ElectricAccelerator

Visual Studio Integration Guide

for version 4.1

Electric Cloud, Inc.
www.electric-cloud.com

Document Rev. 1

Published 12/5/2013

Contents

Chapter 1: Overview

Product Description

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 allows you to build Visual Studio solutions and projects from within the IDE using Electric Make® (eMake). The add-in provides an Electric Cloud build menu and toolbar. The existing build menu remains intact for local (non-eMake) builds. You must install this add-in separately as instructed in Visual Studio IDE Add-in Installation.

- Ecdevenv.exe Utility

  Ecdevenv is a drop-in replacement for devenv.exe that will build Visual Studio solutions and projects using eMake. Ecdevenv provides a number of important features. See Using the ecdevenv.exe Utility for information.

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

  This add-in is a command line add-in used by Electric Make to convert Visual Studio projects into NMAKE makefiles. This add-in is automatically installed with Accelerator.

Chapter 2: What's New

Visual Studio IDE Add-in

What's New for Version 4.1

- Added support for Visual Studio 2013.

What's New for Version 4.0

- Redesigned the settings dialogs and removed global options. (VSP-601)
- Conversions to NMAKE makefile can be done locally. (VSP-211, VSP-521)
- Virtualized the Visual Studio toolchain. (VSP-583)
- The Electric Cloud menu is always displayed. (VSP-615)
- Improved default settings to obtain the best performance. (VSP-616)
- This version of the VS IDE Add-in **requires Accelerator v7.0.2 or later**.

Visual Studio Converter Add-in

What's New for Version 4.1

- Added support for Visual Studio 2013.
- The same file is no longer built multiple times. (VSP-718)
- Fixed a bug where files with the .mm extension were unable to be compiled. (VSP-714)
- Corrected the handling of Visual Studio 2010 preprocessor macros. (VSP-713)
- Ecdevenv now supports environment variables in solution files. (VSP-684)

What's New for Version 4.0

- You can now use a configuration file to set add-in options. (VSP-369)
- Renamed the Solution Support Add-in to Visual Studio Converter Add-in. (VSP-557)
- Prevent linker from including libraries that do not have any exports. (VSP-626)
- Ecdevenv now builds solutions directly from the command-line.

VS Converter Add-in Upgrade Notes

When you upgrade from VS Converter Add-in v3.0, “debug” builds using PDB files will generate many conflicts on the first build. Subsequent builds will be fine. This occurs because the build order may have changed since v3.0.

Electric Cloud recommends regenerating history whenever you upgrade the add-in.

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.
- Due to an issue with previous versions' uninstallers, an upgrade may cause the following error message: "Cannot find script file: C:\ECloud\i686_win32\bin\unregaddin.vbs." You can safely 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 may 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 may break or may not be optimized after upgrading from an earlier version.

  Workaround: See knowledge base article KBEA-00065.
- 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 may 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. It is not possible to use '$' in a preprocessor definition unless the number of '$' is even.

  Workaround: Either avoid having to 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 (i.e., pre-build, pre-link, and post-build events).

  Workaround: Substitute the variables with actual values.
- If you are splitting PDBs (the default for /Zi or /ZI), you may experience slow initial build times after upgrading to v4.0. This occurs because filenames for PDB and IDB files have been changed to

distinguish between PCH and non-PCH files. After history has been regenerated, build times will return to normal.

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-ins for Visual Studio 2010 or later do not currently 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 2003
  - Visual Studio 2002

Prerequisites

For the VS IDE Add-in:

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

For the VS Converter Add-in:

- Microsoft Visual C++ 2005 SP1 Redistributable Package (does not matter which version of Visual Studio you use)

Chapter 5: Visual Studio IDE Add-in Installation

Installing the VS IDE Add-in

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

1. Run the `VSAddIn-<version>-Install.exe` file.
2. Welcome screen—click **Next**.
3. Choose Destination Location screen—accept the default install location or click **Browse** to change the location. Click **Next**.
4. Setup Type screen—select the 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 you specify.

     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.
   - Custom (This option allows you to select multiple setup types from this list.)

   Click **Next**.
5. Start Copying Files screen—review your settings before continuing the installation. Click **Next** to continue or **Back** to make changes.
6. When the wizard displays “Install finished,” your installation is complete. Click **Finish** to close the installer.

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

Silent Installation

To do 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.

Installation Options

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

The following options are available to customize your installation:

Option	Description
`/help`	Displays help information.
`/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: `addin`, `uiaddin` or `cluster`.
`/version`	Displays installer version information.

Chapter 6: VS IDE Add-in Interface

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

- Electric Make - 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.

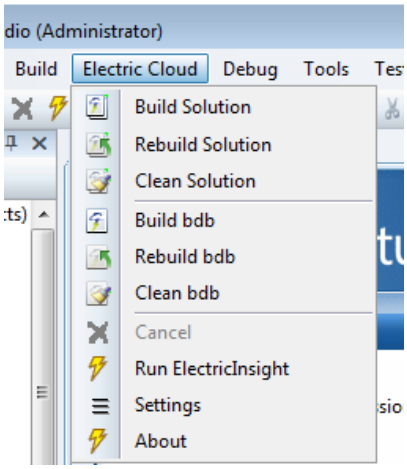
  Do not run local (non-eMake) builds while running eMake builds and vice-versa.

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

Topics:

Main Menu and Toolbar

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:

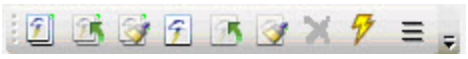
- Build Solution - Builds the current solution
- Build Solution Locally - Builds the current solution locally with eMake but ***without using*** remote agents or local agents (this function is equivalent to turning off the Cluster Manager and local agents). This function is hidden by default. See Build 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 may 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.
- 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.

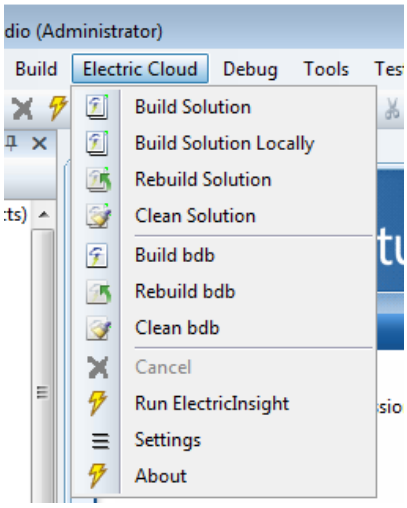


Build Solution Locally

You can choose to build a solution locally with eMake but ***without using*** remote agents or local agents. You may 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.

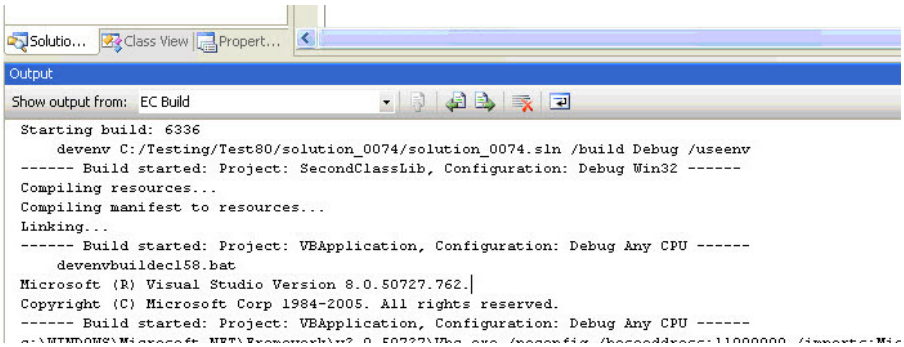


Advisories for Build Solution Locally

- The eMake local build does ***not*** support autodepend. This means changes in header files may not cause dependent source files to be recompiled.
- The eMake local build does ***not*** produce an annotation.
- Because history is not generated, unexpected conflicts may occur on subsequent eMake cluster builds.

Output Pane

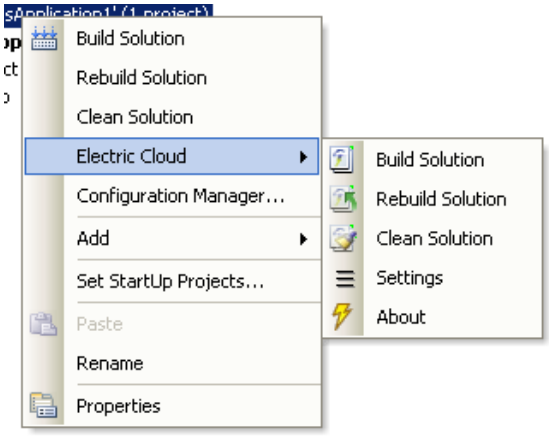
Output from an eMake build is displayed in the EC Build output pane (displayed in the following screenshot).



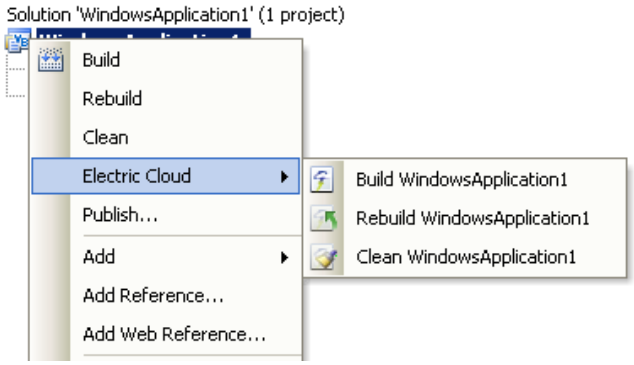
Context-Sensitive Menus

The add-in provides additional context-sensitive menus.

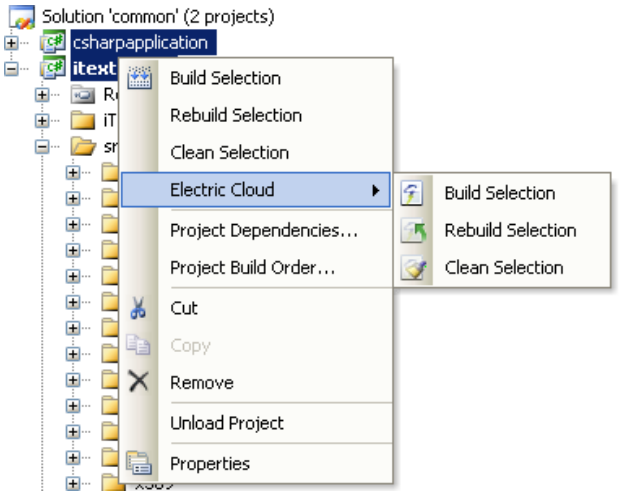
The following screenshot illustrates the Solution menu.



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
- Options
- Debug
- Advanced
- Command Line

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 Visual Studio Environment Variables for environment variable descriptions.

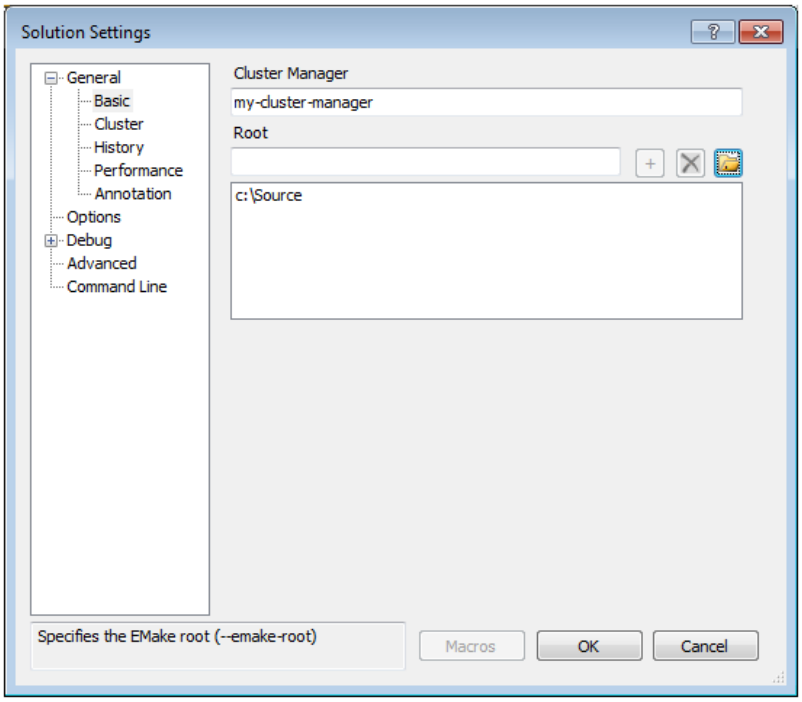
General

This category contains most frequently used settings:

- Basic
- Cluster
- History
- Performance
- Annotation

Basic

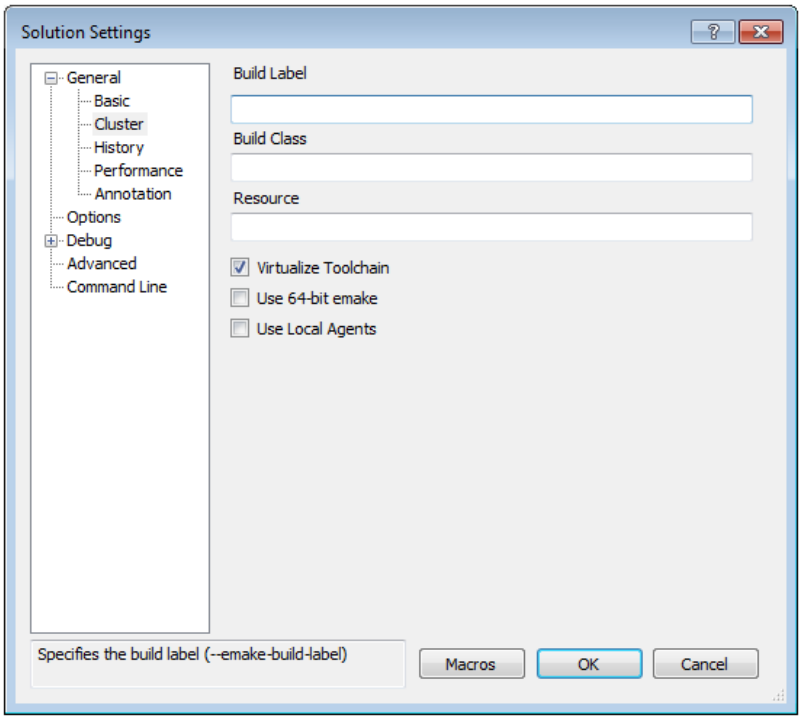
The following screenshot illustrates the Basic sub-category.



- 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, 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, select it in the list and click the x button.

Cluster

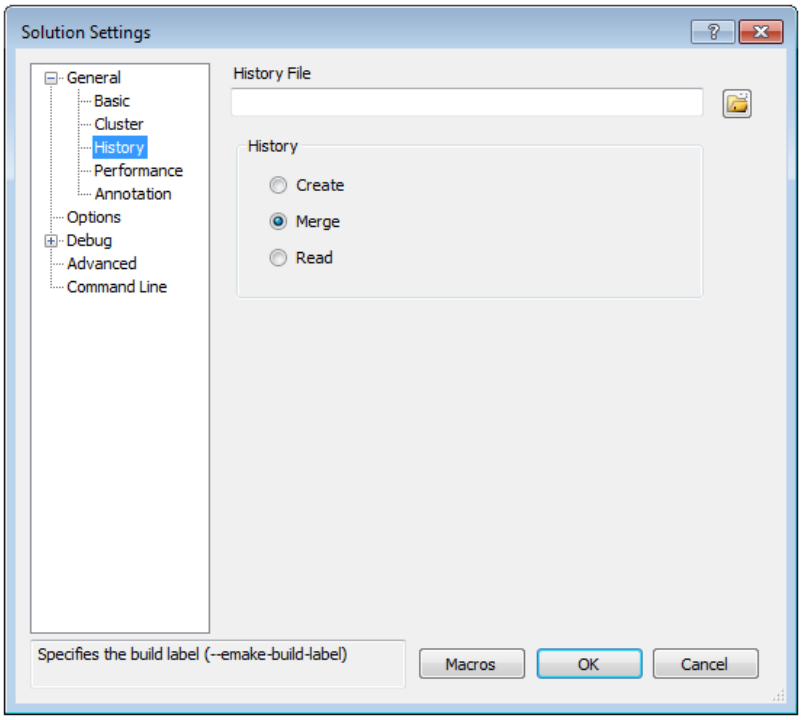
The following screenshot illustrates the Cluster sub-category.



- 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.
- Use 64-bit eMake - Determines whether to use the 64-bit version of eMake.
- Use Local Agents - Determines whether to use local agents (`--emake-localagents`).

History

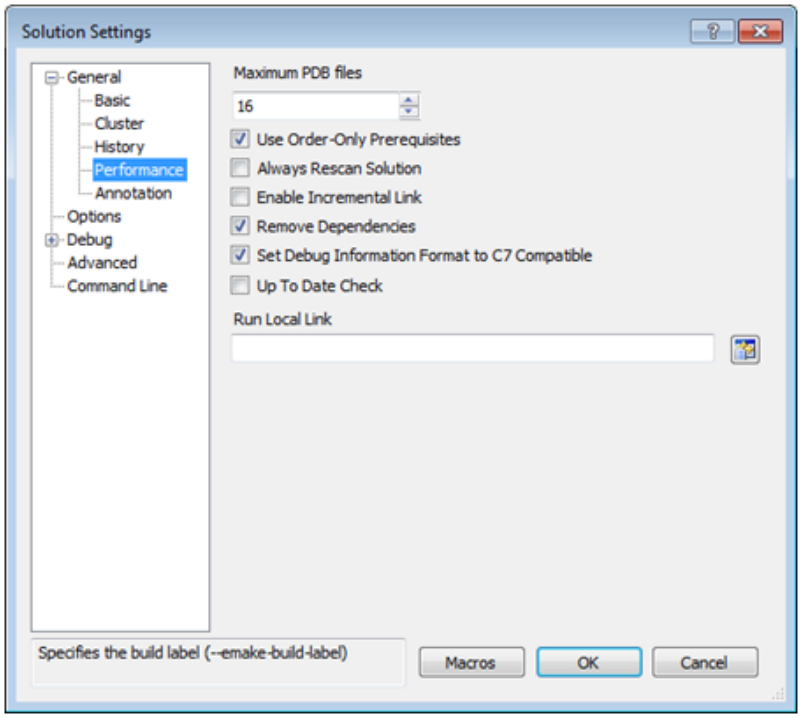
The following screenshot illustrates the History sub-category.



- 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

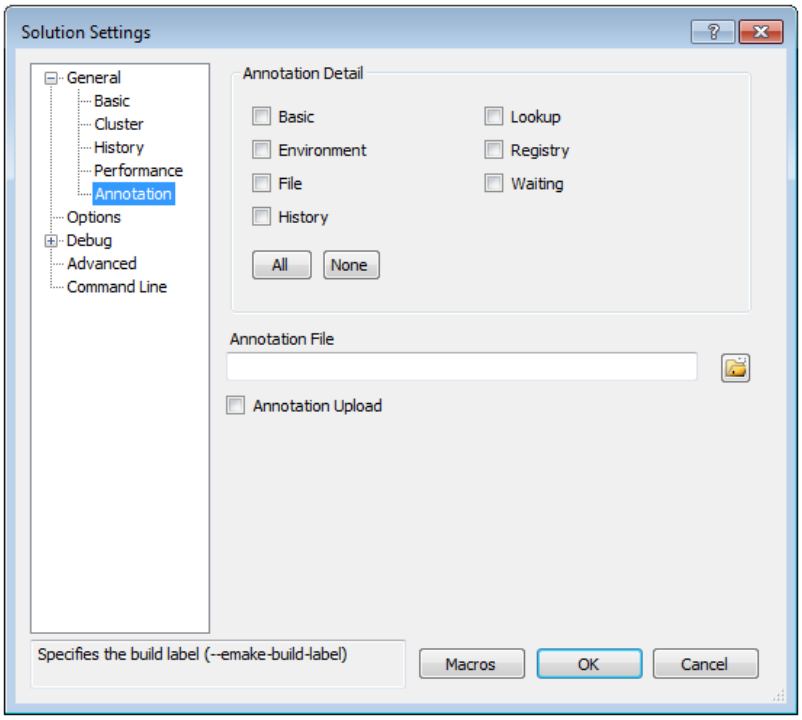
The following screenshot illustrates the Performance sub-category.



- 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_ORTDER_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.

Annotation

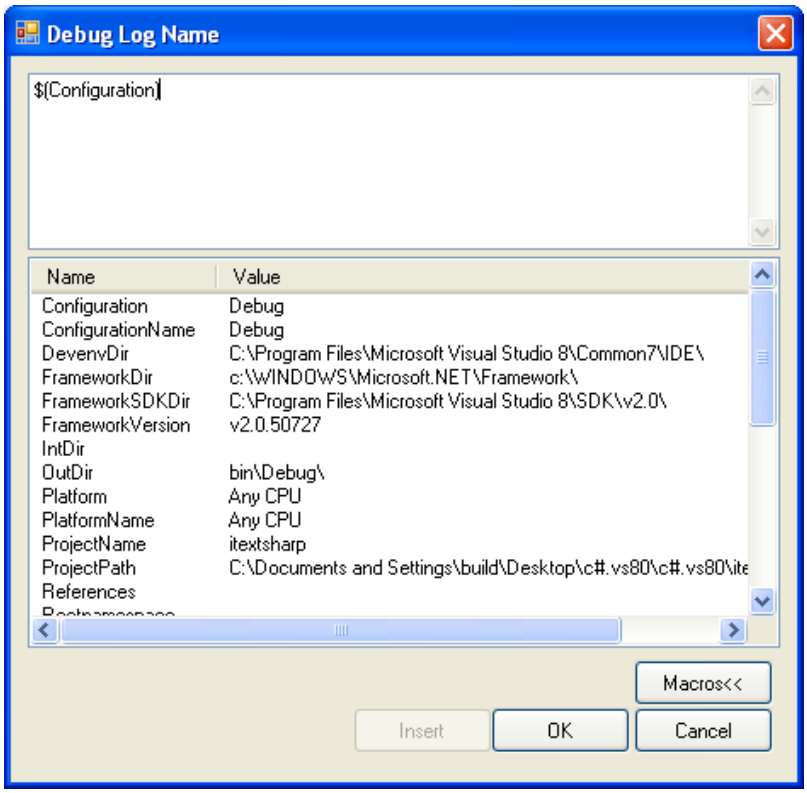
The following screenshot illustrates the Annotation sub-category.



- 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`).

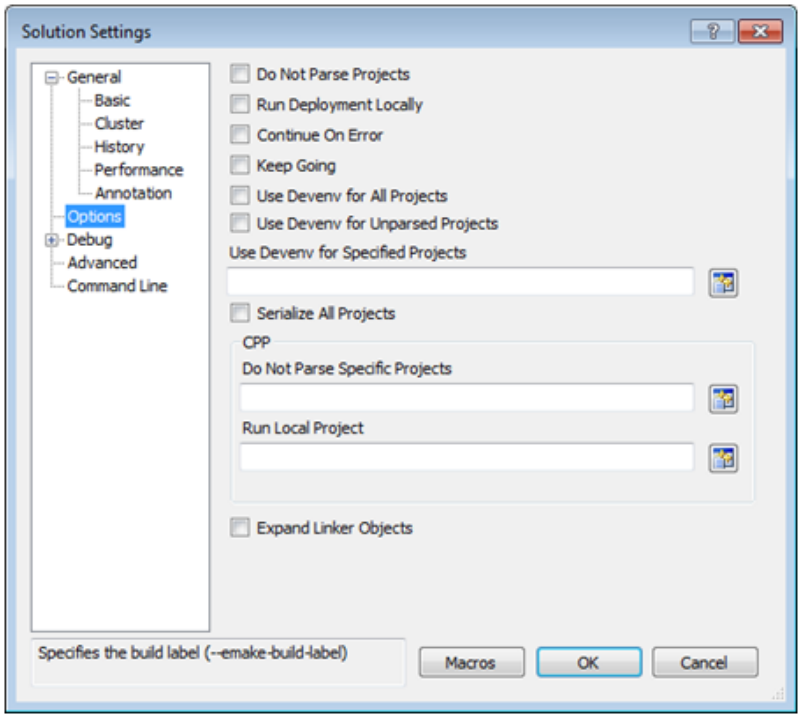
Using Macros

If file name settings include any variables that contain invalid DOS file name characters, such as a '\' [a backslash] or ':' [a colon], this will result in an error at run-time.



Options

This category contains most frequently used optional settings.

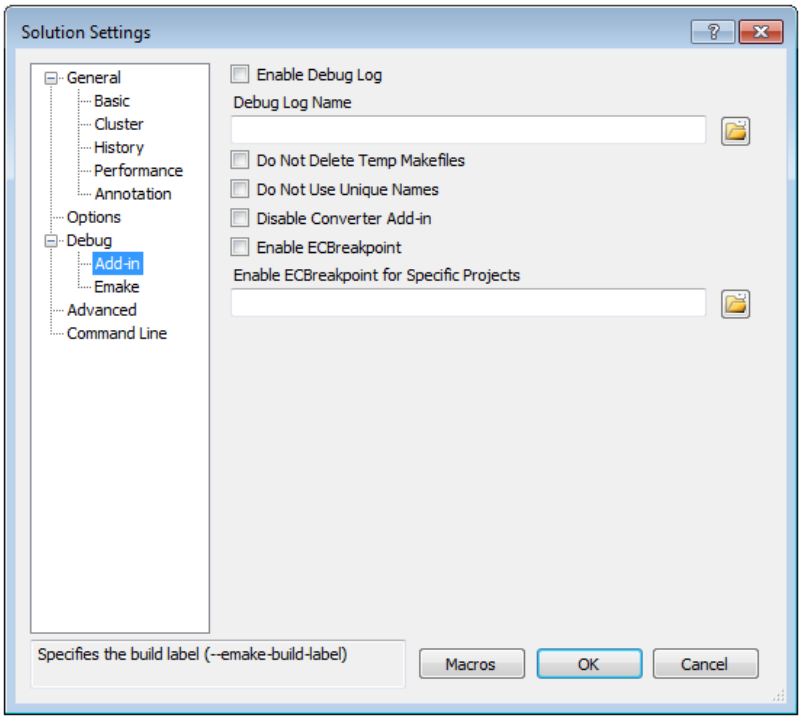


- Do Not Parse Projects - Determines whether to prevent the VS Converter Add-in from breaking up C++ projects (sets `ECADDIN_DONT_PARSE_PROJECTS=true` if enabled).
- Run Deployment Locally - Determines whether to run deployment projects locally using `#pragma runlocal` (sets `ECADDIN_RUN_DEPLOYMENT_PROJECTS_LOCALLY=true` if enabled).
- Continue On Error - Determines whether to continue when an error occurs (adds `/I` to the eMake call).
- Keep Going - Determines whether to keep going when an error occurs (adds `/k` to the eMake call).
- 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=`).
- Use Devenv for Specified Projects - Run specified projects using devenv, not msbuild (sets `ECADDIN_USE_DEVENV_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`).
- Run Local Project - Runs specified projects locally using `#pragma runlocal` (sets `ECADDIN_RUN_LOCAL_PROJECT`).
- Expand Linker Objects - Determines whether to expand linker objects to full pathnames (sets `ECADDIN_EXPAND_LINKER_OBJECTS=true` if enabled).

Debug

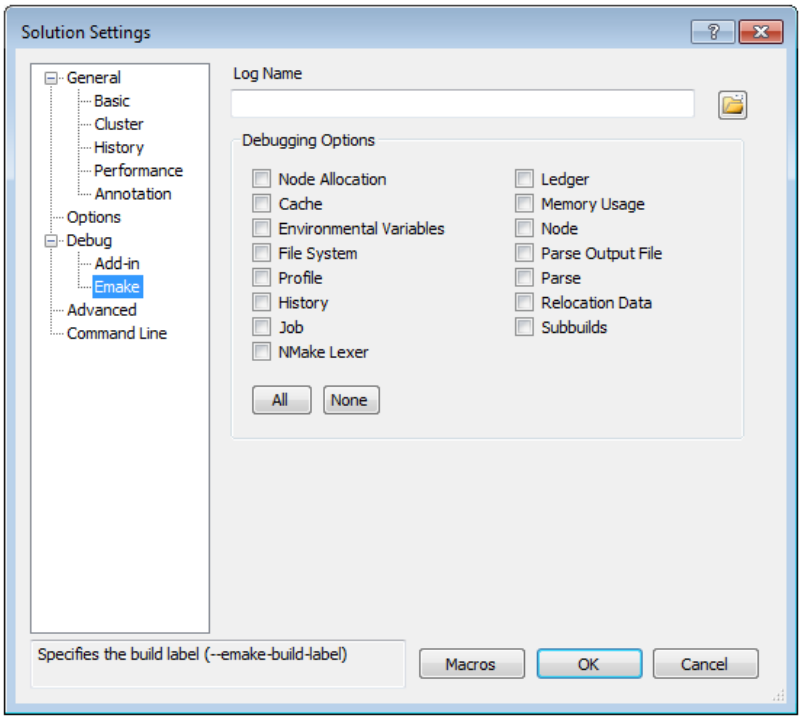
This category contains debug options for the Add-in and eMake.

Add-in



- 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`). Default is `C:\ecdebug<unique>.log`. 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.

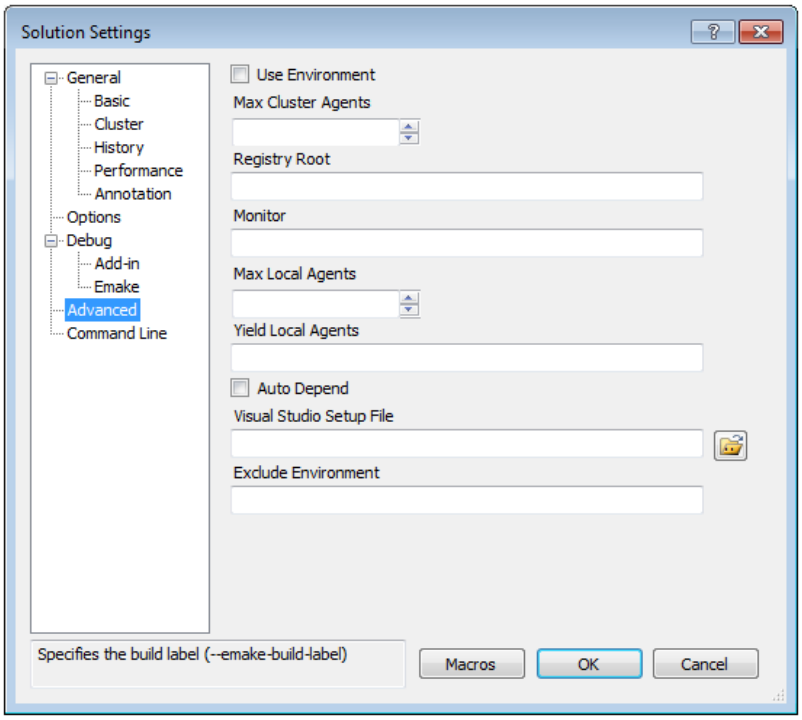
eMake



- Log Name - Specifies the name of the debug log (sets `--emake-logfile`). Use the folder button to select a log file.
- Debugging Options - Select debug categories to log.

Advanced

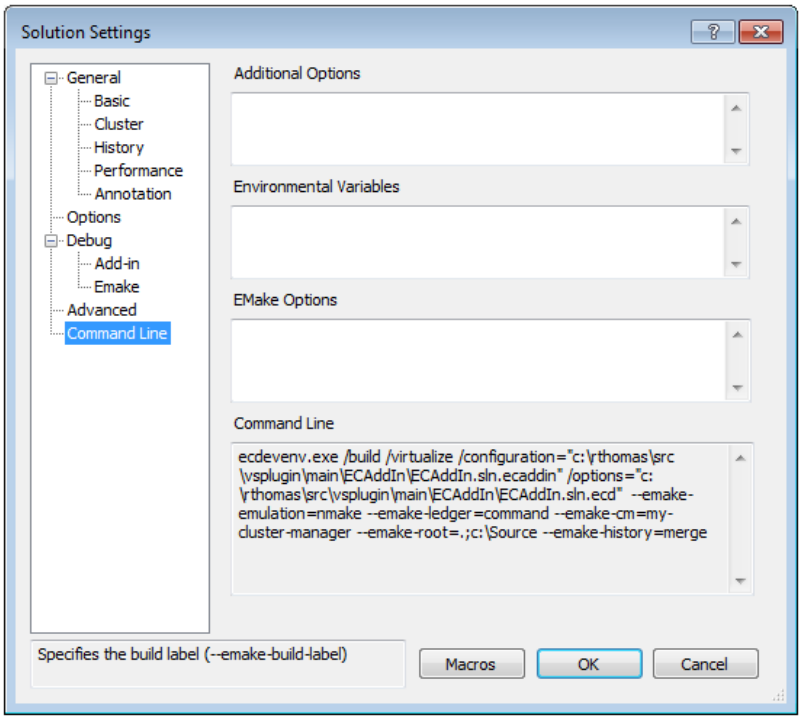
This category contains advanced options.



- 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/disables allowing eMake to automatically determine dependencies. Default is enabled.
- 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

This category allows you to 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 page 9-1
- 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.exe Utility

This version of the Visual Studio integration contains an extensively redesigned version of ecdevenv. The previous version of ecdevenv simply provided a mechanism to skip the generation of NMAKE makefiles. Ecdevenv is now a drop-in replacement for devenv.exe that will build Visual Studio solutions and projects using eMake.

Key ecdevenv features:

- Skip generation of NMAKE makefiles
- No makefiles 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

Ecdevenv is backward compatible, so the old mode of operation is unchanged.

No Makefiles Required

Previously, users had to 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
```

Now you can simply call ecdevenv:

```
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

Visual Studio Toolchain Virtualization

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. You must, however, install the following items:

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

Build Multiple Solutions and Projects

Ecdevenv can also build multiple solutions.

Create a makefile in the following format and optionally specify the projects and project configurations. Then pass the name of the file to ecdevenv using `/configuration=<filename>`.

```
<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>
```

You may not specify a solution or project on the command line when using the `/configuration` option.

Add-in Options

Use `/options=<file>` to specify the add-in options. The file uses the same format as shown on page 9-1

Force

Use `/force` to force the regeneration of makefiles.

Generate Only

Use `/generate` to generate the NMAKE makefiles only (that is, do not run eMake).

Other Options

- 64 bit systems - Use `/use64bit` to use the 64-bit version of eMake.
- Help - Use `/help` to display a list of the commands to ecdevenv and example option and configuration files
- Makefile - Specify an alternative default makefile (rather than ecdevenv.mak) using `/makefile`.

Debugging

To turn on debugging to stdout, set `ECDEVENV_DEBUG=true`. To redirect to a file, set `ECADDIN_DEBUG_LOG=<filename>`.

Miscellaneous Files Generated by ecdevenv

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

- `*.last_build`

  "*.last_build" files are created in the same directory the solution files. One is created for each invocation of ecdevenv. Ecdevenv uses this file to determine whether it needs to regenerate the temp makefiles. Only delete these if you want to force ecdevenv to regenerate makefiles

- `metrics.sln.Release_Win32.ecmak`
  `metrics.vcxproj.Release_Win32.ecmak`

  Temporary makefiles generated by ecdevenv. Do not delete unless you are regenerating the temp 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 filename) Do not delete unless you are regenerating the temp makefiles.

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

  Visual Studio files. Do not delete.

- `ecdevenv.mak`

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

Chapter 8: Visual Studio 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 version of .NET and redistributables are installed.

`devenv` must be in the PATH ***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 whatever options you currently use. Ensure the correct version of `devenv` is in your path:

```
devenv /?
```

and 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 Visual Studio Environment Variables

You can control the way the VS Converter Add-in works by setting these environment variables on the Electric Make machine.

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

For example:

```
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>
```

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

Environment Variable	Variable	Description	Default
ECADDIN_BUILD_ORDER	BuildOrder	Specifies projects' build order. Use only if Accelerator is not building projects in the correct order.	-
ECADDIN_CHECK_ DLLEXPORT	CheckDLLExport	Prevents the linker from including libraries that do not contain any exports. This may be slow.	False
ECADDIN_ CONFIGURATION_FILE	n/a	Sets the filename of the configuration file. If this is set, ***all*** settings are taken from the file and the environment variables are ignored.	-
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 C:\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:\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	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 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_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	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 allows you to work around the problem. When using this variable, you may 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.	False
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).	-

Environment Variable	Variable	Description	Default
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 ectouch.exe. See Setting Visual Studio Environment Variables.	False
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

Environment Variable	Variable	Description	Default
ECADDIN_INCLUDE_CMAKELISTS	IncludeCMakeLists	Excludes any source file with the name CMakeLists.txt. Set this variable to True to execute the file.	False
ECADDIN_MAX_PDB_FILES	MaxPDBFiles	Specifies the maximum number of PDB files produced. See Optimize 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_NORMALIZE_PATHS	NormalizePaths	Normalizes all paths in the makefile.	True
ECADDIN_REMOVE_DEPENDENCIES	RemoveDependencies	Removes project-to-project dependencies to improve parallelization.	True
ECADDIN_REMOVE_DEPENDENCIES	RemoveDependencies	Removes project dependencies.	True
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	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 Electric Make 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, 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.	False
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) for specified projects in the list.	-
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.	-

Environment Variable	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	Allows you to 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.	-

Chapter 10: Performance Tuning

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:

- Improve Build Time for /Zi + PCH Builds
- Improve Build Time for Solutions with Many Projects
- Improve Final Link Time
- Improve Incremental Build Time
- Improve Incremental Linking Time
- Optimize Parallelization Using PDB Splitting

Improve 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 may 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 may 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.

Improve Build Time for Solutions with Many Projects

Some very large solutions with few inter-project dependencies may 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 may reduce the overall build time.

Improve 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` may 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.

Improve Incremental Build Time

By default, the add-in always rebuilds prebuild events. If you have a prebuild event that touches files, it could potentially rebuild far more than Visual Studio would. To prevent this, set `ECADDIN_UP_TO_DATE_CHECK=true`. This first checks whether there is anything out of date. If not, nothing will be built including the prebuild event.

Improve Incremental Linking Time

Using the Add-in

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

To work around this problem, we can “touch” the exe after the link with its current timestamp. This explicit modification of the timestamp instructs eMake to preserve the timestamp, hence keeping the validity of its incremental status.

To enable this feature with the add-in, set `ECADDIN_ENABLE_INCREMENTAL_LINK=true`. This inserts a call to `ectouch.exe`, which performs the action stated above. `ectouch.exe` should be installed in the `PATH`.

Not Using 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 the `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`.)

Optimize Parallelization Using PDB Splitting

Using 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 may 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`.

Not Using 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)
$(PCB_INCLUDE) $< /Fo$@ /Fd$(shell ${path-to-hashstr}/hashstr.exe "$@"
${hashstr-modulus}).pdb
```

Chapter 11: Using MSBuild

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.

Chapter 12: Visual Studio 6

Although Visual Studio 6 does not have an add-in, you can parallelize such builds using the built-in option to generate NMAKE makefiles. See http://ask.electric-cloud.com/questions/441/how-do-i-export-visual-studio-6-projects.

One-Time Export

Consider exporting make files (.mak) for the project and building the project using the NMAKE utility.

To export a makefile, select **Export Makefile** from the Project menu in the Developer Studio. This is a one-time export, after which makefiles and Project files would potentially diverge.

To avoid makefiles and Project files diverging, instruct Visual Studio 6 to export makefiles automatically whenever the project changes:

1. Go to the Tools menu and click **Options**.
2. On the Build tab, select **Export Makefile when saving the project file**.

On-the-fly Export

Makefiles can be generated from .dsw files via the command-line using the following technique documented on MSDN: http://msdn.microsoft.com/en-us/library/aa260829.aspx#autobld_export.

DevStudio provides some built-in commands for automation. To see a list of these commands, select **Customize** under the **Tools** menu and choose the **Keyboard** tab. The commands under the respective categories are listed. These commands can be launched in the same way that add-ins and macro commands are launched. To export a makefile for all projects in a workspace, do the following from the command line:

```
msdev.exe MyProject.dsw -execute BuildProjectExport
```

Note: The command supplied using the `-execute` option is case sensitive. DevStudio should be in the path; otherwise, specify full path in the preceding statement.

The above will launch DevStudio, open the MyProject.dsw workspace file, and then export the makefile for all of the projects. There is a drawback in doing this—you will see the Export Makefile dialog box. The same can also be accomplished using the following VBScript macro:

```
Sub ExportMakFile
    'Export the makefile
    ExecuteCommand "BuildProjectExport"
    'Close the workspace
    ExecuteCommand "WorkspaceClose"
    'Exit from DevStudio
```

```
    Quit
End Sub
```

Chapter 13: Debugging

First steps to take when debugging a failed build:

- Double-check that the build works under Visual Studio.
- Check Troubleshooting and ask.electric-cloud.com.

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

1. Set `ECADDIN_DEBUG=true` and `ECADDIN_DEBUG_LOG_NAME=<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

Make Sure You Initialize 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, etc.

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

1. Go to Tools > Options, Project and Solutions > Build and Run.
2. Set the maximum number of parallel project builds to 1.
3. Go to Help > Customer Feedback Options.
4. Initialize the 'Customer Experience Improvement Program' to either yes or no.

If you continue to 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 commonly encountered issues after you verify that Visual Studio has been 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
- For VS2010/msbuild 4.0 builds, each project is taking around 15 minutes
- 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 Add-in is installed, but the Electric Cloud menu is missing and the Tools menu item is corrupted (shows "Electric Cloud").

The add-in may 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 to reflect the installation directory on the agent.

For VS2010/msbuild 4.0 builds, each project is taking around 15 minutes

Cause

MSBuild v4.0 does not exit immediately after a build for performance reasons.

Solution

Set `MSBUILDNODECONNECTIONTIMEOUT=0`.

See http://ask.electric-cloud.com/questions/34/msbuild-processes-are-slow-and-takes-long-time-to-finish.

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 may 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
- 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 may 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 may result in 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 may 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

W