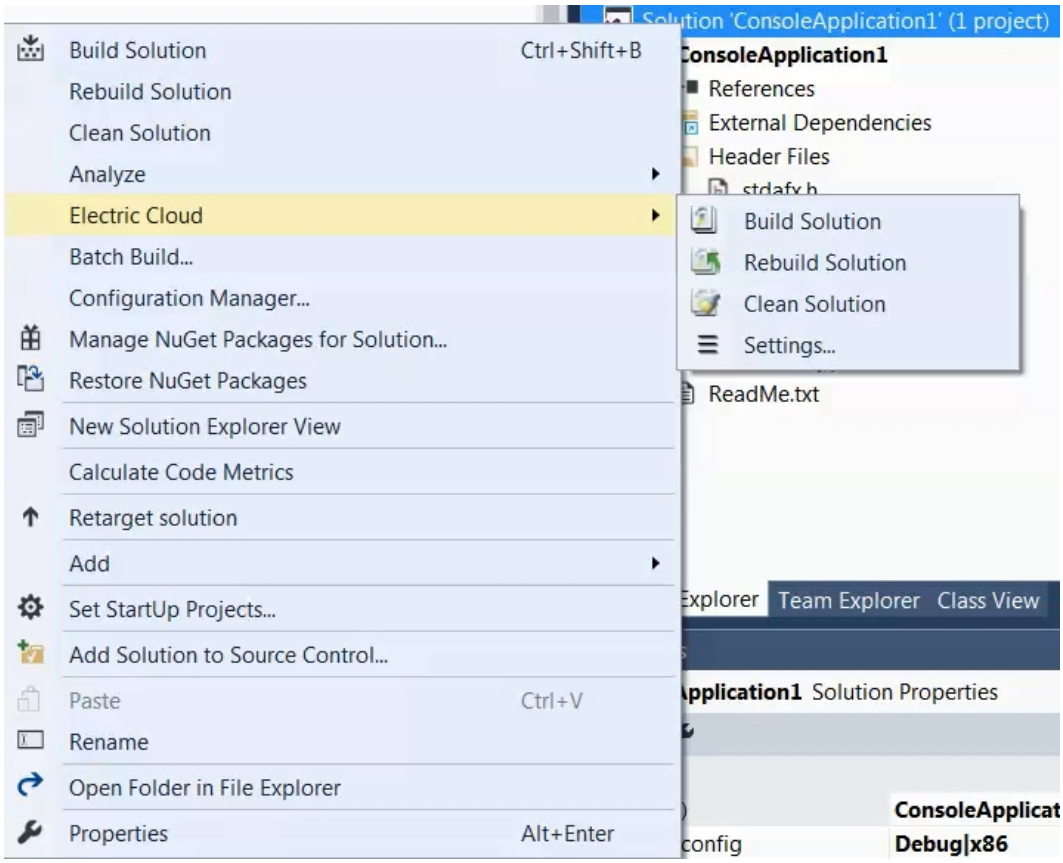
Output from an eMake build is displayed in the EC Build output pane (displayed in the following screenshot).



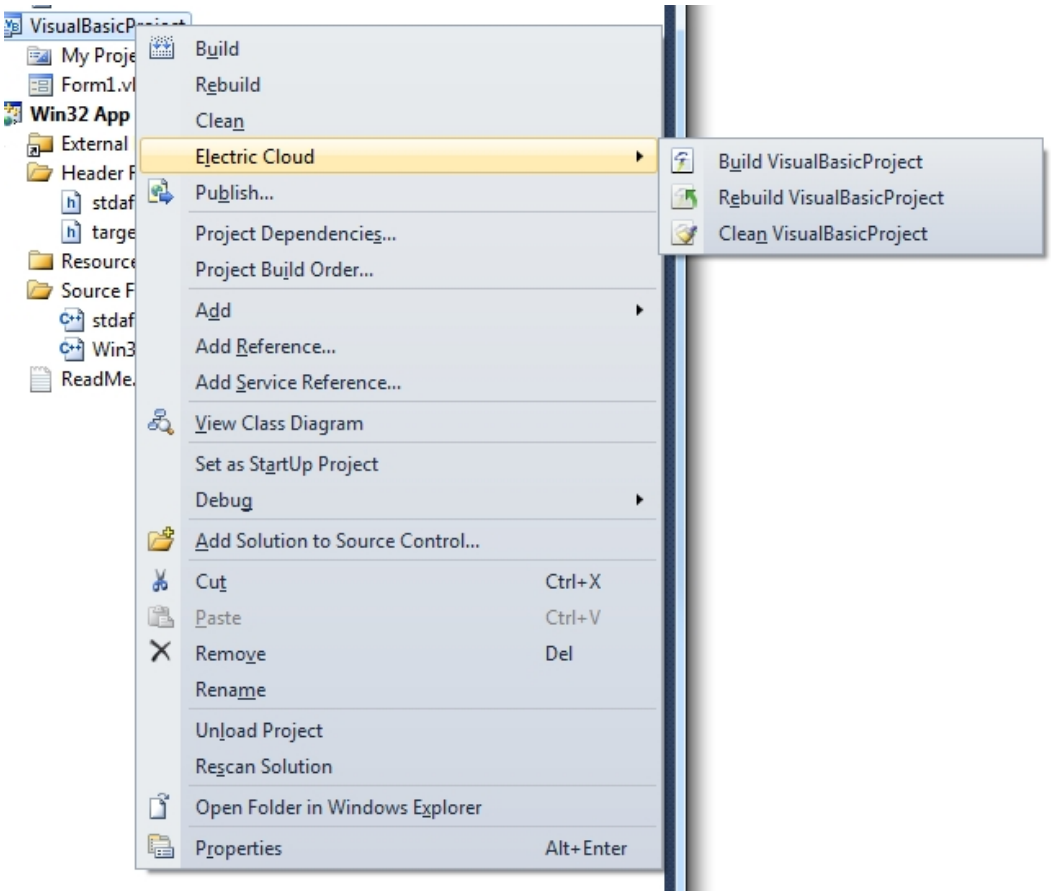
Context-Sensitive Menus

The converter provides additional context-sensitive menus.

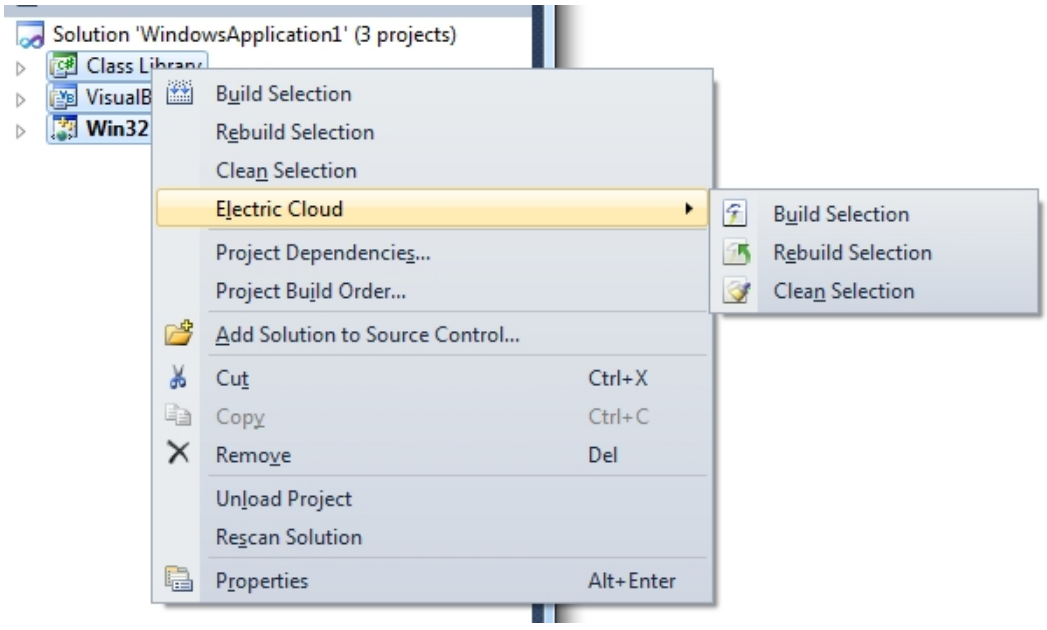
The following screenshot illustrates the Solution menu.



The following screenshot illustrates the Project menu.



The following screenshot illustrates the Selection menu.



Solution Settings

The Solution settings dialog box contains the following Solution Setting categories:

- General Solution Settings
- Options Solution Settings
- Debug Solution Settings
- Advanced Solution Settings
- Command Line Solution Settings

Note: See Setting VS Converter Environment Variables for environment variable descriptions.

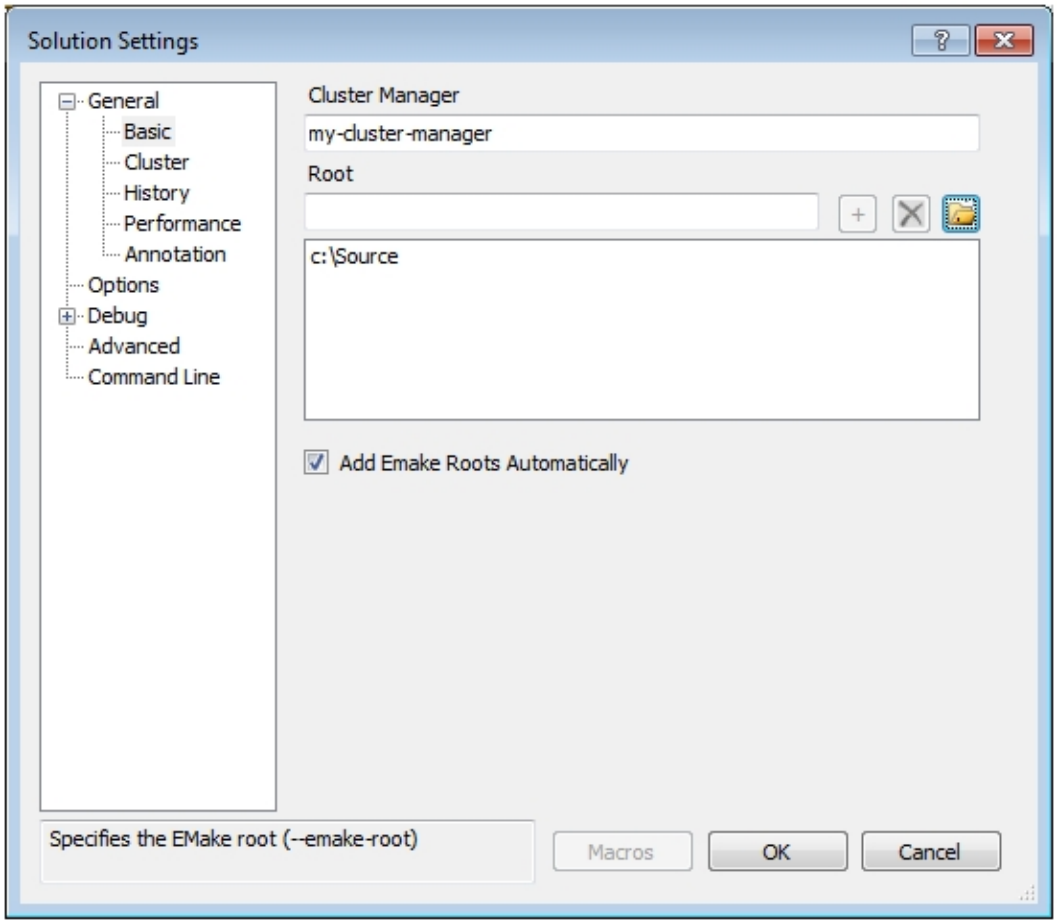
General Solution Settings

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.



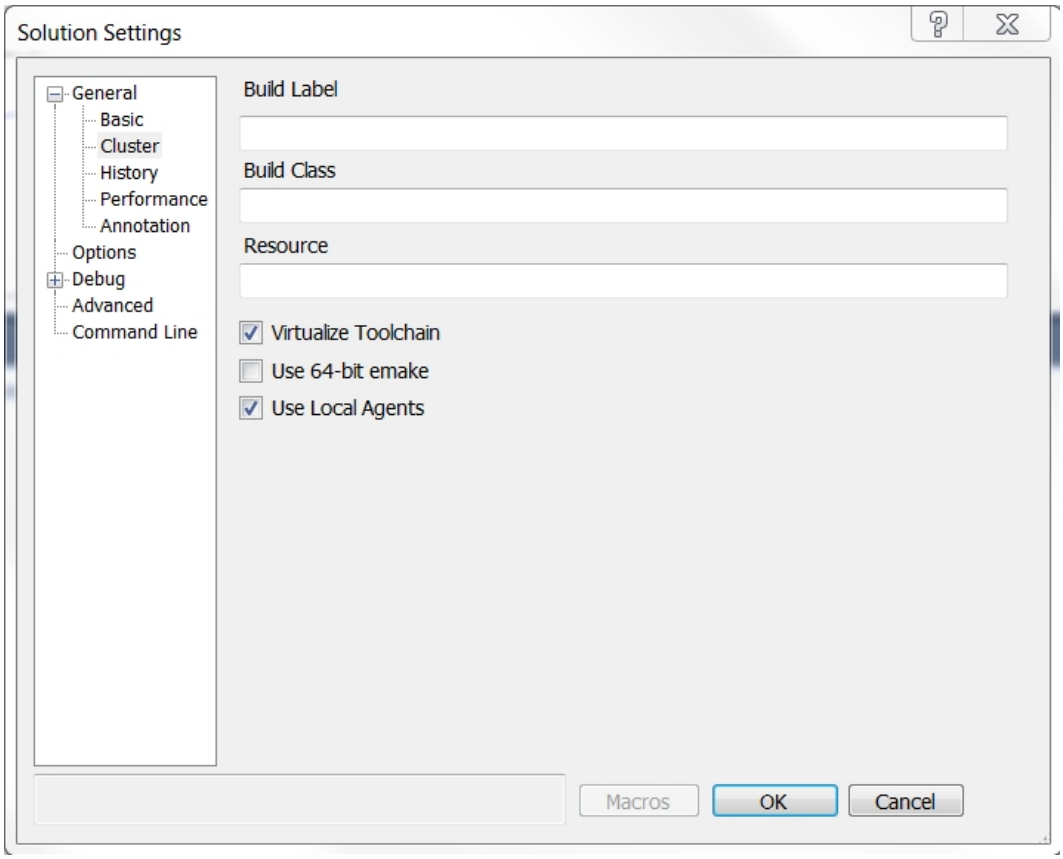
- Cluster Manager—Indicates the eMake Cluster Manager (`--emake-cm`). If this field is empty, an eMake build is performed with local agents when **Use Local Agents** is selected. When **Use Local Agents** is not selected, a local eMake build (without remote or local agents) is performed.
- Root—Specifies 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

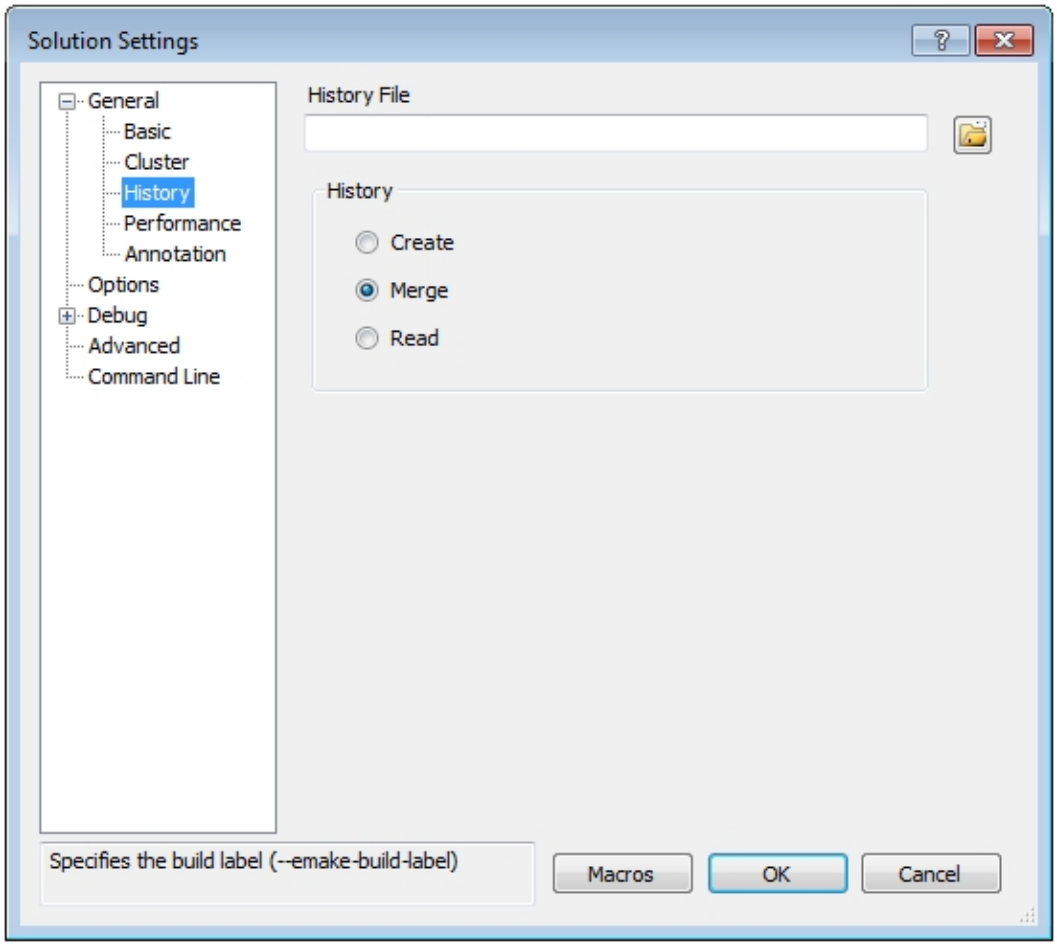
The following screenshot illustrates the Cluster sub-category.



- 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

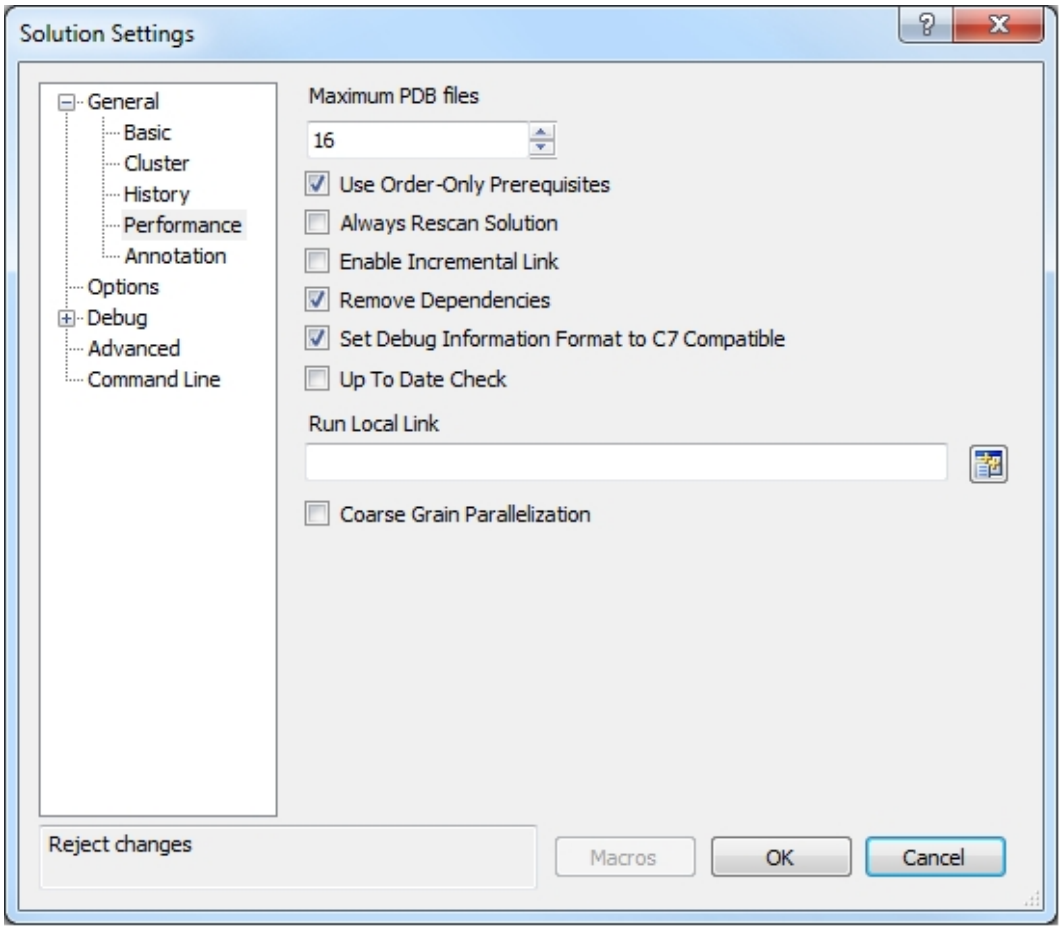
The following screenshot illustrates the History sub-category.



- 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.

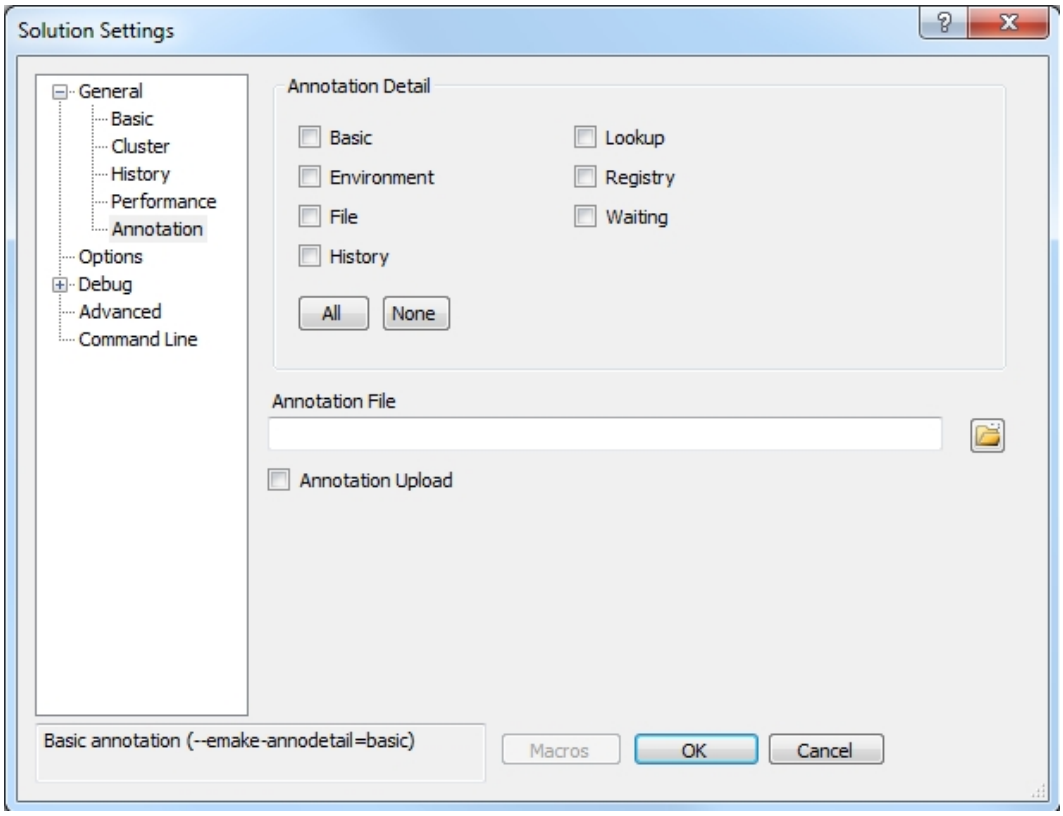


- Maximum PDB Files—Sets the maximum number of PDB files used when splitting (sets `ECADDIN_MAX_PDB_FILES`). Default is 16. If you have fewer than 16 agents, you can decrease this value to be equal to or less than the number of agents.
- Use Order-Only Prerequisites—Determines whether to use order-only 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.

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`).

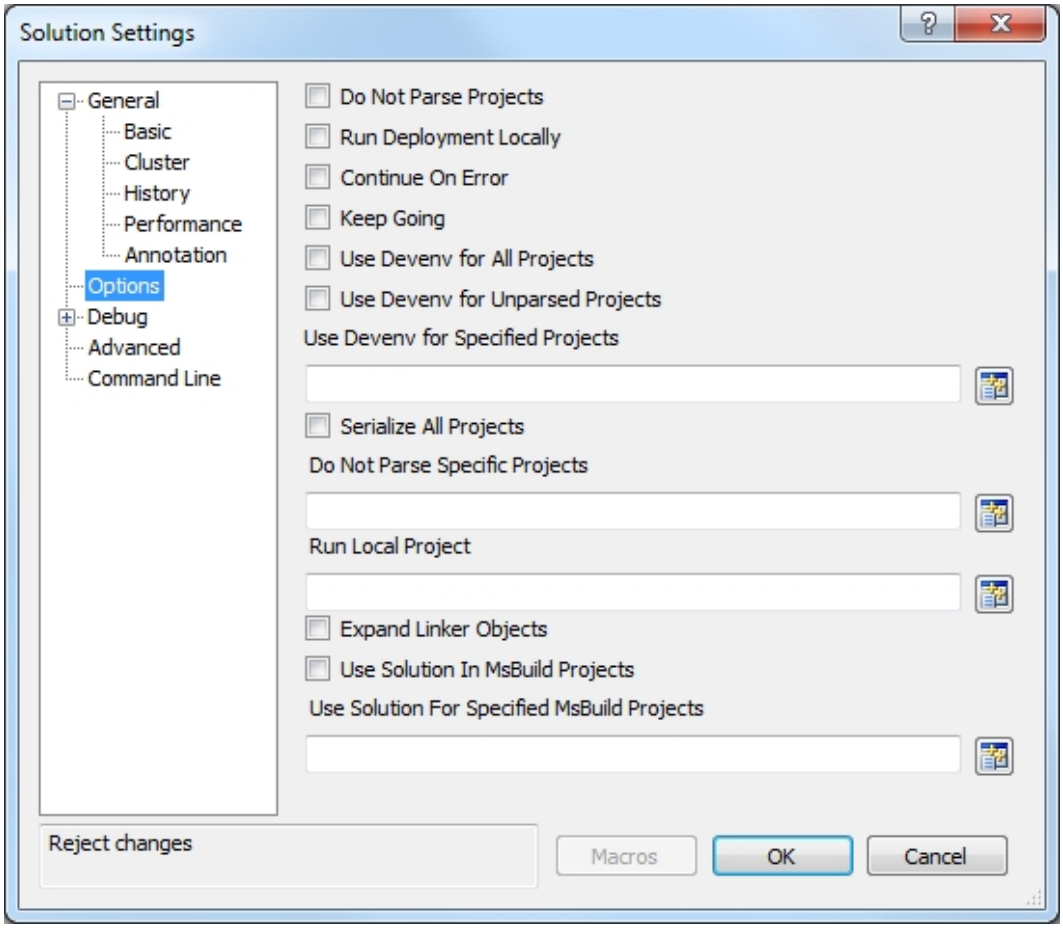
  Note: If your local eMake version is 8.1 or newer, and if you select no Annotation Detail checkboxes, the converter sets the `--emake-annodetail` eMake option to none. In this case, if the version of eMake is older than 8.1 on the *agent* machines, the following error appears at build time:

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

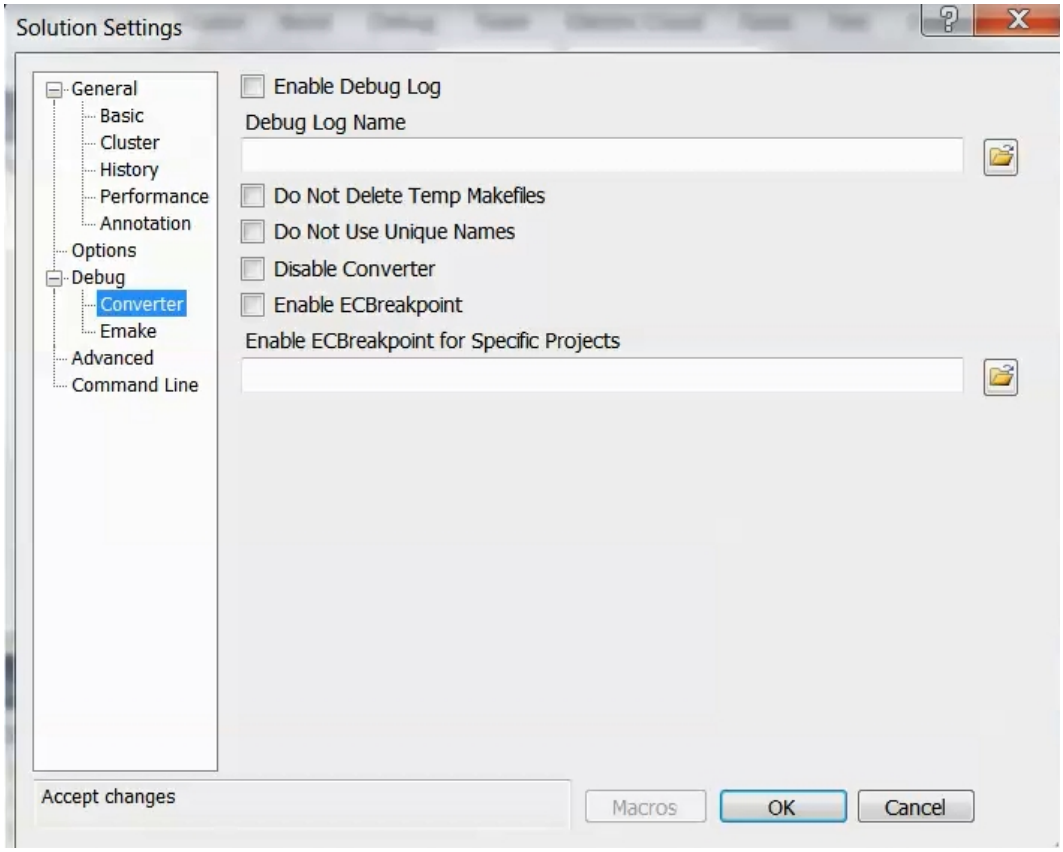
  This variable is useful for deploying the converter. If, for any reason, the converter cannot build some of your projects, this variable lets you work around the problem.
- Run Local Project—Runs specified projects locally using `#pragma runlocal` (sets `ECADDIN_RUN_LOCAL_PROJECT`). You can either type 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 converter. 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 Debug Solution Settings and eMake Solution Settings.

Converter Solution Settings

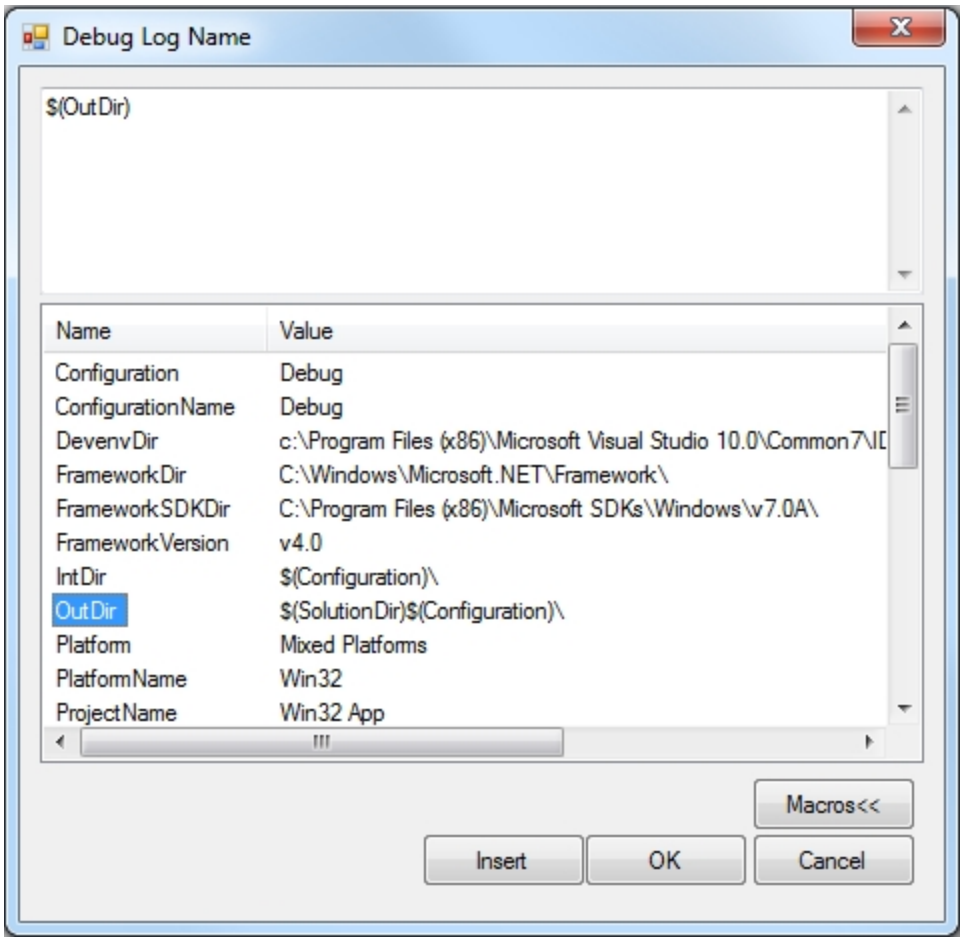


- Enable Debug Log—Determines whether to enable debug logging (sets `ECADDIN_DEBUG=true` if enabled).
- Debug Log Name—Specifies the name of the debug log (sets `ECADDIN_DEBUG_LOG_FILENAME`). The default location is `%TMP%\ecdebug<unique>.log` (where `%TMP%` is usually `C:\Users\<username>\AppData\Local\Temp`). 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

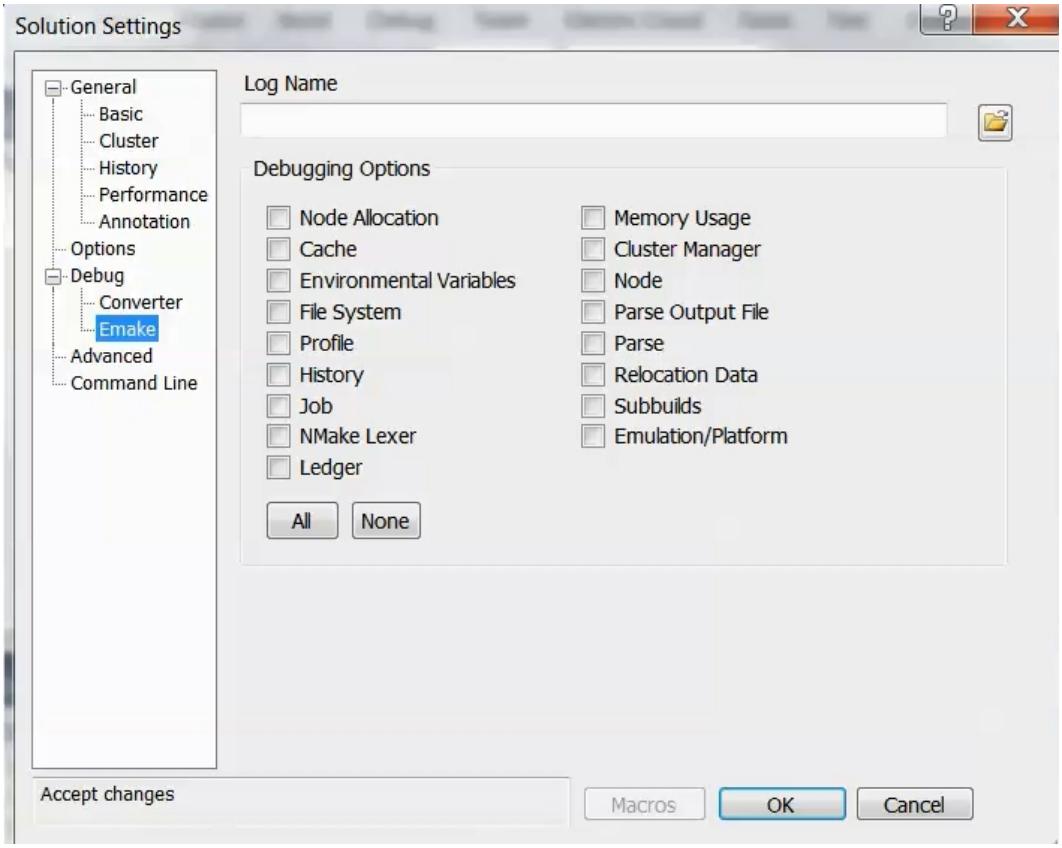
In any string-based field (such as the path name you enter into the **Debug Log Name** field) in the Electric Cloud solution settings, you can use a custom build command instead of a hardcoded string. For example, you can create a command that generates separate debug logs if you are running separate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

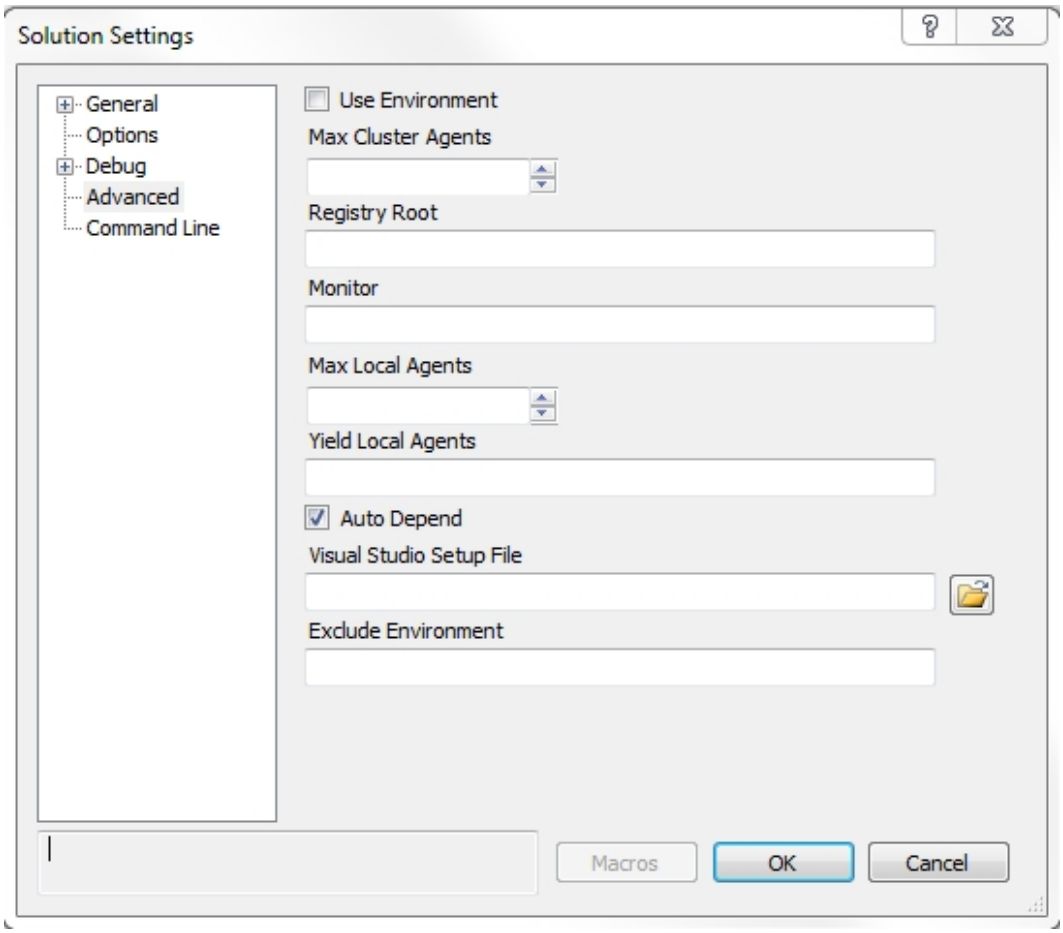
eMake Solution Settings



- 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.

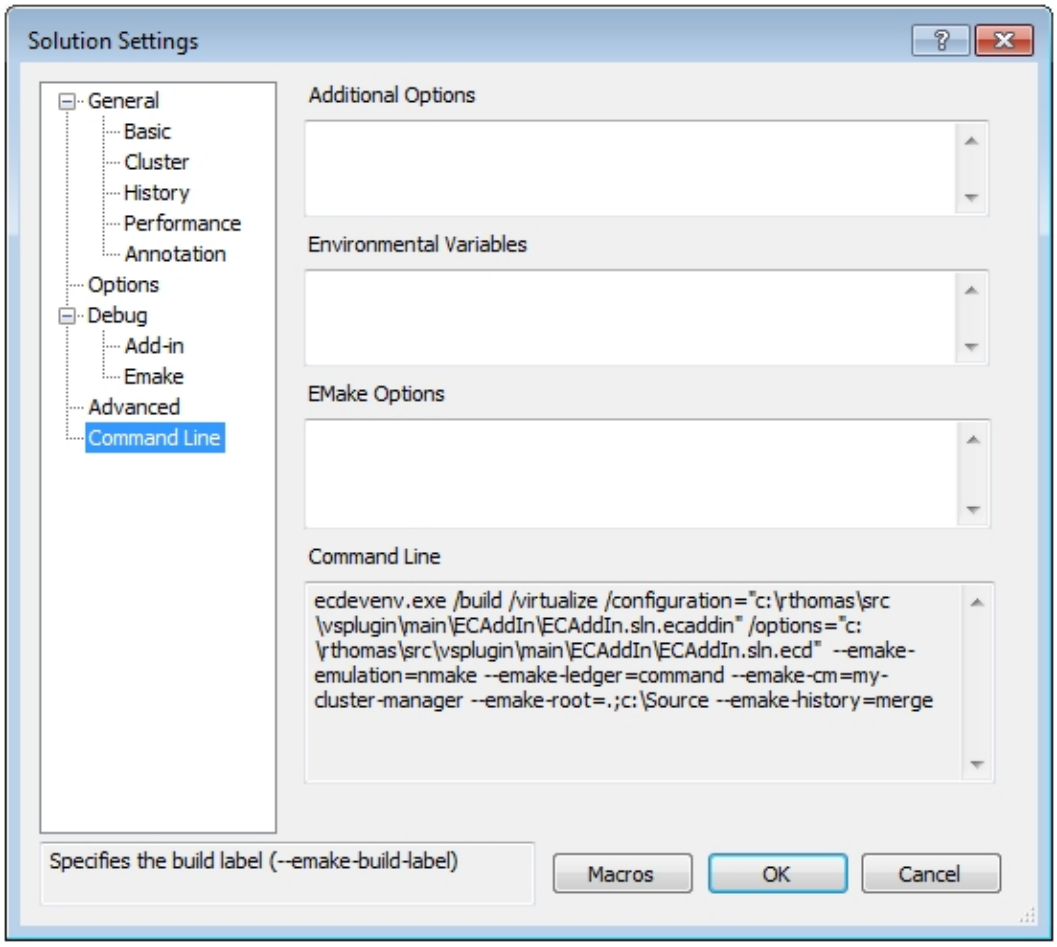
  Note that Auto Depend is now enabled by default in ecdevenv when ecdevenv is invoked from the Visual Studio command prompt. (When Auto Depend is disabled in the IDE Add-In GUI, this default is overridden when ecdevenv is called from the GUI.)

  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 (`--emake-exclude-env`), separated by ':' [a colon].

Command Line Solution Settings

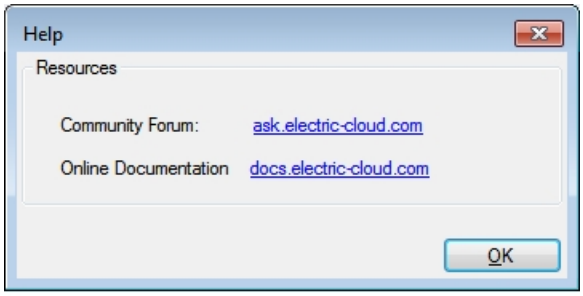
This category lets you add additional options not explicitly specified elsewhere.



- Additional Options—Specifies a list of converter options in this format: `<Name=Value>`. These are the same options specified on Setting VS Converter 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 converter. VS Converter options are stored in the `/options` file.

Help Menu

This dialog box shows the help resources:



The Community Forum (https://ask.electric-cloud.com/) is where you can find solutions to common problems, or ask a question. Go to http://docs.electric-cloud.com/ to find documentation for all Electric Cloud products.

About

This dialog box shows version information about the currently installed IDE add-in or package, the eMake version and the Electric Insight version. Refer to these versions where contacting support or asking questions on the community forum.


ElectricAccelerator VS IDE Add-in v5.0.0.0
Electric Cloud
ElectricAccelerator VS IDE Add-in v5.0.0.0
Electric Make version 8.2.1.66459 32-bit
Electric Insight v5.1.0.5899
Copyright (c) 2016 Electric Cloud Inc.
Provides user-space UI integration for calling ElectricAccelerator from within the Visual Studio IDE v5.0.0.0
OK

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 converter 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.

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

Option	Description
`/skip`	Never regenerates makefiles. You use this when you have cached makefiles. You must manually regenerate makefiles using `/force` when required.
`/use64bit`	Specifies the 64-bit version of eMake.
`/version`	Displays the ecdevenv version number.
`/virtualize`	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 `#pragma runlocal`. All project lists are delimited with semi-colons. See the converter option `ECADDIN_RUN_LOCAL_LINK`.
`ECDEVENV_SKIP_CHECKS`	Skips checking for `devenv.com` in the PATH 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 converter 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.

Supporting multi-byte characters

When building solutions that have multi-byte characters in their pathnames, set the following environmental variables:

- `ECADDIN_USE_INLINE_FILES=false`
- `EMAKE_MAKEFILE_CODEPAGE=65001`

Chapter 8: Building VS from the Command Line

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.

IMPORTANT: For Visual Studio 2015 and later, you must add "`/vsp`" to the command line. (When building from the IDE or from ecdevenv.com, this is added automatically.)

For example:

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

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 Environment Variables

Setting the Converter Variables

You can control the way the VS Converter 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. The EMAKE configuration file (in `$(InstallDir)\i686_win32\conf\emake.conf`) may also override these settings.

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	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 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. 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.	False (converter) True (ecdevenv and IDE)
`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_DISALLOW_PCH	DisallowPCH	Does not generate/use precompiled header files (implied by ECADDIN_MAX_PDB_FILES)	False
ECADDIN_DISALLOW_PDB	DisallowPDB	Does not generate PDB files.	False
ECADDIN_DISALLOW_SBR	DisallowSBR	Does not generate browse information files from sources.	False
ECADDIN_DONT_ADD_PCH_LOCATION	DoNotAddPCHLocation	Prevents the converter from adding the location of the PCH file in all cases. This variable is relevant only if ECADDIN_MAX_PDB_FILES or ECADDIN_DISALLOW_PCH is switched on.	False
ECADDIN_DONT_ALLOW_PCH_AND_PDB	DoNotAllowPCHAndPDB	Switches off PDB and PCH generation.	False
ECADDIN_DONT_PARSE_PROJECT	DoNotParseSpecificProjects	Prevents the VS Converter 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 converter. If, for any reason, the converter cannot build some of your projects, this variable lets you work around the problem. 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.	-

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 converter 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 converter. This environment variable can have any value, it just needs to be set. Also, you can disable the converter on each host by using the Visual Studio Converter 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 ectouch.exe. 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 `false` 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 converter. For details, see the Relocating Makefiles Generated by the Converter 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_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
ECADDIN_RUN_LOCAL_LINK	RunLocalLink	A list of projects where the linker will be run locally (using `#pragma runlocal`).	-

Environment Variable	Configuration File Variable	Description	Default
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 converter. 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, `solution1/project1.vcproj`). Do not add quotation marks or white spaces.	-
ECADDIN_SERIALIZE	SerializeAllProjects	Causes each project to be built serially. It inserts `'#pragma allserial'` into each makefile. This variable is equivalent to setting ECADDIN_DONT_PARSE_PROJECTS.	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 `devenv` (rather than `msbuild`) for all unparsed projects.	False
ECADDIN_USE_DEVENV_FOR_PROJECT	UseDevenvForProject	Uses `devenv` (rather than `msbuild`) to build specific projects. Supply a list of projects (separated by a semicolon) to be built with `devenv`.	-
ECADDIN_USE_LEGACY_CODE	UseLegacyCode	Use this variable to workaround a Visual Studio bug where AdditionalLibraryDirectories does not give the correct value.	False

Environment Variable	Configuration File Variable	Description	Default
ECADDIN_USE_MSBUILD	UseMSBuild	Lets you use msbuild internally for projects that the converter 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 and to allow the build location to be changed without regenerating makefiles.	-
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 Converter

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

(Note that the makefiles are relative to the solution directory.) 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) when it starts. To disable the checks, set the environment variable `ECDEVENV_SKIP_CHECKS=true`.

Chapter 10: Tuning Performance

The converter 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 converter 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 find-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 converter does not rebuild prebuild events. Instead, the converter 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 Converter

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 converter “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 converter inserts a call to `ectouch.exe`, which performs the action stated above. `ectouch.exe` must be in `%PATH%`.

To disable this feature with the converter, set `ECADDIN_ENABLE_INCREMENTAL_LINK=false`.

Improving Incremental Linking Time without the Converter

If you are not using the converter, 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 Converter

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 converter.

`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 Converter solution settings. Go to the Performance section of the Converter 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 Converter

This technique can be used without using the converter. 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",Platfo
rm="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",P
latform="Mixed Platforms"
```

Chapter 12: Uninstalling Visual Studio Integration

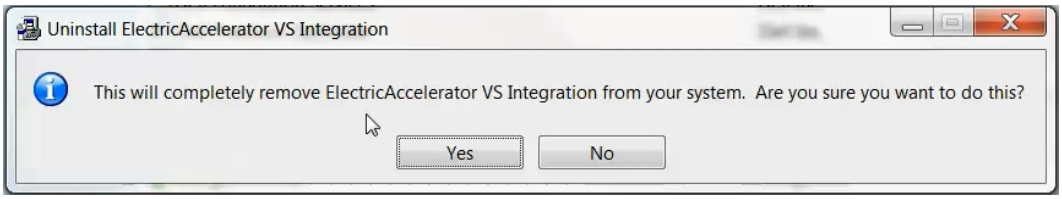
An uninstallation removes all VSP components from a machine. This includes the converters and the ecdevenv.exe utility from the prior installation. To uninstall the software, follow the procedures described in the following topics.

Uninstalling Visual Studio Integration on Windows

The uninstaller removes all Visual Studio Integration components from a machine at the same time.

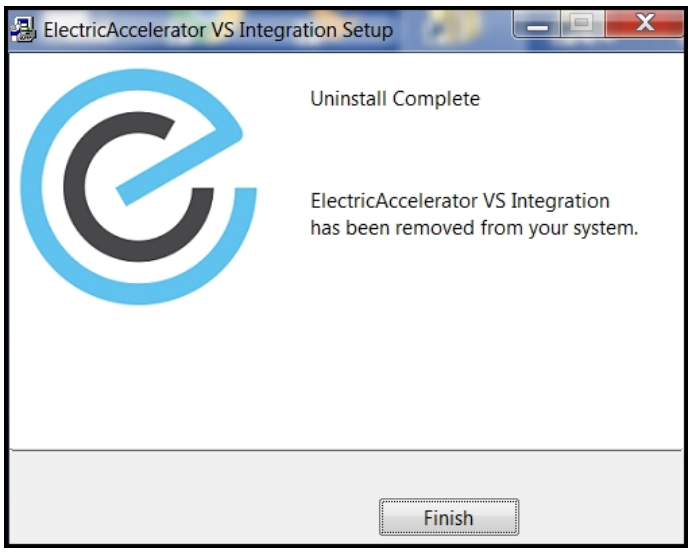
Using the GUI

1. Go to the Electric Cloud installation folder.
2. Run the uninstaller one of these ways:
    a. Enter `uninstall-vsaddin.exe`.
    b. Invoke the uninstaller by using the "Add or Remove Programs" or "Programs and Features" utilities in the Windows Control Panel. The following prompt appears:



3. Click **Yes**.

   The "Uninstall Complete" dialog box appears when the uninstallation is complete:



Using the Command Line

You can also run the uninstaller in silent mode from the command line.

1. Open a command prompt.
2. Enter `cd <Electric Cloud installation folder>` to go to the Electric Cloud installation folder, where `<Electric Cloud installation folder>` is the installation folder.
3. Enter `uninstall-vsaddin.exe /mode silent`.

Chapter 13: 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 ecdevenv, this will be the local machine. When running devenv or ecdevenv remotely, the file will exist on the remote machine.

Chapter 14: 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

Visual Studio is missing the Electric Cloud menu

Description

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

The converter 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 converter requires stdole.dll to be installed and registered.

Solution

1. Close all instances of Visual Studio.
2. Uninstall the converter 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 converter 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 box (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 converter 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 converter. The 1>, 2> is an indication that devenv is being used.

Solution

Check that the VS Converteris 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 converters on the agents. The VS Converter might not be compatible with other build-related converters.

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
- 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 converter. Visual Studio's setup routine has not initialized the converter.

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