ElectricAccelerator
ElectricInsight User Guide

Version 5.0

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

ElectricAccelerator ElectricInsight 5.0

Published 7/27/2015

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

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

Trademarks

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

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

All other trademarks used herein are the property of their respective owners.

Contents

Chapter 1: Introduction

Until now, there has been little visibility into builds to “see” why a build was slow, why a build broke, or which dependencies were involved. ElectricInsight® (Insight) removes the “black box” around software product builds and provides easy-to-understand performance data.

About ElectricInsight

Insight depicts how a build is structured and run, empowering build managers to pinpoint performance problems or conflicts in a parallel build. Developed to work with ElectricAccelerator®, Insight mines information produced by Electric Make® to provide an easy-to-understand, graphical representation of the build structure for performance analysis.

Insight provides detailed information and reports for at-a-glance diagnostics for each job on each host in the build cluster. Users can identify which jobs are performing, when, and with which files. Instead of manually reading through tens of thousands of lines of information in log files, error detection and performance tuning that used to consume hours or days can now occur in a few minutes or seconds.

By observing and tracing serialization sources or slowdowns, you can fine tune your build for maximum speed. Insight gives you the ability to pinpoint areas to improve in your build process—and you will have answers for these questions:

- Are any unnecessary serializations occurring?
- Is there a gap where one or more agents is not busy?
- Which job chains are the longest and can they be shortened?
- Which files are being modified (created, updated, or deleted) multiple times?

Documentation

The *ElectricInsight User Guide* (this document) is available online. To view the user guide, choose **Help > User Guide** in the ElectricInsight application or browse to http://docs.electric-cloud.com/insight_doc/5_0/PDF/InsightUserGuide_5_0.pdf. The *ElectricInsight Release Notes* are available at http://docs.electric-cloud.com/insight_doc/5_0/PDF/InsightReleaseNotes_5_0.pdf.

ElectricAccelerator documentation is available online at http://docs.electric-cloud.com/accelerator_doc/AcceleratorIndex.html:

- *ElectricAccelerator Installation and Configuration Guide*
- *ElectricAccelerator Electric Make User Guide*
- *ElectricAccelerator cmtool Reference Guide*

- *ElectricAccelerator Visual Studio Integration Guide*
- *ElectricAccelerator Release Notes*
- PDF, HTML, and mobile-optimized HTML versions of the online help that is also built into the Cluster Manager

The documents listed above are updated periodically.

Chapter 2: Installing ElectricInsight

The Insight distribution includes the following main files:

- einsight.exe (Windows) or einsight (Linux)—The ElectricInsight program
- annolib.dll (Windows) or annolib.so (Linux)—The annolib library

Because Insight analyzes build annotation files, it is simplest to install it on the system where you run Electric Make or on a Cluster Manager host where you can upload build annotation files.

Installer Files

ElectricInsight is available in 32- and 64-bit versions on Windows and on Linux. Electric Cloud recommends the 64-bit version for loading large annotation files or for 64-bit Linux systems that do not have 32-bit libraries (such as ia32-lib) installed.

Note: The live build monitoring feature is not available in 64-bit ElectricInsight.

Three installer files are available:

- 32- and 64-bit Windows
- 32-bit Linux
- 64-bit Linux

Each installer file installs both the 32-bit executables and the 64-bit executables. They are installed in the following locations.

- 32-bit executables are installed in <*InstallDir*>/<*arch*>/bin. <*InstallDir*> is C:\ECloud by default on Windows and /opt/ecloud by default on Linux. <*arch*>is i686_win32 on Windows and i686_Linux on Linux.
- 64-bit executables are installed in <*InstallDir*>/<*arch*>/64/bin.
    - On Windows, for example, if you install in the C:\ECloud directory, the 64-bit executables are installed in C:\ECloud\i686_win32\64\bin.
    - On Linux, for example, if you install in the /opt/ecloud directory, the 64-bit executables are installed in /opt/ecloud/i686_Linux/64/bin.

The installers set up the menu option for invoking ElectricInsight to point to the proper executable based on your system's bitness. Also, on Windows, a shortcut pointing to the proper executable is created on your desktop.

Installation Log File

The installation log file is named einsight_install.log. By default, the file is in the following location.

- Windows: The folder defined by the `TEMP` environment variable (typically C:\Users\<*username*>\AppData\Local\Temp)
- Linux: The /usr/tmp directory

Library Requirements for Linux Systems

On Linux systems, ElectricInsight requires the following libraries:

libX11.so.6
libXss.so.1
libXext.so.6
libXft.so.2
libfreetype.so.6
libfontconfig.so.1
libXrender.so.1
libxcb.so.1
libexpat.so.1
libXau.so.6

Installing ElectricInsight Using the GUI

To install Insight on Windows or Linux using the GUI:

1. Run the installer file to begin installation.

   For Windows: `ElectricInsight-<version>-Windows-Install.exe`
   You must run the installer as administrator. For example, on Windows systems running Windows 2008 or later, right-click the installer and click **Run as administrator**.

   For Linux (32-bit): `ElectricInsight-<version>-Linux-x86-Install`

   For Linux (64-bit): `ElectricInsight-<version>-Linux-x86_64-Install`

2. Click **Next** on the Welcome screen.
3. Accept the default installation location or click **Browse** to choose a different location, and then click **Next**.
4. Click **Browse** to locate your license file, and click **Next** to continue.

   The installer copies the license file to both <*InstallDir*>/<*arch*>/bin/ and <*InstallDir*>/<*arch*>/64/bin.

   If you want to provide the license file after installation, you must name it license.xml and copy it to the appropriate bin directory for your bitness: <*InstallDir*>/<*arch*>/bin or <*InstallDir*>/<*arch*>/64/bin.

5. Review your settings and click **Next** to continue or **Back** to make changes.
6. Click **Finish** to close the installer.

Installing ElectricInsight Using the Console with Installation Options

You can use multiple options to run an installation with customized settings and minimal interaction. Use this syntax:

```
<installer filename> [options]
```

The following options are available.

Option	Description
`/help`	Displays help information.

`/licensefile`	Sets the license file that the installer imports. Type the full path to the license file. For example:
	`ElectricInsight-5.0.0.5101-Windows-Install.exe /licensefile C:\mylicenses`
	or
	`ElectricInsight-5.0.0.5101-Linux-x86_64 /licensefile /opt/license.xml`

`/mode` *`<mode>`*	Sets the installation mode. The following modes are available. • `console`—(Linux only) Runs the installation in the console (not the GUI). • `standard`—Runs the standard installation mode, which includes all user prompts • `silent`—Runs the installation without user prompts • `default`—Runs the installation while accepting all installation defaults, resulting in fewer user prompts

`/prefix`	Sets the installation directory. Type the full path. For example:
	`ElectricInsight-5.0.0.5101-Windows-Install.exe /prefix C:\ecloud2`
	or
	`ElectricInsight-5.0.0.5101-Linux-x86_64 /prefix /opt/ecloud2`

`/temp`	Sets the temporary directory used by the installer.
`/version`	Displays installer version information.

For example, to run in silent mode, specify the license file location, and specify a nondefault installation directory, enter

```
ElectricInsight-5.0.0.5101-Windows-Install.exe /mode silent /licensefile C:\mylicen
ses /prefix C:\ecloud2
```

Installing ElectricInsight Using the Console with No Installation Options (Linux Only)

To install Insight on Linux using the console with no installation options:

1. Run the installer file.

   32-bit: `ElectricInsight-`*`<version>`*`-Linux-x86-Install`

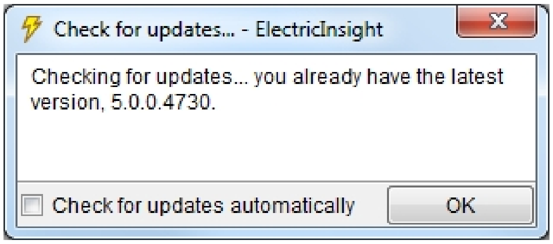
   64-bit: `ElectricInsight-`*`<version>`*`-Linux-x86_64-Install`

2. Press **Enter** to start the installation.
3. Press **Enter** to accept the default installation location (`/opt/ecloud/`) or enter a different location.
4. Type the location of your license file and press **Enter**.

Your installation is complete.

Checking for Software Updates

ElectricInsight is updated periodically to improve its functionality or to fix bugs. To see if an update is available, click **Help > Check for updates…**.



Check for updates... - ElectricInsight

Checking for updates... you already have the latest version, 5.0.0.4730.

□ Check for updates automatically
OK

If you want to make these checks automatic, check the **Check for updates automatically** checkbox and click **OK**.

Chapter 3: Getting Started

The following sections discuss what you need to know to start using Insight.

Topics:

- Annotation
- Starting ElectricInsight
- Navigating the Interface

Annotation

ElectricInsight gathers information from build annotation files to create a picture of your build. To use Insight, you must first run your builds with the `--emake-annodetail` parameter.

For example, on the eMake command line, run:

```
% emake ... --emake-annodetail=file,waiting,history ...
```

Note: Collecting annotation information might cause your build to run approximately 5% slower.

Supported Annotation Detail Levels

`basic`	Detailed information about each “job” in the build, including command arguments, output, exit code, timing, and source location. Basic annotation includes annotation for the JobCache add-on, which lets a build avoid recompiling object files that it previously built, if their inputs have not changed.
`env`	Environment variable modification information
`file`	Filesystem operations information, excluding lookups
`history`	Information about implicit dependencies discovered by Electric Make
`lookup`	Filesystem operations information, including lookups
`md5`	MD5 checksums for files read and written by the build
`registry`	Registry operations information
`waiting`	Information required to reconstruct the complete dependency graph

For more information about annotation or about the JobCache add-on, see the *ElectricAccelerator Electric Make User Guide* at http://docs.electric-cloud.com/accelerator_doc/AcceleratorIndex.html.

Annotation File Splitting

Because of limitations in the 32-bit version of the ElectricInsight tool, eMake as well as Electrify automatically partition annotation files into 1.6 GB “chunks.” The first chunk is named using the file name that you specify with the `--emake-annofile` option or with “emake.xml,” if `--emake-annofile` is not specified. The second chunk uses that name as the base but adds the suffix _1, the third chunk adds the suffix _2, and so on. For example, a four-part annotation file might consist of files named emake.xml, emake.xml_1, emake.xml_2, and emake.xml_3.

No special action is required to load a multipart annotation file into ElectricInsight. If all parts are present in the same directory, ElectricInsight automatically finds and loads the content of each file—simply specify the name of the first chunk when opening the file in ElectricInsight.

For loading large annotation files, Electric Cloud recommends the 64-bit version of ElectricInsight.

Starting ElectricInsight

To start ElectricInsight:

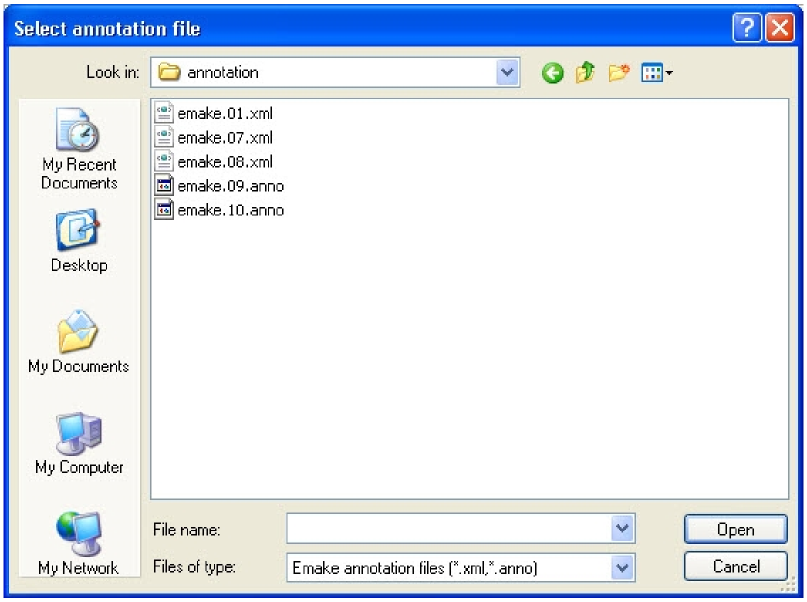
1. Click **Start > All Programs > Electric Cloud > ElectricInsight**.

   or

Type `einsight` at the command prompt. Optionally, you can specify the target annotation file.

```
% einsight [build-annotation-file]
```

2. If you did not specify a build annotation file on the command line, click the folder icon on the toolbar or click **File > Load annotation…**.



Look in: annotation

emake.01.xml
emake.07.xml
emake.08.xml
emake.09.anno
emake.10.anno

File name:

Files of type: Emake annotation files (*.xml,*.anno)

3. Navigate to the annotation file of a completed build that you want to analyze and click **Open**.

   Your build is now displayed.

Navigating the Interface

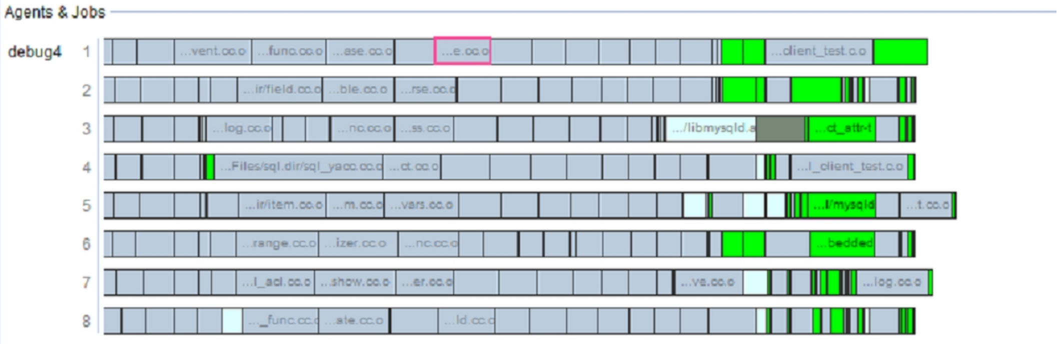
This section will familiarize you with the various portions of Insight's navigation.

Topics:

- Agents & Jobs
- Navigation
- Overview
- Legend
- Job Summary
- Zoom
- Replay
- Monitor Live Build

Agents & Jobs

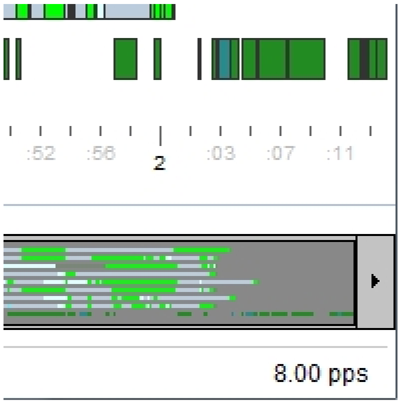
In the Agents & Jobs section, each agent used in the build is represented by one row. Host names and agent number designations appear to the left of the agent job bars. Use the view frame and arrows in the Navigation section to view agents that are not currently visible.



The section's x-axis is time, with the build starting at the left. The first job completed by each agent is at the extreme left of an agent's job bar. As you scroll to the right on an agent's job bar, you see the progress of jobs assigned to the agent.

The time grid helps you determine how long jobs take. The grid has major and minor grid lines. By default, major grid lines are 60 seconds (pixels) apart and minor grid lines are 15 seconds (pixels) apart. Build magnification is measured in pixels per second (pps). As you zoom in or out, this ratio increases or decreases.

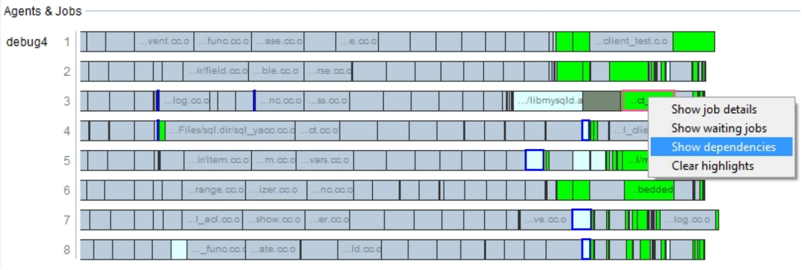
Magnification is displayed in the lower right corner. In the example below, magnification is 8.00 pps.



To see how long some jobs took to complete, click and drag from the left edge of the first job to the end of the last job of interest. The job ruler appears and displays a time measurement. You can also use the ruler by dragging from right to left.

Dependencies and Waiting Jobs Popup

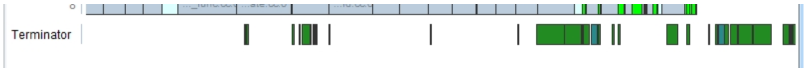
You can right-click a job and click **Show dependencies** or **Show waiting jobs**. Dependencies are outlined in blue. Waiting jobs (not present in the screenshot) are outlined in red.



For dependency details between a pair of jobs, load an annotation file that contains waiting and history level annotation and run the Serialization Analysis report.

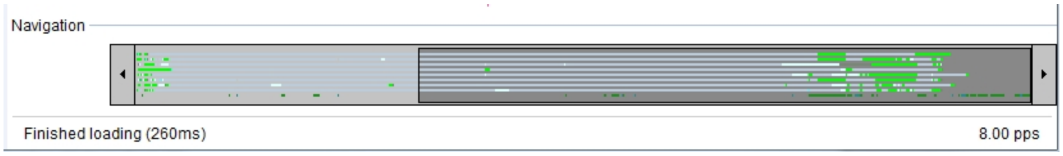
Terminator Activity

The terminator's activity (commit, write-to-disk, and wait-for-completion) appears on the row labeled Terminator. The terminator activity on a per-job basis is typically small.



Navigation

In the Navigation section, the view frame indicates which part of the build the Agents & Jobs section currently displays. If the entire build is displayed, the view frame is the same size as the scroll bar. Drag the frame to view different build stages. You can also use the arrows on either side of the navigation bar to move the view frame.



Finished loading (260ms)

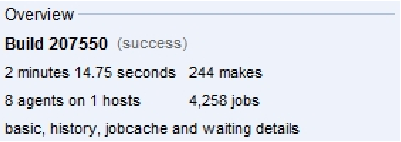
8.00 pps

Below the Navigation section is a field showing how long it took Insight to load build annotation information.

Overview

The Overview section displays the following build information:

- Build ID and whether it was successful
- Build duration
- Number of makes in the build
- Number of agents and hosts used by the build
- Number of jobs in the build
- Annotation details collected in the build



Overview

Build 207550 (success)

2 minutes 14.75 seconds 244 makes

8 agents on 1 hosts 4,258 jobs

basic, history, jobcache and waiting details

Keep in mind the following terminology:

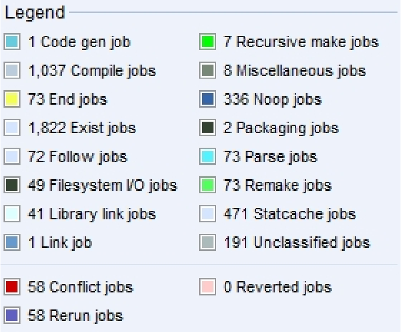
- Each makefile has one or more *rules* (lines of text).
- A *target* is the *rule* output.
- A *command* is a single shell invocation in a *rule*.
- A *job* corresponds to a *rule* scheduled as part of a build—in most cases.

- There are jobs that parse a makefile. Jobs also have status—possible status values include *normal*, *conflict*, *rerun*, *reverted*, and *skipped*. For more details about job status, see Supported Search Fields.

Note: You can hide the Overview, Legend, and Job Summary sections, by clicking the gray bar between those sections and the Agents & Jobs section. Click it again to show the hidden sections.

Legend

The Legend section displays the total number of jobs for each type and status within the build. Click a job type or status to list all of its instances within the build. Legend colors correspond to the job type and status for individual jobs within agent job bars.



Legend

1 Code gen job
1,037 Compile jobs
73 End jobs
1,822 Exist jobs
72 Follow jobs
49 Filesystem I/O jobs
41 Library link jobs
1 Link job

7 Recursive make jobs
8 Miscellaneous jobs
336 Noop jobs
2 Packaging jobs
73 Parse jobs
73 Remake jobs
471 Statcache jobs
191 Unclassified jobs

58 Conflict jobs
58 Rerun jobs

0 Reverted jobs

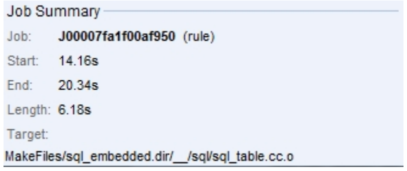
The Agents & Jobs section displays two dimensions of job data—job status and job type. If the job status is not unusual, then only the job type (such as parse or rule) is displayed. If a job has an unusual status (such as conflict or reverted), that color is used for the lower portion of the job and the upper portion retains the color of the job type.

Job Summary

To display a job's summary information, mouse over any job in an agent job bar. The selected job is outlined in pink in the agent job bar.

The Job Summary section displays the following information about the selected job:

- Job ID and job type or status (for additional information about job types and status, see Supported Search Fields)
- Start time (as an offset from the build start time)
- End time
- Job length in hours/minutes/seconds
- Output target name, if applicable



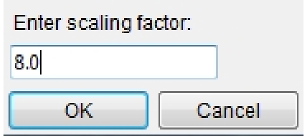
Job Summary

Job: **J00007fa1f00af950** (rule)
Start: 14.16s
End: 20.34s
Length: 6.18s
Target:
MakeFiles/sql_embedded.dir/__/sql/sql_table.cc.o

Zoom

The Tools menu contains the following magnification options:

- **Zoom in** and **Zoom out**—you can also zoom in or out using the magnifying glass icons in the toolbar or by pressing CTRL-[equal sign] or CTRL-[minus sign].
- **Zoom to…**—lets you specify a magnification factor for the display. To return to the initial, unzoomed view, click **Tools > Zoom to…**, enter a zoom factor of 1.0, then click **OK**.


Enter scaling factor: 8.0

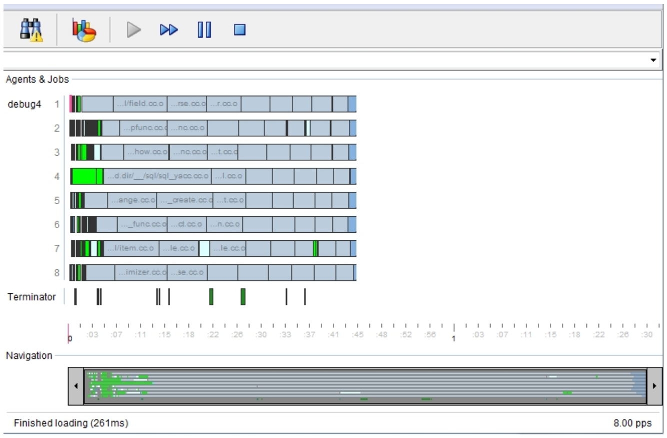
- **Zoom to fit—**sizes the display to fit the available space. Doing this might leave empty space to the left of all agent job bars, which depicts the time when eMake was parsing the makefile, before agents were assigned jobs. It is also possible that agents you are not currently viewing were working, while agents you are viewing were not assigned to this build.

Replay

Insight lets you replay your build in the Agents & Jobs section so you can observe its progress.

You can display the replay toolbar by clicking **Replay > Show toolbar**.

The following screenshot shows the replay toolbar and a build replay in progress.



Finished loading (261ms)
8.00 pps

Replay controls:

- Play—starts the replay.
- Fast forward—increases replay speed. You can click fast forward up to three times to speed up the replay. Clicking a fourth time returns the speed to normal.
- Pause—pauses the replay. Click play to start again.
- Stop—stops the replay.

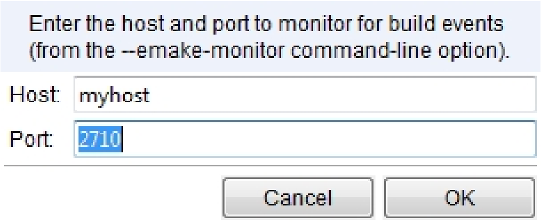
Monitor Live Build

Insight lets you monitor a live build in the Agents & Jobs section.

Note: Live build monitoring is not available on the 64-bit version of ElectricInsight.

Follow these steps:

1. Click **File > Monitor live build…** to display the following dialog.



Enter the host and port to monitor for build events (from the --emake-monitor command-line option).

Host: myhost

Port: 2710

2. Provide the host and port information from the `--emake-monitor` command-line option:
   - **Host** is the hostname or IP address of the Insight machine where the build will be monitored.
   - **Port** is the port number of the Insight machine that listens for the data.
3. Click **OK**.
4. Make sure `--emake-monitor=<hostname/IP>:<port>` is added to the command line.
5. Start the build.

   Note: To monitor a live build, you must launch the Insight live build monitor before you start the build.

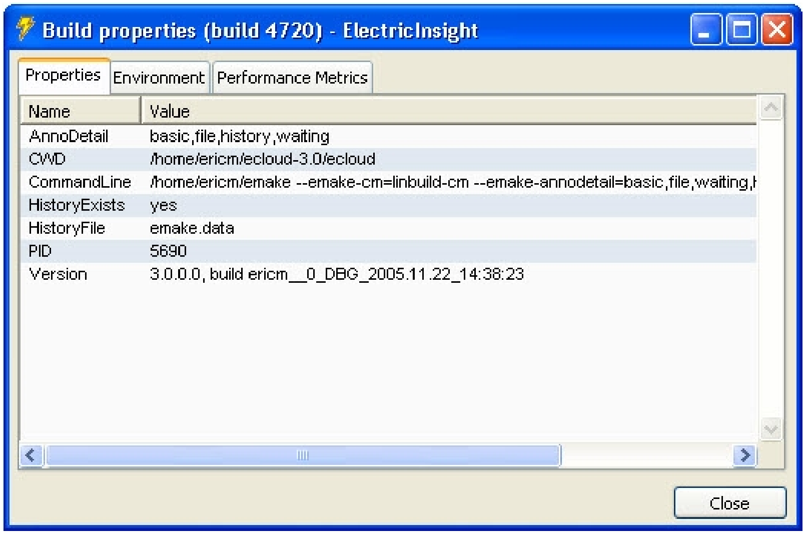
Chapter 4: Build, Job, and Make Information

This section discusses how to view detailed build, job, and make information.

Topics:

Build Properties

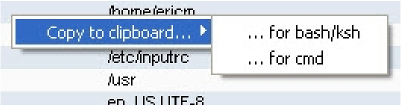
To view overall build properties, click **File > Build Properties**…. Three tabs are available: Properties, Environment, and Performance Metrics.



Name	Value
AnnoDetail	basic,file,history,waiting
CWD	/home/ericm/ecloud-3.0/ecloud
CommandLine	/home/ericm/emake --emake-cm=linbuild-cm --emake-annodetail=basic,file,waiting,
HistoryExists	yes
HistoryFile	emake.data
PID	5690
Version	3.0.0.0, build ericm__0_DBG_2005.11.22_14:38:23

The Environment tab displays build environment information, which you can copy to the clipboard:

1. Right-click anywhere in the window.
2. Click **Copy to clipboard**.
3. Click **…for bash/ksh** or **…for cmd**.



Job Details Dialog

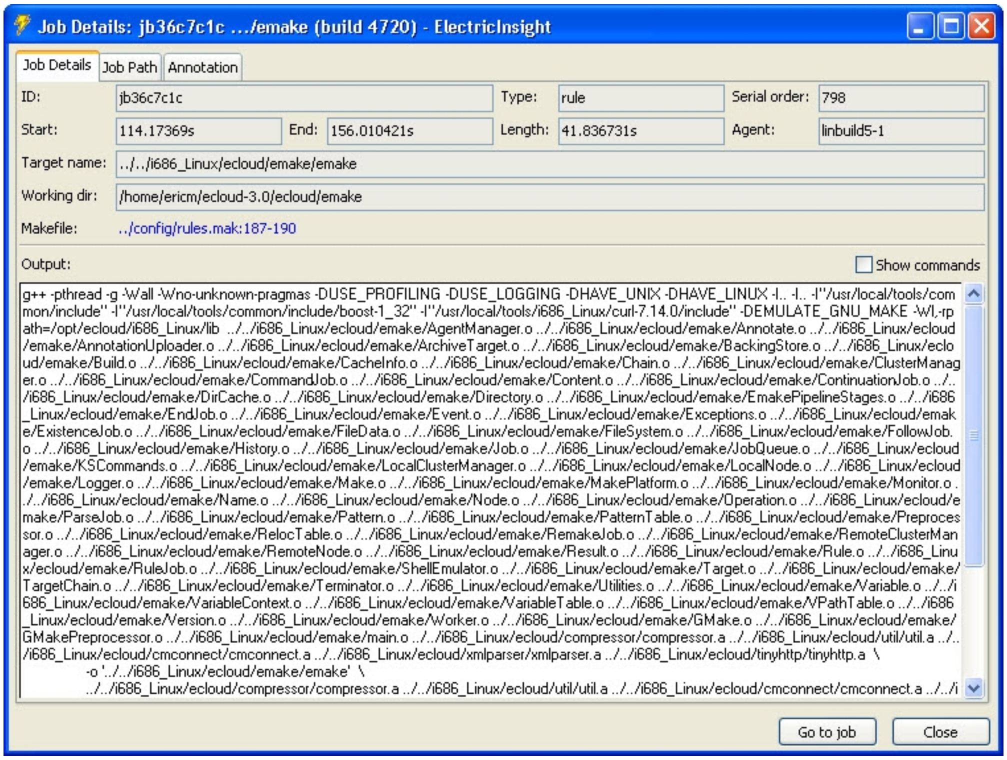
To open the Job Details dialog, double-click a job in the Agents & Jobs section. Three tabs are always available: Job Details, Job Path, and Annotation. The Submakes tab is available if the job includes submakes. The Environment tab is available if you specified the `env` annotation level.

Topics:

- Job Details
- Job Path
- Annotation
- Environment
- Submakes

Job Details

This tab displays summary information about the job. The following fields contain the same information as they do on the main screen: ID, Type, Start, End, and Length.


Job Details: jb36c7c1c .../emake (build 4720) - ElectricInsight
Job Details
Job Path
Annotation
Go to job
Close

ID: jb36c7c1c
Type: rule
Serial order: 798
Start: 114.17369s
End: 156.010421s
Length: 41.836731s
Agent: linbuild5-1
Target name: ../../i686_Linux/ecloud/emake/emake
Working dir: /home/ericm/ecloud-3.0/ecloud/emake
Makefile: ../config/rules.mak:187-190
Output:
□ Show commands

```
g++ -pthread -g -Wall -Wno-unknown-pragmas -DUSE_PROFILING -DUSE_LOGGING -DHAVE_UNIX -DHAVE_LINUX -I.. -I.. -I"/usr/local/tools/com
mon/include" -I"/usr/local/tools/common/include/boost-1_32" -I"/usr/local/tools/i686_Linux/curl-7.14.0/include" -DEMULATE_GNU_MAKE -Wl,-rp
ath=/opt/ecloud/i686_Linux/lib  ../../i686_Linux/ecloud/emake/AgentManager.o ../../i686_Linux/ecloud/emake/Annotate.o ../../i686_Linux/ecloud
/emake/AnnotationUploader.o ../../i686_Linux/ecloud/emake/ArchiveTarget.o ../../i686_Linux/ecloud/emake/BackingStore.o ../../i686_Linux/eclo
ud/emake/Build.o ../../i686_Linux/ecloud/emake/CacheInfo.o ../../i686_Linux/ecloud/emake/Chain.o ../../i686_Linux/ecloud/emake/ClusterManag
er.o ../../i686_Linux/ecloud/emake/CommandJob.o ../../i686_Linux/ecloud/emake/Content.o ../../i686_Linux/ecloud/emake/ContinuationJob.o ../..
/i686_Linux/ecloud/emake/DirCache.o ../../i686_Linux/ecloud/emake/Directory.o ../../i686_Linux/ecloud/emake/EmakePipelineStages.o ../../i686
_Linux/ecloud/emake/EndJob.o ../../i686_Linux/ecloud/emake/Event.o ../../i686_Linux/ecloud/emake/Exceptions.o ../../i686_Linux/ecloud/emak
e/ExistenceJob.o ../../i686_Linux/ecloud/emake/FileData.o ../../i686_Linux/ecloud/emake/FileSystem.o ../../i686_Linux/ecloud/emake/FollowJob.
o ../../i686_Linux/ecloud/emake/History.o ../../i686_Linux/ecloud/emake/Job.o ../../i686_Linux/ecloud/emake/JobQueue.o ../../i686_Linux/ecloud
/emake/KSCommands.o ../../i686_Linux/ecloud/emake/LocalClusterManager.o ../../i686_Linux/ecloud/emake/LocalNode.o ../../i686_Linux/ecloud
/emake/Logger.o ../../i686_Linux/ecloud/emake/Make.o ../../i686_Linux/ecloud/emake/MakePlatform.o ../../i686_Linux/ecloud/emake/Monitor.o .
./../i686_Linux/ecloud/emake/Name.o ../../i686_Linux/ecloud/emake/Node.o ../../i686_Linux/ecloud/emake/Operation.o ../../i686_Linux/ecloud/e
make/ParseJob.o ../../i686_Linux/ecloud/emake/Pattern.o ../../i686_Linux/ecloud/emake/PatternTable.o ../../i686_Linux/ecloud/emake/Preproces
sor.o ../../i686_Linux/ecloud/emake/RelocTable.o ../../i686_Linux/ecloud/emake/RemakeJob.o ../../i686_Linux/ecloud/emake/RemoteClusterMan
ager.o ../../i686_Linux/ecloud/emake/RemoteNode.o ../../i686_Linux/ecloud/emake/Result.o ../../i686_Linux/ecloud/emake/Rule.o ../../i686_Linu
x/ecloud/emake/RuleJob.o ../../i686_Linux/ecloud/emake/ShellEmulator.o ../../i686_Linux/ecloud/emake/Target.o ../../i686_Linux/ecloud/emake/
TargetChain.o ../../i686_Linux/ecloud/emake/Terminator.o ../../i686_Linux/ecloud/emake/Utilities.o ../../i686_Linux/ecloud/emake/Variable.o ../../i
686_Linux/ecloud/emake/VariableContext.o ../../i686_Linux/ecloud/emake/VariableTable.o ../../i686_Linux/ecloud/emake/VPathTable.o ../../i686
_Linux/ecloud/emake/Version.o ../../i686_Linux/ecloud/emake/Worker.o ../../i686_Linux/ecloud/emake/GMake.o ../../i686_Linux/ecloud/emake/
GMakePreprocessor.o ../../i686_Linux/ecloud/emake/main.o ../../i686_Linux/ecloud/compressor/compressor.a ../../i686_Linux/ecloud/util/util.a ../..
/i686_Linux/ecloud/cmconnect/cmconnect.a ../../i686_Linux/ecloud/xmlparser/xmlparser.a ../../i686_Linux/ecloud/tinyhttp/tinyhttp.a  \
        -o '../../i686_Linux/ecloud/emake/emake'  \
        ../../i686_Linux/ecloud/compressor/compressor.a ../../i686_Linux/ecloud/util/util.a ../../i686_Linux/ecloud/cmconnect/cmconnect.a ../../i
```

For some job types, the **Target name**, **Makefile**, and **Output** fields are populated.

If the **Output** field is populated, you can enable **Show commands** to see the makefile line(s) that correspond to the job.

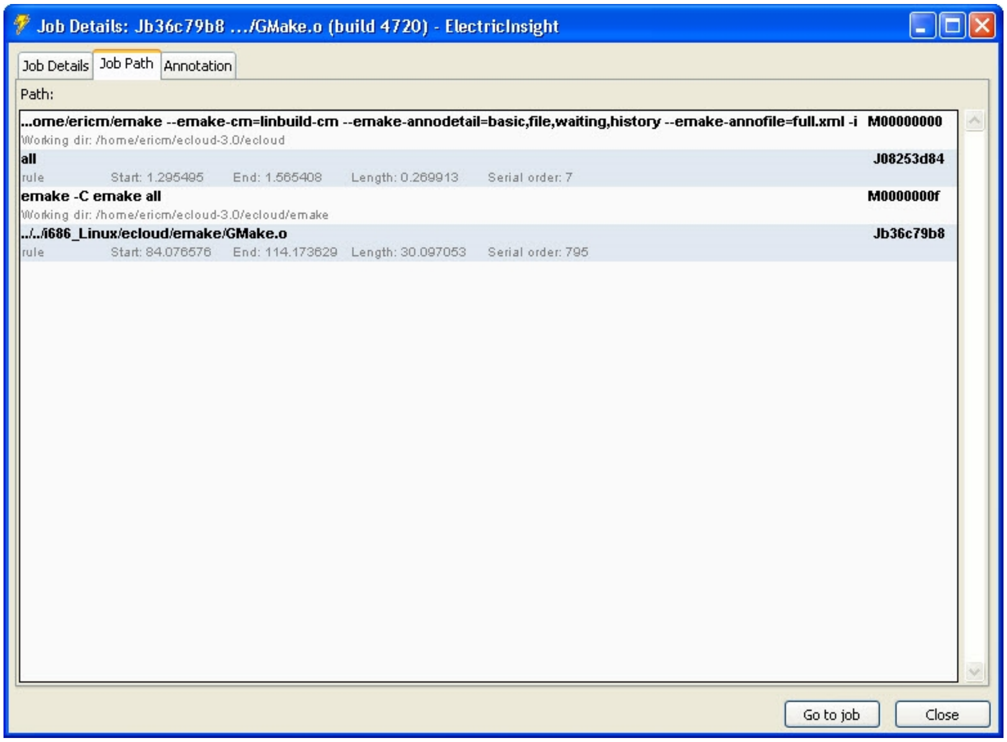
The **Makefile** field identifies the makefile and the relevant line in that makefile that created the job. You can click the makefile to open it in an editor. The **Makefile** field is empty for parse-type jobs because parse jobs are not created from a line in a makefile. For example, there is no rule in the makefile that instructs eMake to parse the makefile. To find which makefile a job parses, refer to the Directory and Name columns in the Job Path tab.

You can search the Output field by pressing CTRL-F. Pressing CTRL-G finds the next occurrence.

Job Path

This tab displays the path to the makefile line(s) that created the job.

The first line contains the command that invokes the top level makefile. Typically, this line invokes another makefile. If it does, the second line displays the job ID. If it does not invoke another makefile, the second line displays the name of the target the job executes.


Job Details: Jb36c79b8 .../GMake.o (build 4720) - ElectricInsight
Job Details
Job Path
Annotation
Path:
Go to job
Close

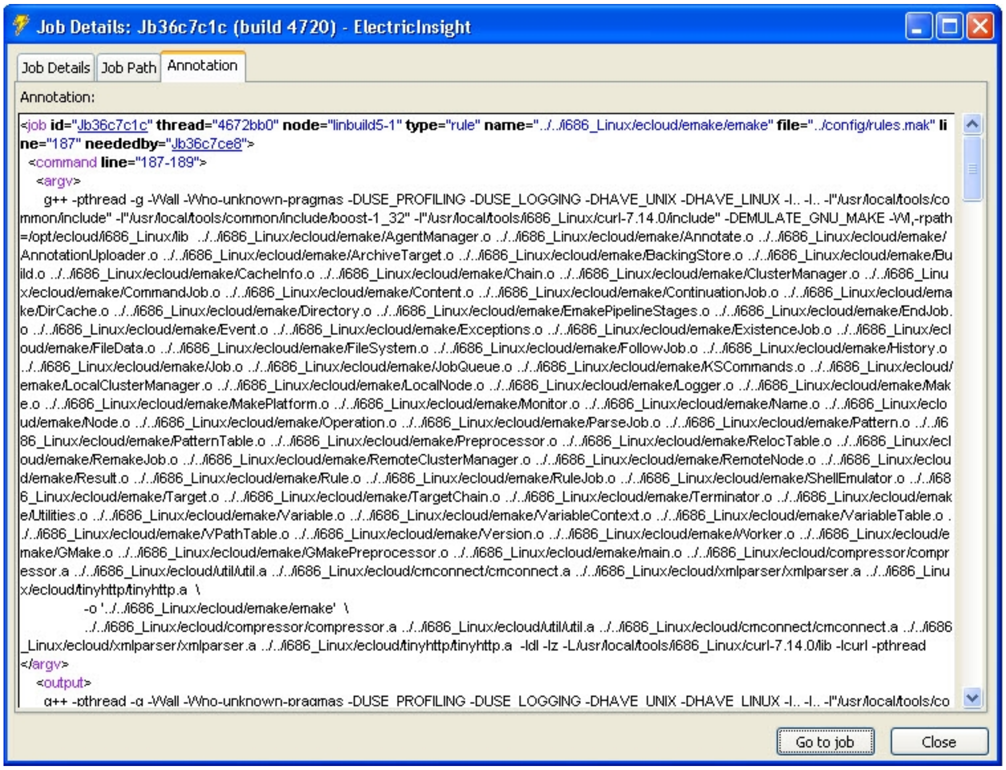
```
...ome/ericm/emake --emake-cm=linbuild-cm --emake-annodetail=basic,file,waiting,history --emake-annofile=full.xml -i  M00000000
Working dir: /home/ericm/ecloud-3.0/ecloud
all                                                                                                                J08253d84
rule          Start: 1.295495      End: 1.565408      Length: 0.269913      Serial order: 7
emake -C emake all                                                                                                 M0000000f
Working dir: /home/ericm/ecloud-3.0/ecloud/emake
../../i686_Linux/ecloud/emake/GMake.o                                                                              Jb36c79b8
rule          Start: 84.076576     End: 114.173629    Length: 30.097053     Serial order: 795
```

If the makefile line that created the job invoked a second makefile, this pattern of makefile lines and job IDs is repeated, chaining through makefiles until the target that created the job is reached.

Annotation

This tab displays the portion of the annotation file pertaining to the job.

Pressing CTRL-F searches for and highlight text; CTRL-G finds the next occurrence.



```
<job id="Jb36c7c1c" thread="4672bb0" node="linbuild5-1" type="rule" name="../../i686_Linux/ecloud/emake/emake" file="../config/rules.mak" li
ne="187" neededby="Jb36c7ce8">
 <command line="187-189">
  <argv>
   g++ -pthread -g -Wall -Wno-unknown-pragmas -DUSE_PROFILING -DUSE_LOGGING -DHAVE_UNIX -DHAVE_LINUX -I.. -I.. -I"/usr/local/tools/co
mmon/include" -I"/usr/local/tools/common/include/boost-1_32" -I"/usr/local/tools/i686_Linux/curl-7.14.0/include" -DEMULATE_GNU_MAKE -Wl,-rpath
=/opt/ecloud/i686_Linux/lib  ../../i686_Linux/ecloud/emake/AgentManager.o ../../i686_Linux/ecloud/emake/Annotate.o ../../i686_Linux/ecloud/emake/
AnnotationUploader.o ../../i686_Linux/ecloud/emake/ArchiveTarget.o ../../i686_Linux/ecloud/emake/BackingStore.o ../../i686_Linux/ecloud/emake/Bu
ild.o ../../i686_Linux/ecloud/emake/CacheInfo.o ../../i686_Linux/ecloud/emake/Chain.o ../../i686_Linux/ecloud/emake/ClusterManager.o ../../i686_Linu
x/ecloud/emake/CommandJob.o ../../i686_Linux/ecloud/emake/Content.o ../../i686_Linux/ecloud/emake/ContinuationJob.o ../../i686_Linux/ecloud/ema
ke/DirCache.o ../../i686_Linux/ecloud/emake/Directory.o ../../i686_Linux/ecloud/emake/EmakePipelineStages.o ../../i686_Linux/ecloud/emake/EndJob.
o ../../i686_Linux/ecloud/emake/Event.o ../../i686_Linux/ecloud/emake/Exceptions.o ../../i686_Linux/ecloud/emake/ExistenceJob.o ../../i686_Linux/ecl
oud/emake/FileData.o ../../i686_Linux/ecloud/emake/FileSystem.o ../../i686_Linux/ecloud/emake/FollowJob.o ../../i686_Linux/ecloud/emake/History.o
../../i686_Linux/ecloud/emake/Job.o ../../i686_Linux/ecloud/emake/JobQueue.o ../../i686_Linux/ecloud/emake/KSCommands.o ../../i686_Linux/ecloud/
emake/LocalClusterManager.o ../../i686_Linux/ecloud/emake/LocalNode.o ../../i686_Linux/ecloud/emake/Logger.o ../../i686_Linux/ecloud/emake/Mak
e.o ../../i686_Linux/ecloud/emake/MakePlatform.o ../../i686_Linux/ecloud/emake/Monitor.o ../../i686_Linux/ecloud/emake/Name.o ../../i686_Linux/eclo
ud/emake/Node.o ../../i686_Linux/ecloud/emake/Operation.o ../../i686_Linux/ecloud/emake/ParseJob.o ../../i686_Linux/ecloud/emake/Pattern.o ../../i6
86_Linux/ecloud/emake/PatternTable.o ../../i686_Linux/ecloud/emake/Preprocessor.o ../../i686_Linux/ecloud/emake/RelocTable.o ../../i686_Linux/ecl
oud/emake/RemakeJob.o ../../i686_Linux/ecloud/emake/RemoteClusterManager.o ../../i686_Linux/ecloud/emake/RemoteNode.o ../../i686_Linux/eclou
d/emake/Result.o ../../i686_Linux/ecloud/emake/Rule.o ../../i686_Linux/ecloud/emake/RuleJob.o ../../i686_Linux/ecloud/emake/ShellEmulator.o ../../i68
6_Linux/ecloud/emake/Target.o ../../i686_Linux/ecloud/emake/TargetChain.o ../../i686_Linux/ecloud/emake/Terminator.o ../../i686_Linux/ecloud/emak
e/Utilities.o ../../i686_Linux/ecloud/emake/Variable.o ../../i686_Linux/ecloud/emake/VariableContext.o ../../i686_Linux/ecloud/emake/VariableTable.o .
./../i686_Linux/ecloud/emake/VPathTable.o ../../i686_Linux/ecloud/emake/Version.o ../../i686_Linux/ecloud/emake/Worker.o ../../i686_Linux/ecloud/e
make/GMake.o ../../i686_Linux/ecloud/emake/GMakePreprocessor.o ../../i686_Linux/ecloud/emake/main.o ../../i686_Linux/ecloud/compressor/compr
essor.a ../../i686_Linux/ecloud/util/util.a ../../i686_Linux/ecloud/cmconnect/cmconnect.a ../../i686_Linux/ecloud/xmlparser/xmlparser.a ../../i686_Linu
x/ecloud/tinyhttp/tinyhttp.a  \
          -o '../../i686_Linux/ecloud/emake/emake'  \
          ../../i686_Linux/ecloud/compressor/compressor.a ../../i686_Linux/ecloud/util/util.a ../../i686_Linux/ecloud/cmconnect/cmconnect.a ../../i686
_Linux/ecloud/xmlparser/xmlparser.a ../../i686_Linux/ecloud/tinyhttp/tinyhttp.a  -ldl -lz -L/usr/local/tools/i686_Linux/curl-7.14.0/lib -lcurl -pthread
</argv>
  <output>
   g++ -pthread -g -Wall -Wno-unknown-pragmas -DUSE_PROFILING -DUSE_LOGGING -DHAVE_UNIX -DHAVE_LINUX -I.. -I.. -I"/usr/local/tools/co
```

Environment

This tab displays environment information that was used to run the job. The Environment tab is available only if your annotation file includes `env` level annotation.

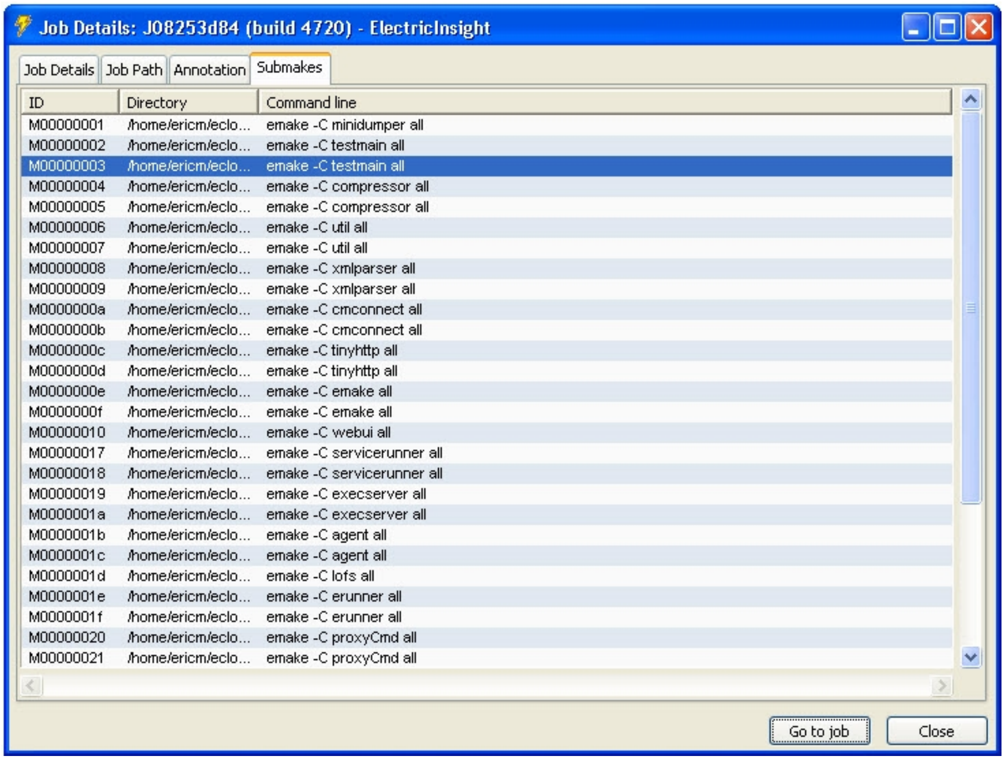
You can copy environment information to the clipboard:

1. Right-click anywhere in the window.
2. Click **Copy to clipboard**.
3. Click **…for bash/ksh** or **…for cmd**.

Submakes

This tab lists all submakes that the job runs, including their ID, Directory, and Command line make command. The Submakes tab is available only if the job includes submakes.

Double-click an entry in the list to display the job’s detail dialog.



ID	Directory	Command line
M00000001	/home/ericm/eclo...	emake -C minidumper all
M00000002	/home/ericm/eclo...	emake -C testmain all
M00000003	/home/ericm/eclo...	emake -C testmain all
M00000004	/home/ericm/eclo...	emake -C compressor all
M00000005	/home/ericm/eclo...	emake -C compressor all
M00000006	/home/ericm/eclo...	emake -C util all
M00000007	/home/ericm/eclo...	emake -C util all
M00000008	/home/ericm/eclo...	emake -C xmlparser all
M00000009	/home/ericm/eclo...	emake -C xmlparser all
M0000000a	/home/ericm/eclo...	emake -C cmconnect all
M0000000b	/home/ericm/eclo...	emake -C cmconnect all
M0000000c	/home/ericm/eclo...	emake -C tinyhttp all
M0000000d	/home/ericm/eclo...	emake -C tinyhttp all
M0000000e	/home/ericm/eclo...	emake -C emake all
M0000000f	/home/ericm/eclo...	emake -C emake all
M00000010	/home/ericm/eclo...	emake -C webui all
M00000017	/home/ericm/eclo...	emake -C servicerunner all
M00000018	/home/ericm/eclo...	emake -C servicerunner all
M00000019	/home/ericm/eclo...	emake -C execserver all
M0000001a	/home/ericm/eclo...	emake -C execserver all
M0000001b	/home/ericm/eclo...	emake -C agent all
M0000001c	/home/ericm/eclo...	emake -C agent all
M0000001d	/home/ericm/eclo...	emake -C lofs all
M0000001e	/home/ericm/eclo...	emake -C erunner all
M0000001f	/home/ericm/eclo...	emake -C erunner all
M00000020	/home/ericm/eclo...	emake -C proxyCmd all
M00000021	/home/ericm/eclo...	emake -C proxyCmd all

Make Details Dialog

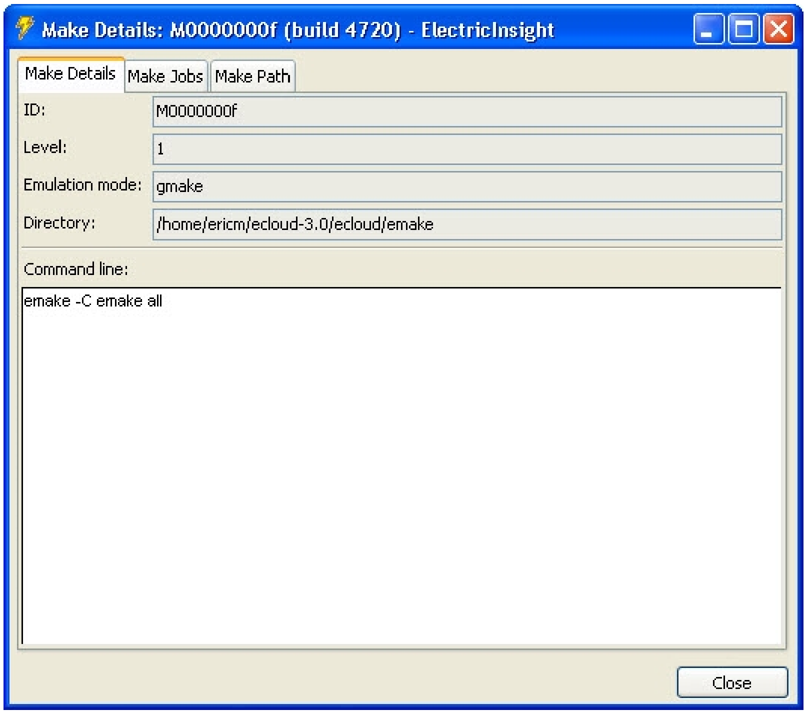
To open the Make Details dialog, double-click a make command, for example, in the Job Path tab of the Job Details dialog. Three tabs are available: Make Details, Make Jobs, and Make Path.

Topics:

- Make Details
- Make Jobs
- Make Path

Make Details

This tab displays information about make. The tab has the following fields: ID, Level, Emulation mode, Directory, and Command line.



ID: M0000000f

Level: 1

Emulation mode: gmake

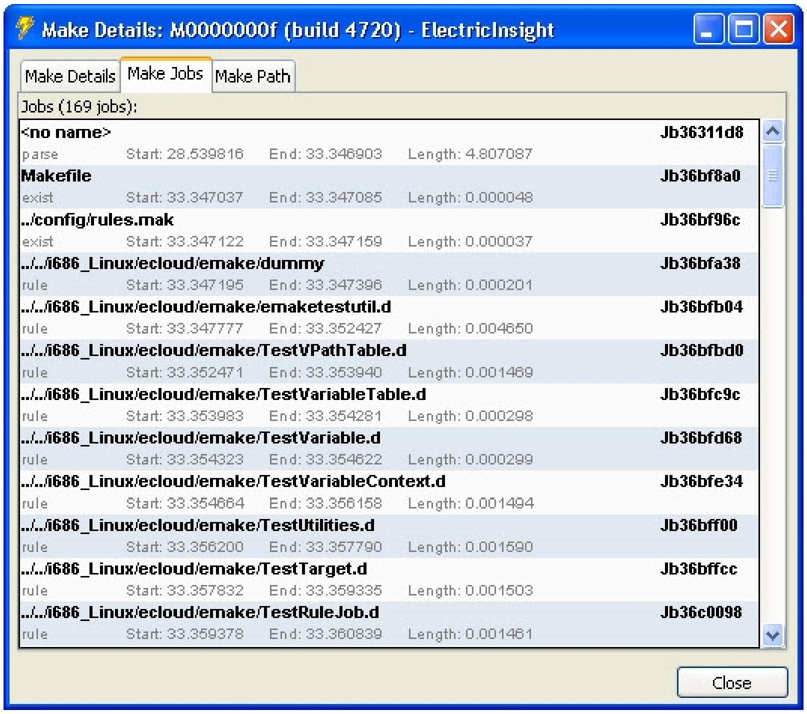
Directory: /home/ericm/ecloud-3.0/ecloud/emake

Command line:

```
emake -C emake all
```

Make Jobs

This tab lists all jobs that the makefile runs.



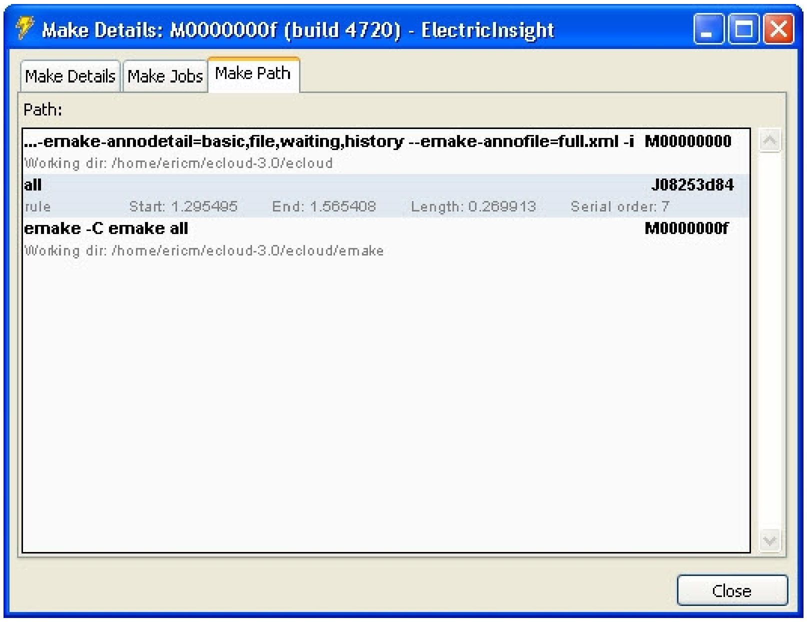
Jobs (169 jobs):

```
<no name>                                                        Jb36311d8
parse      Start: 28.539816   End: 33.346903   Length: 4.807087
Makefile                                                         Jb36bf8a0
exist      Start: 33.347037   End: 33.347085   Length: 0.000048
../config/rules.mak                                              Jb36bf96c
exist      Start: 33.347122   End: 33.347159   Length: 0.000037
../../i686_Linux/ecloud/emake/dummy                              Jb36bfa38
rule       Start: 33.347195   End: 33.347396   Length: 0.000201
../../i686_Linux/ecloud/emake/emaketestutil.d                    Jb36bfb04
rule       Start: 33.347777   End: 33.352427   Length: 0.004650
../../i686_Linux/ecloud/emake/TestVPathTable.d                   Jb36bfbd0
rule       Start: 33.352471   End: 33.353940   Length: 0.001469
../../i686_Linux/ecloud/emake/TestVariableTable.d                Jb36bfc9c
rule       Start: 33.353983   End: 33.354281   Length: 0.000298
../../i686_Linux/ecloud/emake/TestVariable.d                     Jb36bfd68
rule       Start: 33.354323   End: 33.354622   Length: 0.000299
../../i686_Linux/ecloud/emake/TestVariableContext.d              Jb36bfe34
rule       Start: 33.354664   End: 33.356158   Length: 0.001494
../../i686_Linux/ecloud/emake/TestUtilities.d                    Jb36bff00
rule       Start: 33.356200   End: 33.357790   Length: 0.001590
../../i686_Linux/ecloud/emake/TestTarget.d                       Jb36bffcc
rule       Start: 33.357832   End: 33.359335   Length: 0.001503
../../i686_Linux/ecloud/emake/TestRuleJob.d                      Jb36c0098
rule       Start: 33.359378   End: 33.360839   Length: 0.001461
```

Make Path

This tab displays the path to the makefile line(s) that created the make.

The first line contains the command that invokes the top level makefile. Typically, this line invokes another makefile. If it does, the second line displays the job ID. If it does not invoke another makefile, the second line displays the name of the target the job executes.

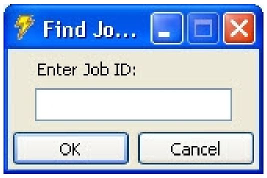


```
...-emake-annodetail=basic,file,waiting,history --emake-annofile=full.xml -i  M00000000
Working dir: /home/ericm/ecloud-3.0/ecloud
all                                                                       J08253d84
rule          Start: 1.295495      End: 1.565408      Length: 0.269913      Serial order: 7
emake -C emake all                                                        M0000000f
Working dir: /home/ericm/ecloud-3.0/ecloud/emake
```

Searching for Jobs

If you ***know*** the job ID you want to find:

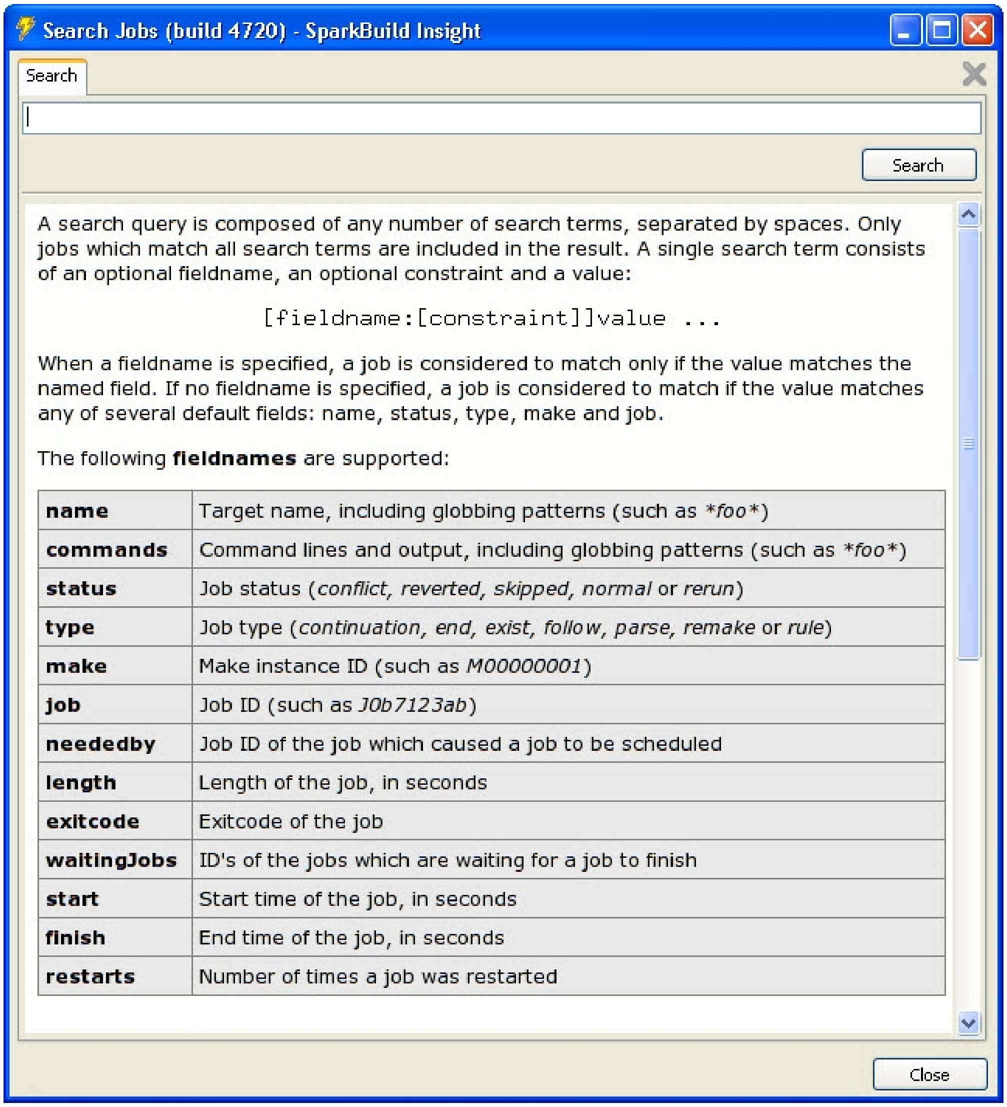
1. Click **Tools > Find job by ID…**.



2. Enter the job ID.
3. Click **OK**.

If you ***do not know*** the job ID, you can search for jobs using several criteria. Follow these steps:

1. Click **Tools > Search jobs…**.



A search query is composed of any number of search terms, separated by spaces. Only jobs which match all search terms are included in the result. A single search term consists of an optional fieldname, an optional constraint and a value:

```
[fieldname:[constraint]]value ...
```

When a fieldname is specified, a job is considered to match only if the value matches the named field. If no fieldname is specified, a job is considered to match if the value matches any of several default fields: name, status, type, make and job.

The following **fieldnames** are supported:

name	Target name, including globbing patterns (such as **foo**)
commands	Command lines and output, including globbing patterns (such as **foo**)
status	Job status (*conflict, reverted, skipped, normal* or *rerun*)
type	Job type (*continuation, end, exist, follow, parse, remake* or *rule*)
make	Make instance ID (such as *M00000001*)
job	Job ID (such as *J0b7123ab*)
neededby	Job ID of the job which caused a job to be scheduled
length	Length of the job, in seconds
exitcode	Exitcode of the job
waitingJobs	ID's of the jobs which are waiting for a job to finish
start	Start time of the job, in seconds
finish	End time of the job, in seconds
restarts	Number of times a job was restarted

2. Type in the strings you want to find (example, “gcc conflict”). You can also constrain your search by searching for specific values in specific fields.
3. Click **Search**.

   The values come from fields in the build annotation file.
4. You can sort your results by choosing an option from the drop-down.
5. Double-click a row to display a job’s Job Details dialog.
6. In the dialog’s lower-right corner, click **Go to job**.

   The job is now highlighted in the Agents & Jobs section.

Search Examples

To find rule jobs involved in conflicts:

```
rule conflict
```

or

```
type:rule status:conflict
```

To find conflicts involving jobs named “export”:

```
conflict name:export
```

To find rerun jobs with names including “port”:

```
rerun name:*port*
```

To find non-reverted jobs with names ending in “Makefile”:

```
!reverted name:*Makefile
```

To find jobs for the parse type of job caching:

```
jobcache:parse
```

(For the JobCache add-on) To find jobs for the gcc type of job caching:

```
jobcache:gcc
```

To find jobs that had a job cache hit:

```
jobcache:hit
```

To combine searches to find jobs that used the gcc job cache type *and* had a job cache miss:

```
jobcache:gcc jobcache:miss
```

The JobCache add-on lets a build avoid recompiling object files that it previously built, if their inputs have not changed. For more information about the add-on, see the *ElectricAccelerator Electric Make User Guide* at http://docs.electric-cloud.com/accelerator_doc/AcceleratorIndex.html.

Supported Search Fields

The following fields are supported:

- name—the name of a job is the name of the target that created it. Continuation and parser jobs do not have names.
- commands—search makefile command lines and output. Globbing patterns are supported (for example, commands:gcc *foo*).
- status—possible values include:
  - normal—a successfully run job
  - conflict—a job involved in a conflict. Many of these jobs or jobs on which they rely are reverted. Typically, conflicts occur when there is no history file—when the environment does not provide Electric Make with enough information to predict all job dependencies.
  - rerun—some reverted jobs must be run with correct file system context. Rerun jobs are always serialized to guarantee correct context.
  - reverted—a job is reverted when Electric Make serializes job results and the job in question was executed before a (logically) preceding job that failed. The preceding job might have failed because of a conflict. Reverting a job restores the Agent’s file system to the state before the job was run.
  - skipped—skipped jobs are the same as reverted jobs except the skipped job did not run before the preceding job failed.
- type: possible values include:

  - continuation—a continuation job occurs when commands associated with a makefile target include a submake (also known as, *recursive* make). Typically, these submake commands begin with `$(MAKE)`. The commands below a submake, down to and including the next submake command (for the same target), comprise a continuation job. For a target, there are as many continuation jobs as there are submakes with commands after them.

    ```
    all:
      echo abc
      $(MAKE) abc
      echo def
      echo ghi
      $(MAKE) ghi
      echo jkl
      $(MAKE) jkl
    ```

    In this example, the first continuation job starts with `echo def` and ends with `$(MAKE) ghi`. The second and last continuation job consists of two commands involving `jkl`.

  - end—clean-up tasks that occur at the end of makefiles (for example, removing intermediate files).
  - exist—jobs corresponding to top-level targets without rules. Electric Make verifies the target exists.
  - follow—one of these jobs follows every submake to handle submake output and exit status for the parent.
  - parse—jobs that parse makefiles.
  - remake—jobs that remake makefiles (only for GNU make).
  - rule—lists jobs corresponding to makefile rules. Sorting results by name lets you find the job that executed a specific rule. If you have several rules (in different makefiles) with the same name, you can display several Job Details dialogs at once.

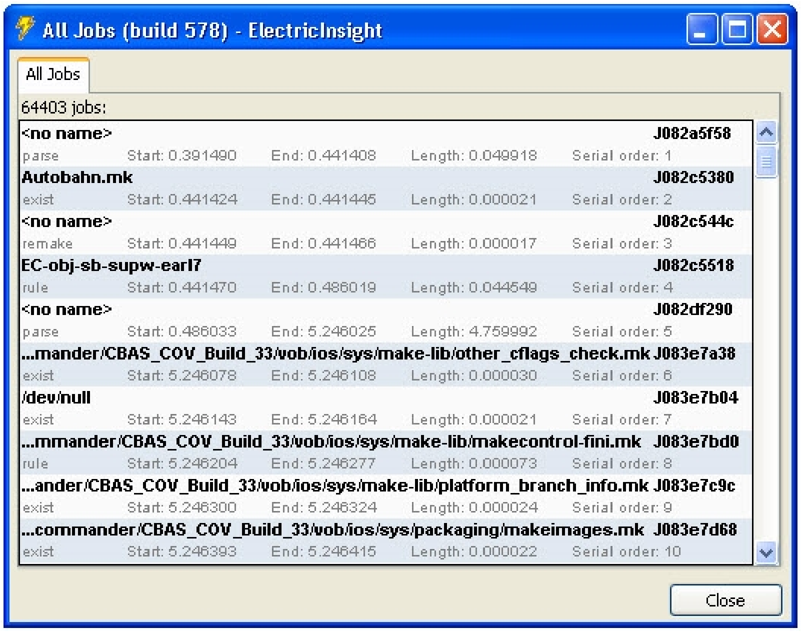
- make—this is the ID of the makefile that created the job. The top-level makefile has ID 0 (zero). For example, `make:0` returns a list of jobs created directly from the top-level makefile.
- job—the job ID, for example, `job:J01b11200`
- neededby—the job in question is needed by another job. Specify the job ID of the job that needs the job in question, for example, `neededby:J01b11200`. This is useful for tracking inter-job dependencies.
- length—job length, in seconds. If you specify `length:12.34`, you find jobs that lasted exactly 12.34 seconds only. You can also search using `length:>12.34`, `length:>=12.34`, `length:==12.34`, `length:<12.34`, and `length:<=12.34`. To find jobs that lasted 10-15 seconds, specify `length:>=10.0 length:<=15.0`.
- exitcode—the job exit code. To find jobs with a non-zero exit code use `exitcode:!=0`. In general, use arithmetic relations (=, >, <, >=, <=) to search numerical values and use no relation symbols when searching for string values.
- waitingJobs:*<jobID>*—to find prerequisite jobs of <jobID>.
- start—job start time (in seconds) after the build start time. To find other jobs that started in the first minute of the job, use `start:<60`. To find jobs that took more than 60 seconds and started during the 10th minute into the build, type `length:>60 start:>=600 start:<660`.
- finish—the job end time (in seconds) after the build start time.
- restarts—lists jobs restarted a specified number of times. Jobs are restarted because of cluster sharing and agent failure.
- jobcache—lists jobs by job cache type and status. Possible values include

- hit—jobs for which eMake had a cache hit.
- miss—jobs for which eMake had a cache miss.
- newslot—jobs for which eMake created a new cache slot. The object files that were previously cached were from compilations that used different command-line arguments, environment variables, or working directories, or any combination of these.
- uncacheable—jobs for which something went wrong with updating the relevant cache slot.
- rootschanged—jobs for which there is no natural mapping from the old eMake roots to the new eMake roots.
- parse—jobs with job cache type `parse`.
- gcc—jobs with job cache type `gcc`.

Note: You can restrict the search scope for the following fields only: type, status, name, make, and job.

Viewing All Jobs

If you want to list all of the build's jobs, click **Tools > View all jobs…**.



```
64403 jobs:
<no name>                                                        J082a5f58
parse      Start: 0.391490   End: 0.441408   Length: 0.049918   Serial order: 1
Autobahn.mk                                                      J082c5380
exist      Start: 0.441424   End: 0.441445   Length: 0.000021   Serial order: 2
<no name>                                                        J082c544c
remake     Start: 0.441449   End: 0.441466   Length: 0.000017   Serial order: 3
EC-obj-sb-supw-earl7                                             J082c5518
rule       Start: 0.441470   End: 0.486019   Length: 0.044549   Serial order: 4
<no name>                                                        J082df290
parse      Start: 0.486033   End: 5.246025   Length: 4.759992   Serial order: 5
...mander/CBAS_COV_Build_33/vob/ios/sys/make-lib/other_cflags_check.mk J083e7a38
exist      Start: 5.246078   End: 5.246108   Length: 0.000030   Serial order: 6
/dev/null                                                        J083e7b04
exist      Start: 5.246143   End: 5.246164   Length: 0.000021   Serial order: 7
...mmander/CBAS_COV_Build_33/vob/ios/sys/make-lib/makecontrol-fini.mk  J083e7bd0
rule       Start: 5.246204   End: 5.246277   Length: 0.000073   Serial order: 8
...ander/CBAS_COV_Build_33/vob/ios/sys/make-lib/platform_branch_info.mk J083e7c9c
exist      Start: 5.246300   End: 5.246324   Length: 0.000024   Serial order: 9
...commander/CBAS_COV_Build_33/vob/ios/sys/packaging/makeimages.mk  J083e7d68
exist      Start: 5.246393   End: 5.246415   Length: 0.000022   Serial order: 10
```

Using the Hyperlog

The hyperlog presents an augmented version of the familiar build output log, leveraging the extra information provided by Electric Make to enable highlighting errors and warnings, folding of submake output, and line numbering.

To access the hyperlog, click the **Go to log** button on any Job Details dialog. When the hyperlog opens, you go directly to the corresponding job's location. You can also click **Tools > View build log…** to access the hyperlog.

Double-clicking any line of log output displays the corresponding job's details dialog.

Note: The hyperlog is not available when monitoring a live build because the log file doesn't exist.

Chapter 5: Reports

You can generate reports from the GUI and from the command line.

In the GUI, to display the Reports dialog, click **Tools > Reports** or click the **View build reports** icon on the toolbar. You can open one report at a time for each Insight instance, but you can open several Insight instances. You can run most reports by double-clicking the report name. Some reports have mandatory parameters that you must provide before running them.

Some reports listed here can be generated through both methods, and some reports can be generated through one method only.

With ElectricInsight's command-line reporting capability, the product can be integrated to generate batch trend reports as part of a build automation system. For a list of all command-line-enabled reports and their associated commands, see Command-Line-Enabled Reports.

Topics:

Build Manifest

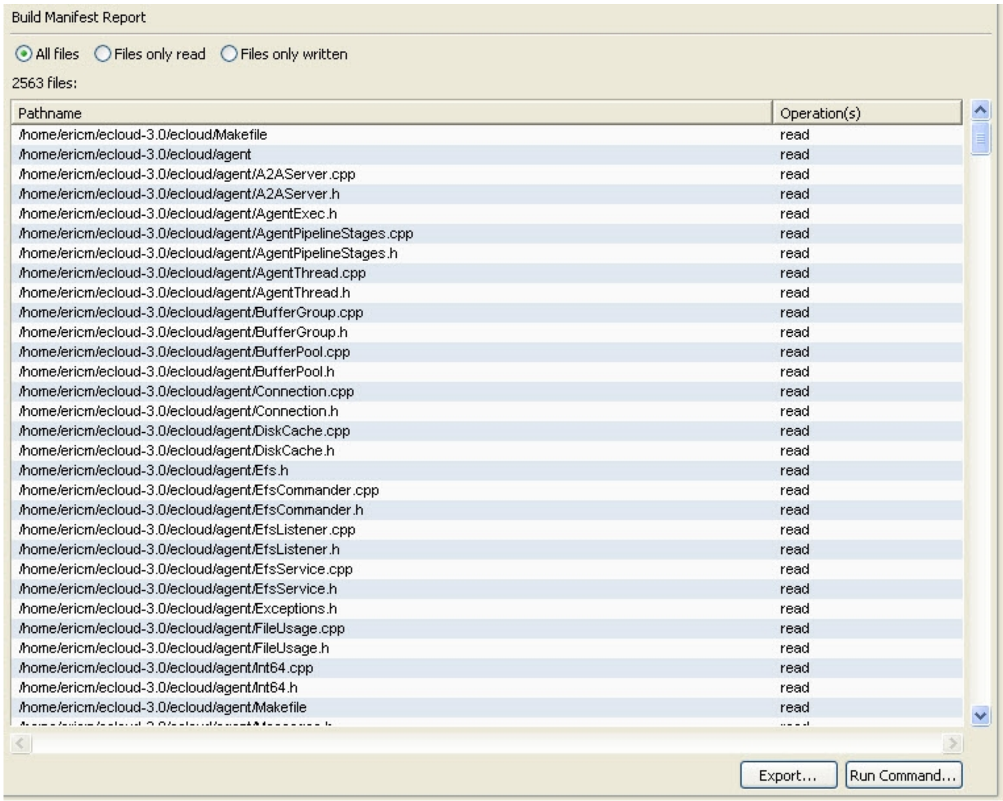
The Build Manifest report lists all files that were read and/or written by the build.

This report can help to:

- Verify that all files read by the build are the latest version from source control
- Add all files written by the build to an archive
- Produce a list of all inputs and a list of all outputs
- Identify files that are written but never read

This report requires

```
--emake-annodetail=basic,file
```


Build Manifest Report

● All files ○ Files only read ○ Files only written

2563 files:

Pathname	Operation(s)
/home/ericm/ecloud-3.0/ecloud/Makefile	read
/home/ericm/ecloud-3.0/ecloud/agent	read
/home/ericm/ecloud-3.0/ecloud/agent/A2AServer.cpp	read
/home/ericm/ecloud-3.0/ecloud/agent/A2AServer.h	read
/home/ericm/ecloud-3.0/ecloud/agent/AgentExec.h	read
/home/ericm/ecloud-3.0/ecloud/agent/AgentPipelineStages.cpp	read
/home/ericm/ecloud-3.0/ecloud/agent/AgentPipelineStages.h	read
/home/ericm/ecloud-3.0/ecloud/agent/AgentThread.cpp	read
/home/ericm/ecloud-3.0/ecloud/agent/AgentThread.h	read
/home/ericm/ecloud-3.0/ecloud/agent/BufferGroup.cpp	read
/home/ericm/ecloud-3.0/ecloud/agent/BufferGroup.h	read
/home/ericm/ecloud-3.0/ecloud/agent/BufferPool.cpp	read
/home/ericm/ecloud-3.0/ecloud/agent/BufferPool.h	read
/home/ericm/ecloud-3.0/ecloud/agent/Connection.cpp	read
/home/ericm/ecloud-3.0/ecloud/agent/Connection.h	read
/home/ericm/ecloud-3.0/ecloud/agent/DiskCache.cpp	read
/home/ericm/ecloud-3.0/ecloud/agent/DiskCache.h	read
/home/ericm/ecloud-3.0/ecloud/agent/Efs.h	read
/home/ericm/ecloud-3.0/ecloud/agent/EfsCommander.cpp	read
/home/ericm/ecloud-3.0/ecloud/agent/EfsCommander.h	read
/home/ericm/ecloud-3.0/ecloud/agent/EfsListener.cpp	read
/home/ericm/ecloud-3.0/ecloud/agent/EfsListener.h	read
/home/ericm/ecloud-3.0/ecloud/agent/EfsService.cpp	read
/home/ericm/ecloud-3.0/ecloud/agent/EfsService.h	read
/home/ericm/ecloud-3.0/ecloud/agent/Exceptions.h	read
/home/ericm/ecloud-3.0/ecloud/agent/FileUsage.cpp	read
/home/ericm/ecloud-3.0/ecloud/agent/FileUsage.h	read
/home/ericm/ecloud-3.0/ecloud/agent/Int64.cpp	read
/home/ericm/ecloud-3.0/ecloud/agent/Int64.h	read
/home/ericm/ecloud-3.0/ecloud/agent/Makefile	read
/home/ericm/ecloud-3.0/ecloud/agent/Messages.h	read

To export the report to a .csv file, click **Export**.

To run a shell command:

1. Click **Run Command**.
2. Type a command and a pathname in the Run Shell Command dialog.

   The shell command is run with each file in the filtered list as an argument. Results and errors are displayed under their corresponding tabs.

Build Metrics

This command-line-only report prints the content of the <metrics> element in the annotation.

Run:

```
einsight --report=BuildMetrics <annotation>
```

Note: On Windows, use `einsight-cmd` instead of `einsight` when running reports from the command line.

Where <*annotation*> is the path of the annotation file.

Results format:

```
name,value
```

Build Summary

This command-line-only report prints the information from the left-side of the main Insight display.

Run:

```
einsight --report=BuildSummary <annotation>
```

Note: On Windows, use `einsight-cmd` instead of `einsight` when running reports from the command line.

Where <*annotation*> is the path of the annotation file.

Results format:

```
name,value
```

Cluster Sharing

The Cluster Sharing report shows the total work time lost (sum of aborted job run-times) and net build time lost (total work time minus overlaps) due to cluster sharing over the lifetime of the build.

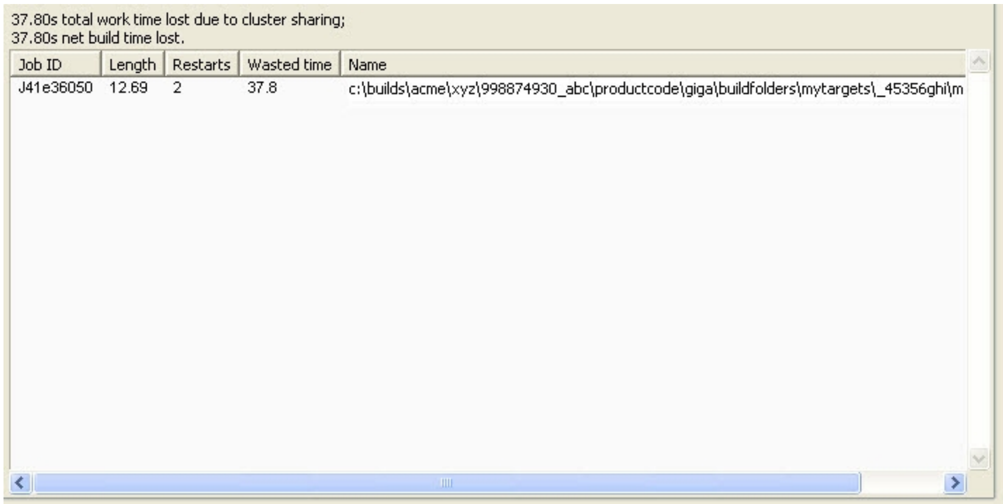
This report is useful if your build aborts jobs because an agent host is de-allocated from the build. The host might have been allocated to another build, moved to another cluster, or simply shut down. The aborted jobs must be rerun, resulting in a slower build.

This report does not consider the effect of not using some agent hosts because other builds are using them. To evaluate this factor, use the ElectricSimulator report.

Example

- Job A starts 5 seconds into the build.
- Job B starts 10 seconds into the build.
- Both jobs are terminated 20 seconds into the build.

The total work time lost is 15 + 10 seconds, but the net build time lost is only 15 seconds because the build time lost by B overlaps the build time lost by A.



37.80s total work time lost due to cluster sharing;
37.80s net build time lost.

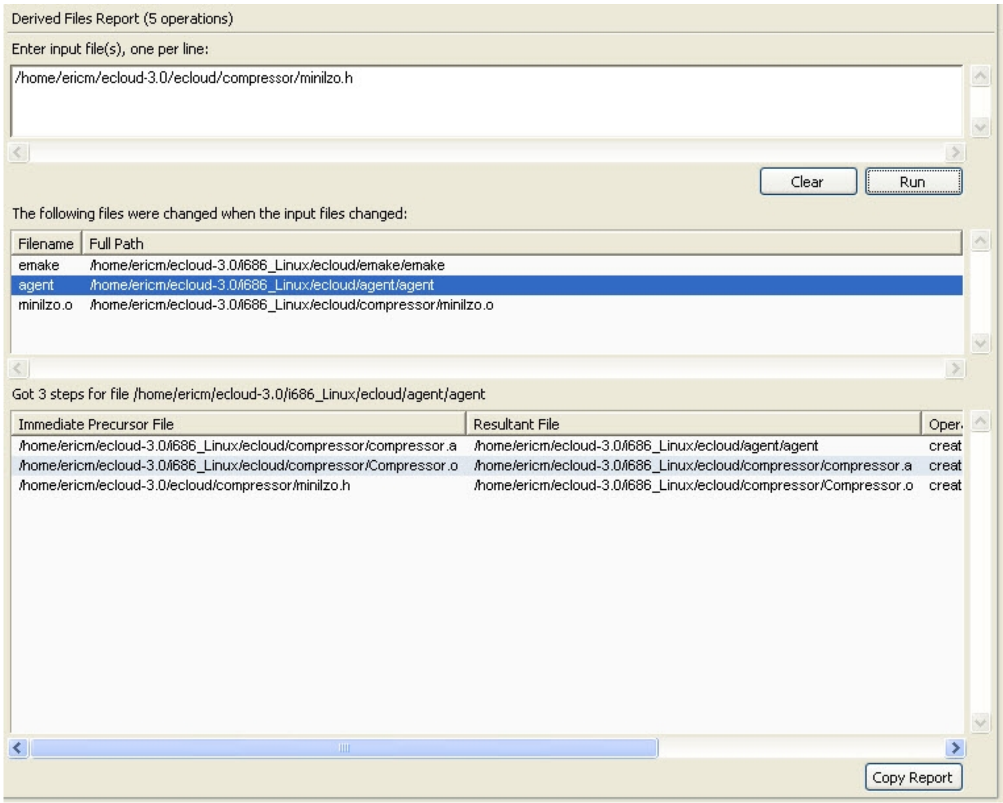
Job ID	Length	Restarts	Wasted time	Name
J41e36050	12.69	2	37.8	c:\builds\acme\xyz\998874930_abc\productcode\giga\buildfolders\mytargets_45356ghi\m

Derived Files Analysis

The Derived Files Analysis report lets you view a specified file's affected outputs. For example, this report answers the question, "If I change file X, what outputs will be affected?" A file is likely to be affected if it is produced concurrently with, or subsequent to, reads of the specified file.

This report requires

```
--emake-annodetail=file,history
```



Derived Files Report (5 operations)

Enter input file(s), one per line:

/home/ericm/ecloud-3.0/ecloud/compressor/minilzo.h

The following files were changed when the input files changed:

Filename	Full Path
emake	/home/ericm/ecloud-3.0/i686_Linux/ecloud/emake/emake
agent	/home/ericm/ecloud-3.0/i686_Linux/ecloud/agent/agent
minilzo.o	/home/ericm/ecloud-3.0/i686_Linux/ecloud/compressor/minilzo.o

Got 3 steps for file /home/ericm/ecloud-3.0/i686_Linux/ecloud/agent/agent

Immediate Precursor File	Resultant File	Oper.
/home/ericm/ecloud-3.0/i686_Linux/ecloud/compressor/compressor.a	/home/ericm/ecloud-3.0/i686_Linux/ecloud/agent/agent	creat
/home/ericm/ecloud-3.0/i686_Linux/ecloud/compressor/Compressor.o	/home/ericm/ecloud-3.0/i686_Linux/ecloud/compressor/compressor.a	creat
/home/ericm/ecloud-3.0/ecloud/compressor/minilzo.h	/home/ericm/ecloud-3.0/i686_Linux/ecloud/compressor/Compressor.o	creat

To run the report:

1. Enter the file(s) with full paths in the input field.
2. Click **Run**.

The middle display area lists files that will change when the input file changes. Double-click a file in this list to display its chain of operations in the lower display area.

Double-click an item in the lower display area to display its Job Details dialog.

To clear all fields, click **Clear**.

To copy the displayed chain of operations (in the lower display area) to the clipboard, click **Copy Report**.

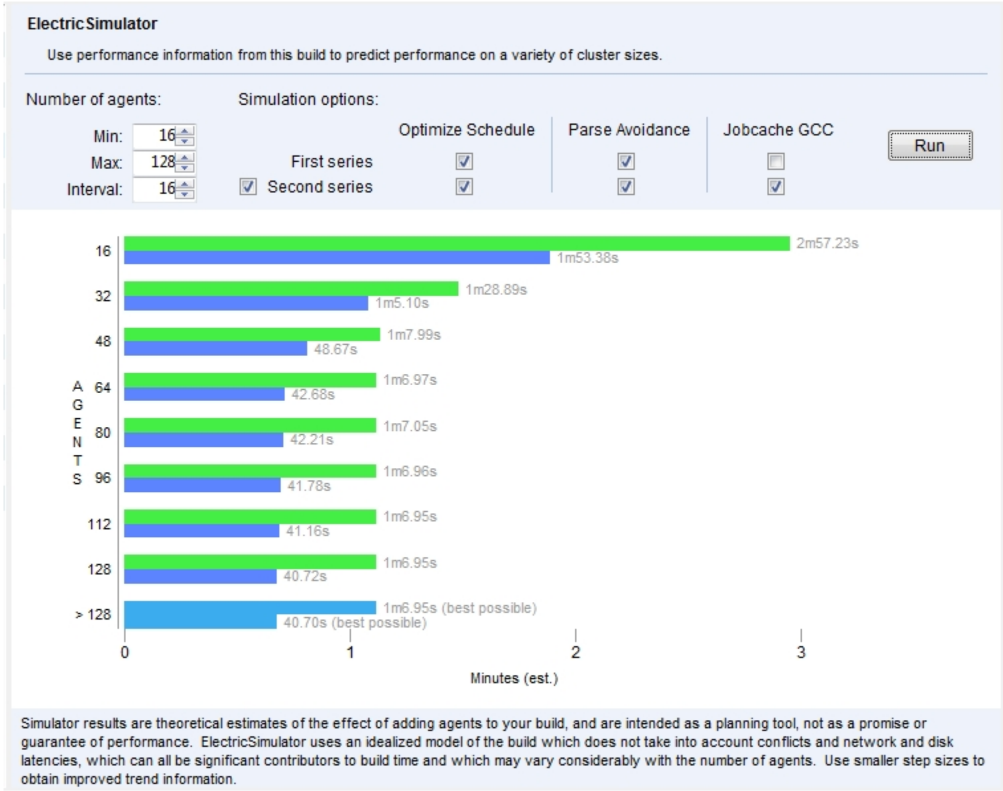
ElectricSimulator

The ElectricSimulator report predicts build performance on a variety of cluster sizes. The report lets you simulate build performance with or without gcc job caching enabled, with or without schedule optimization enabled, and with or without parse avoidance enabled. The bottom bar in the bar chart is the best possible estimated build time.

The report includes a **Second series** checkbox, which lets you set two combinations of the above simulation options at the same time, so you can compare two predicted performances on one chart.

The report requires

```
--emake-annodetail=waiting
```



ElectricSimulator

Use performance information from this build to predict performance on a variety of cluster sizes.

Number of agents:

Min: 16
Max: 128
Interval: 16

Simulation options:

	Optimize Schedule	Parse Avoidance	Jobcache GCC
First series	☑	☑	□
☑ Second series	☑	☑	☑

Simulator results are theoretical estimates of the effect of adding agents to your build, and are intended as a planning tool, not as a promise or guarantee of performance. ElectricSimulator uses an idealized model of the build which does not take into account conflicts and network and disk latencies, which can all be significant contributors to build time and which may vary considerably with the number of agents. Use smaller step sizes to obtain improved trend information.

Note: The simulator time results are estimates only. They do not indicate guaranteed performance.

Setting the Parameters for Numbers of Agents

There are three parameters for number of agents:

- **Min**—The minimum number of agents to simulate. The default is 16.
- **Max**—The maximum number of agents to simulate. The default is 128.
- **Interval**—The number of agents to increment the count each time. The default is 16.

Setting the Simulation Options

- **Optimize Schedule**—Enables or disables simulation with the schedule optimization feature, which uses performance and dependency information from previous builds to optimize the runtime ordering of jobs in subsequent builds. Schedule optimization is enabled by default in builds, but even if you have disabled schedule optimization, the report can accurately predict schedule optimization results.
- **Parse Avoidance**—Enable or disables simulation with the parse avoidance feature, which caches and reuses parse result files to speed up both full and incremental builds by minimizing makefile parse time. Parse avoidance is not enabled by default in builds; with parse avoidance disabled, the report attempts to simulate the effects of parse avoidance, but it must make assumptions that make the report results less accurate.
- **Jobcache GCC**—Enables or disables simulation with the JobCache add-on. The add-on lets a build avoid recompiling object files that it previously built, if their inputs have not changed. For more information about the add-on, see the *ElectricAccelerator Electric Make User Guide* at http://docs.electric-cloud.com/accelerator_doc/AcceleratorIndex.html.

  The annotation file includes the duration of each job that populated the cache, so that the simulator can accurately simulate a build *without* JobCache by using only annotation from a build *with* JobCache.

  Job cache annotation is included in basic annotation. Even without job cache enabled in the build, the report attempts to simulate the effects of JobCache, but it must make assumptions that make the report results less accurate.

  For more information about basic annotation and other annotation levels, see Chapter 8, *Annotation*, in the *ElectricAccelerator Electric Make User Guide* at http://docs.electric-cloud.com/accelerator_doc/AcceleratorIndex.html.

Command-Line Interface Report

Run this command:

```
einsight --report="ElectricSimulator [minagents] [maxagents] [interval]" <annotation>
```

Note: On Windows, use `einsight-cmd` instead of `einsight` when running reports from the command line.

Where:

[minagents] (optional) is the minimum number of agents to simulate. The default is 4.

[maxagents] (optional) is the maximum number of agents to simulate. The default is 32.

[interval] (optional) is the number of agents to increment the count each time. The default is 4.

<annotation> is the path of the annotation file.

Results format:

```
agents,duration
```

Export Timeline

This command-line-only report exports the main build timeline view to a graphic.

Run:

```
einsight --report=ExportTimeline <filename> [mode] [scale] [rowheight] [rowspace] <
annotation>
```

Note: On Windows, use `einsight-cmd` instead of `einsight` when running reports from the command line.

Where:

- *<filename>* is the name of the PNG file to write to. (Required)
- *[mode]* indicates if you want the image in color or black and white. Valid values are `color` or `bw`. Default is `color`. (Optional)
- *[scale]* is the number of seconds per pixel. Default is `2`. (Optional)
- *[rowheight]* is the number of pixels per row of jobs. Default is `1`. (Optional)
- *[rowspace]* is the number of pixels separating rows. Default is `1`. (Optional)
- *<annotation>* is the path of the annotation file. (Required)

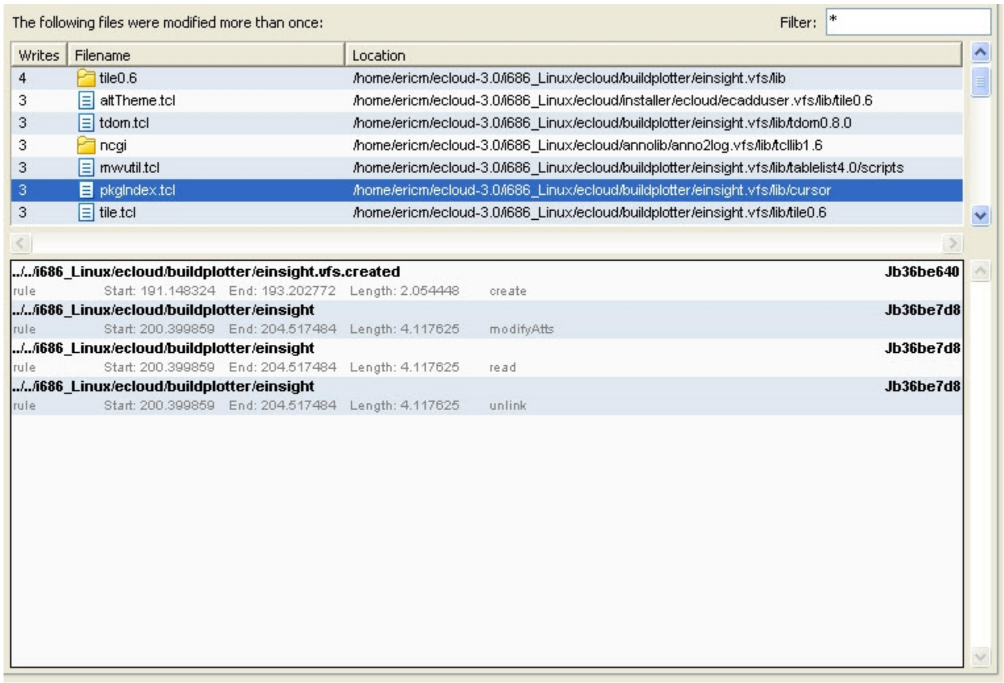
Files Modified Multiple Times

This report shows which files are modified (created, updated, or deleted) multiple times during a build.

Jobs involving these files require careful serialization to ensure the file operations sequence is performed in the correct order.

This report requires

```
--emake-annodetail=file
```



The following files were modified more than once:

Filter: *

Writes	Filename	Location
4	tile0.6	/home/ericm/ecloud-3.0/i686_Linux/ecloud/buildplotter/einsight.vfs/lib
3	altTheme.tcl	/home/ericm/ecloud-3.0/i686_Linux/ecloud/installer/ecloud/ecadduser.vfs/lib/tile0.6
3	tdom.tcl	/home/ericm/ecloud-3.0/i686_Linux/ecloud/buildplotter/einsight.vfs/lib/tdom0.8.0
3	ncgi	/home/ericm/ecloud-3.0/i686_Linux/ecloud/annolib/anno2log.vfs/lib/tcllib1.6
3	mwutil.tcl	/home/ericm/ecloud-3.0/i686_Linux/ecloud/buildplotter/einsight.vfs/lib/tablelist4.0/scripts
3	pkgIndex.tcl	/home/ericm/ecloud-3.0/i686_Linux/ecloud/buildplotter/einsight.vfs/lib/cursor
3	tile.tcl	/home/ericm/ecloud-3.0/i686_Linux/ecloud/buildplotter/einsight.vfs/lib/tile0.6

```
../../i686_Linux/ecloud/buildplotter/einsight.vfs.created                     Jb36be640
rule      Start: 191.148324   End: 193.202772   Length: 2.054448    create
../../i686_Linux/ecloud/buildplotter/einsight                                 Jb36be7d8
rule      Start: 200.399859   End: 204.517484   Length: 4.117625    modifyAtts
../../i686_Linux/ecloud/buildplotter/einsight                                 Jb36be7d8
rule      Start: 200.399859   End: 204.517484   Length: 4.117625    read
../../i686_Linux/ecloud/buildplotter/einsight                                 Jb36be7d8
rule      Start: 200.399859   End: 204.517484   Length: 4.117625    unlink
```

After you run the report, filter the results:

1. Type the string you want to filter for in the **Filter** field.

   Use an asterisk to match any number of characters, and use a question mark to match any single character.

   You can also use simple regular expressions, for example, *[xz].o and *[x-z].o

   Filters are case sensitive.

2. Press **Enter**.

Job Cache Misses

The Job Cache Misses report provides a summary of the causes of cache misses for any job using job caching: jobs using gcc (for the JobCache add-on) or jobs using the parse avoidance feature.

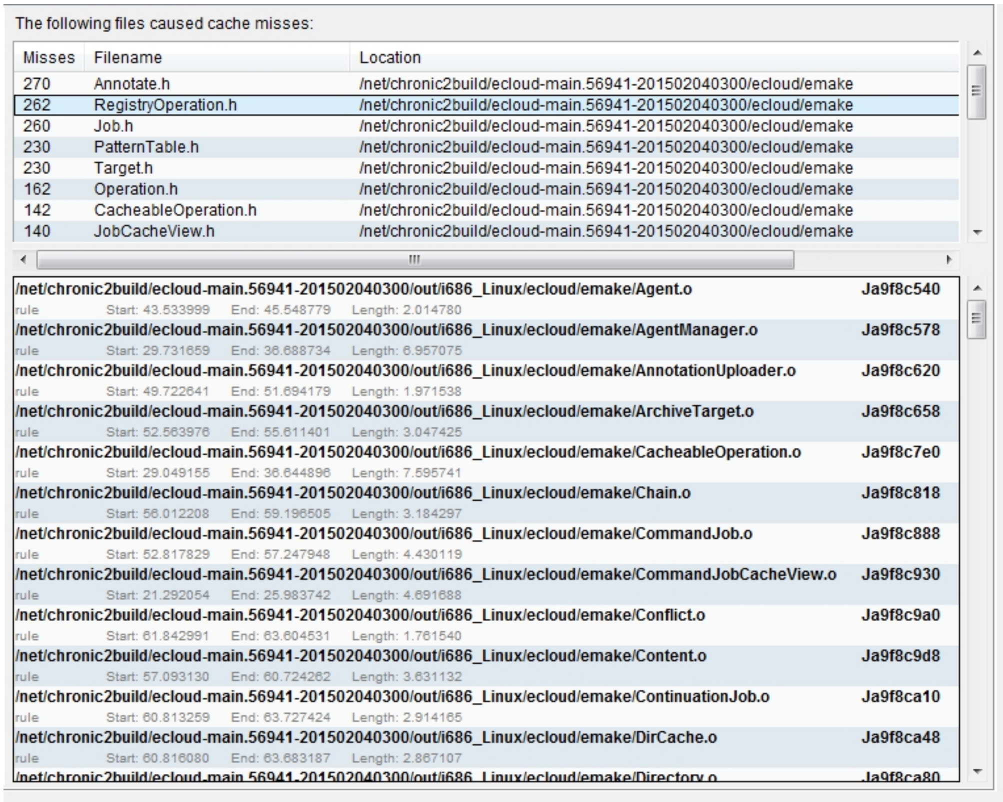
JobCache lets a build avoid recompiling object files that it previously built, if their inputs have not changed. For more information about JobCache, see the *ElectricAccelerator Electric Make User Guide* at http://docs.electric-cloud.com/accelerator_doc/AcceleratorIndex.html. Parse avoidance lets a build almost eliminate makefile parse time by caching and reusing parse result files, if their inputs have not changed.

This report finds files that caused cache misses, counts the number of jobs missed because of changes in each file (such as modified file contents) and displays what changed. This report lists the files that caused misses, how many misses each file caused, and which misses each file caused.

The report requires

```
--emake-annodetail=jobcache
```

To see the list of jobs that were cache misses because of changes to a specific file, click the file. Following is a list of misses for a file named RegistryOperation.h:



The following files caused cache misses:

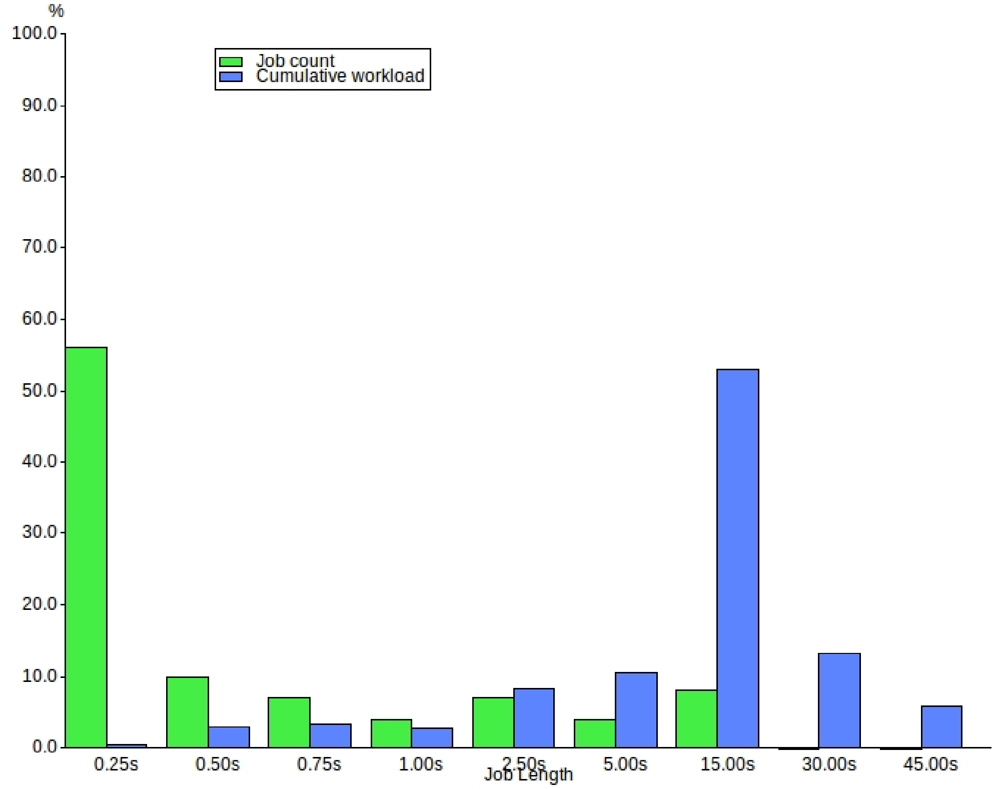
Misses	Filename	Location
270	Annotate.h	/net/chronic2build/ecloud-main.56941-201502040300/ecloud/emake
262	RegistryOperation.h	/net/chronic2build/ecloud-main.56941-201502040300/ecloud/emake
260	Job.h	/net/chronic2build/ecloud-main.56941-201502040300/ecloud/emake
230	PatternTable.h	/net/chronic2build/ecloud-main.56941-201502040300/ecloud/emake
230	Target.h	/net/chronic2build/ecloud-main.56941-201502040300/ecloud/emake
162	Operation.h	/net/chronic2build/ecloud-main.56941-201502040300/ecloud/emake
142	CacheableOperation.h	/net/chronic2build/ecloud-main.56941-201502040300/ecloud/emake
140	JobCacheView.h	/net/chronic2build/ecloud-main.56941-201502040300/ecloud/emake

```
/net/chronic2build/ecloud-main.56941-201502040300/out/i686_Linux/ecloud/emake/Agent.o                  Ja9f8c540
rule         Start: 43.533999    End: 45.548779    Length: 2.014780
/net/chronic2build/ecloud-main.56941-201502040300/out/i686_Linux/ecloud/emake/AgentManager.o           Ja9f8c578
rule         Start: 29.731659    End: 36.688734    Length: 6.957075
/net/chronic2build/ecloud-main.56941-201502040300/out/i686_Linux/ecloud/emake/AnnotationUploader.o     Ja9f8c620
rule         Start: 49.722641    End: 51.694179    Length: 1.971538
/net/chronic2build/ecloud-main.56941-201502040300/out/i686_Linux/ecloud/emake/ArchiveTarget.o          Ja9f8c658
rule         Start: 52.563976    End: 55.611401    Length: 3.047425
/net/chronic2build/ecloud-main.56941-201502040300/out/i686_Linux/ecloud/emake/CacheableOperation.o     Ja9f8c7e0
rule         Start: 29.049155    End: 36.644896    Length: 7.595741
/net/chronic2build/ecloud-main.56941-201502040300/out/i686_Linux/ecloud/emake/Chain.o                  Ja9f8c818
rule         Start: 56.012208    End: 59.196505    Length: 3.184297
/net/chronic2build/ecloud-main.56941-201502040300/out/i686_Linux/ecloud/emake/CommandJob.o             Ja9f8c888
rule         Start: 52.817829    End: 57.247948    Length: 4.430119
/net/chronic2build/ecloud-main.56941-201502040300/out/i686_Linux/ecloud/emake/CommandJobCacheView.o    Ja9f8c930
rule         Start: 21.292054    End: 25.983742    Length: 4.691688
/net/chronic2build/ecloud-main.56941-201502040300/out/i686_Linux/ecloud/emake/Conflict.o               Ja9f8c9a0
rule         Start: 61.842991    End: 63.604531    Length: 1.761540
/net/chronic2build/ecloud-main.56941-201502040300/out/i686_Linux/ecloud/emake/Content.o                Ja9f8c9d8
rule         Start: 57.093130    End: 60.724262    Length: 3.631132
/net/chronic2build/ecloud-main.56941-201502040300/out/i686_Linux/ecloud/emake/ContinuationJob.o        Ja9f8ca10
rule         Start: 60.813259    End: 63.727424    Length: 2.914165
/net/chronic2build/ecloud-main.56941-201502040300/out/i686_Linux/ecloud/emake/DirCache.o               Ja9f8ca48
rule         Start: 60.816080    End: 63.683187    Length: 2.867107
/net/chronic2build/ecloud-main.56941-201502040300/out/i686_Linux/ecloud/emake/Directory.o              Ja9f8ca80
```

Note: The sum of the miss counts might not equal the total number of jobs with cache misses, because when a job has a miss because of two or more files, it increases the miss counts for each of those files. For example, if two files each cause misses for two different jobs, a count of two is displayed for each file.

Job Stats

This report groups jobs based on duration and shows the percentage of jobs in each group, as well as the percentage of total build workload represented by the jobs in each group.



Command-Line Interface Report

Run this command:

```
einsight --report=JobStats <annotation>
```

Note: On Windows, use `einsight-cmd` instead of `einsight` when running reports from the command line.

Where <*annotation*> is the path of the annotation file.

You can also limit results to a job class, such as "compile" or "link". To limit your results, run this command:

```
einsight --report="JobStats <jobclass>" <annotation>
```

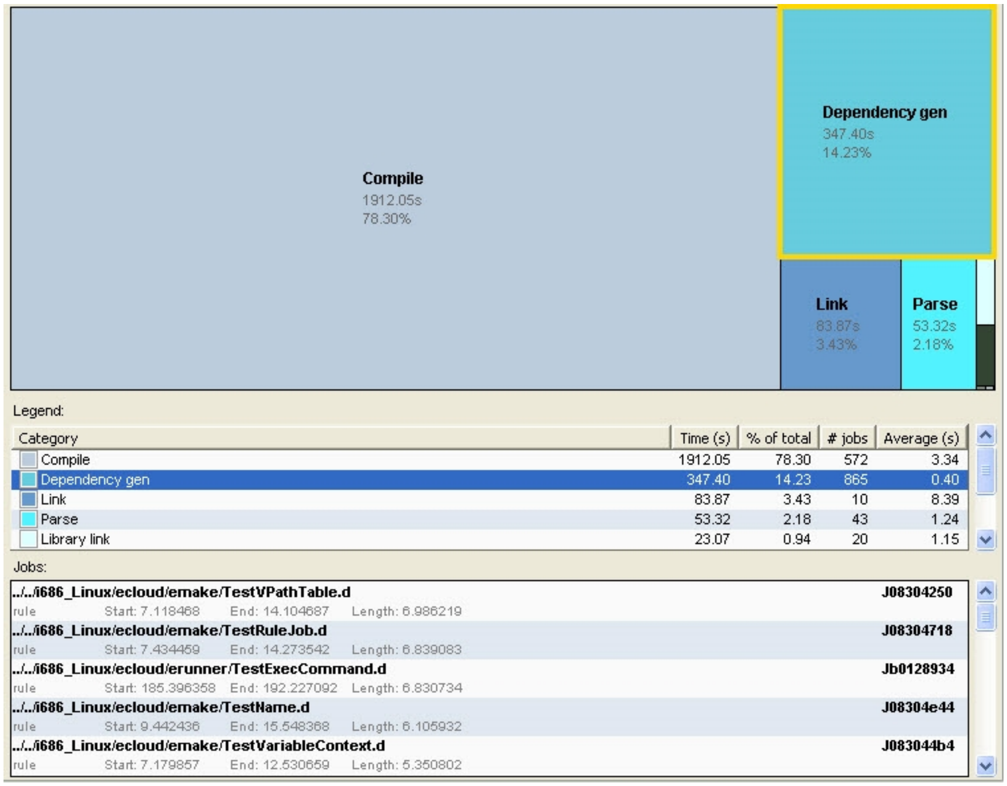
Where <*jobclass*> is the only job class you want to see, which can be link, compile, parse, etc.

Results format:

```
duration,count,time,count percent,time percent
```

Job Time by Type

This report shows the portion of total job time consumed by each job type. The area covered by each color is proportional to the total time consumed by its corresponding job type.



Legend:

Category	Time (s)	% of total	# jobs	Average (s)
Compile	1912.05	78.30	572	3.34
Dependency gen	347.40	14.23	865	0.40
Link	83.87	3.43	10	8.39
Parse	53.32	2.18	43	1.24
Library link	23.07	0.94	20	1.15

Jobs:

```
../../i686_Linux/ecloud/emake/TestVPathTable.d                J08304250
rule       Start: 7.118468     End: 14.104687    Length: 6.986219
../../i686_Linux/ecloud/emake/TestRuleJob.d                   J08304718
rule       Start: 7.434459     End: 14.273542    Length: 6.839083
../../i686_Linux/ecloud/erunner/TestExecCommand.d             Jb0128934
rule       Start: 185.396358   End: 192.227092   Length: 6.830734
../../i686_Linux/ecloud/emake/TestName.d                      J08304e44
rule       Start: 9.442436     End: 15.548368    Length: 6.105932
../../i686_Linux/ecloud/emake/TestVariableContext.d           J083044b4
rule       Start: 7.179857     End: 12.530659    Length: 5.350802
```

The legend displays corresponding colors for job types. The following information is available:

- Category
- Time (s)
- % of total
- # jobs
- Average (s)

Note: To sort data, click a column heading.

Clicking a row outlines the corresponding area in the treemap in yellow and lists all jobs of that type in the Jobs section. Double-clicking a job from the list displays that job’s details. You can also display all jobs of a specific type by clicking an area of the treemap.

Command-Line Interface Report

Run this command:

```
einsight --report=JobTimeByType <annotation>
```

Note: On Windows, use `einsight-cmd` instead of `einsight` when running reports from the command line.

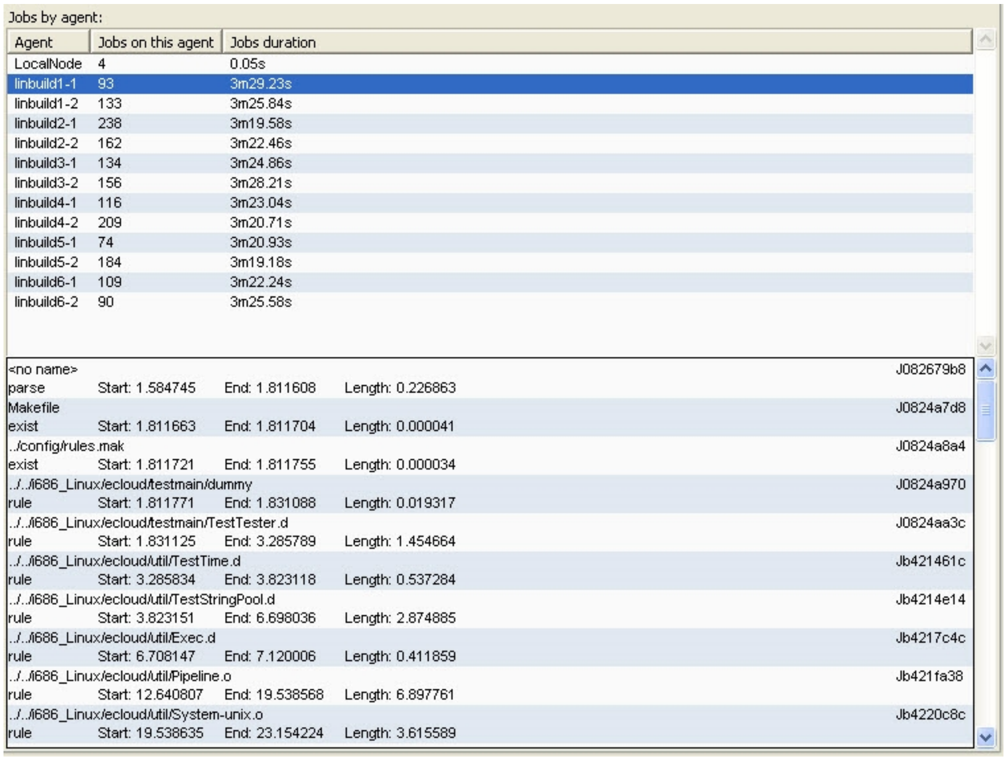
Where <*annotation*> is the path of the annotation file.

Results format:

```
class,count,percent,seconds,average
```

Jobs by Agent

This report lists the number of jobs run by each agent in the build.



Jobs by agent:

Agent	Jobs on this agent	Jobs duration
LocalNode	4	0.05s
linbuild1-1	93	3m29.23s
linbuild1-2	133	3m25.84s
linbuild2-1	238	3m19.58s
linbuild2-2	162	3m22.46s
linbuild3-1	134	3m24.86s
linbuild3-2	156	3m28.21s
linbuild4-1	116	3m23.04s
linbuild4-2	209	3m20.71s
linbuild5-1	74	3m20.93s
linbuild5-2	184	3m19.18s
linbuild6-1	109	3m22.24s
linbuild6-2	90	3m25.58s

```
<no name>                                                                   J082679b8
parse      Start: 1.584745    End: 1.811608     Length: 0.226863
Makefile                                                                    J0824a7d8
exist      Start: 1.811663    End: 1.811704     Length: 0.000041
../config/rules.mak                                                         J0824a8a4
exist      Start: 1.811721    End: 1.811755     Length: 0.000034
../../i686_Linux/ecloud/testmain/dummy                                      J0824a970
rule       Start: 1.811771    End: 1.831088     Length: 0.019317
../../i686_Linux/ecloud/testmain/TestTester.d                               J0824aa3c
rule       Start: 1.831125    End: 3.285789     Length: 1.454664
../../i686_Linux/ecloud/util/TestTime.d                                     Jb421461c
rule       Start: 3.285834    End: 3.823118     Length: 0.537284
../../i686_Linux/ecloud/util/TestStringPool.d                               Jb4214e14
rule       Start: 3.823151    End: 6.698036     Length: 2.874885
../../i686_Linux/ecloud/util/Exec.d                                         Jb4217c4c
rule       Start: 6.708147    End: 7.120006     Length: 0.411859
../../i686_Linux/ecloud/util/Pipeline.o                                     Jb421fa38
rule       Start: 12.640807   End: 19.538568    Length: 6.897761
../../i686_Linux/ecloud/util/System-unix.o                                  Jb4220c8c
rule       Start: 19.538635   End: 23.154224    Length: 3.615589
```

Click an agent to display the jobs that it ran. Double-click a job to display its details.

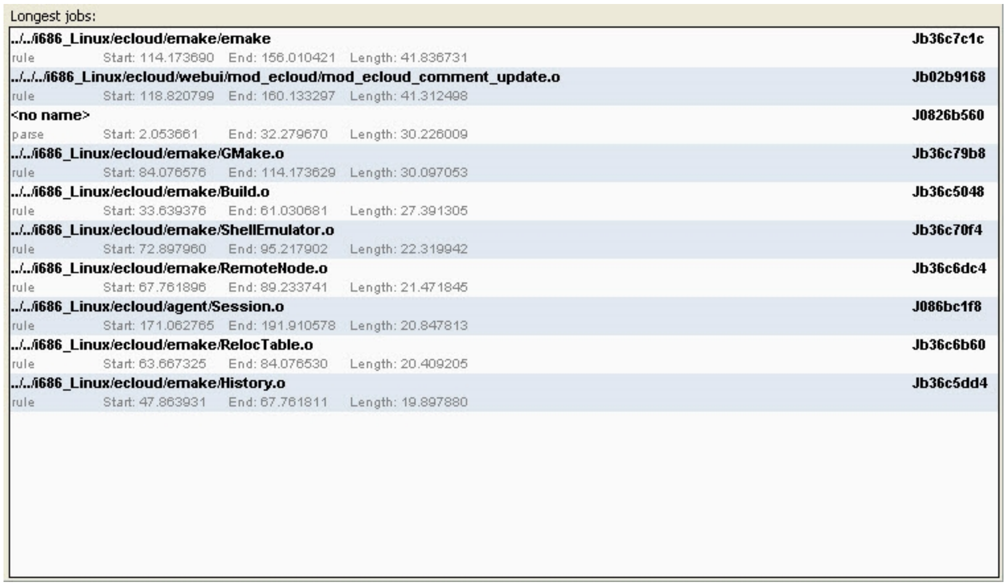
Jobs by File

This report lists which jobs read or wrote a particular file.

Type in or browse to the file you want to analyze and click **Analyze**.

Longest Jobs

This report lists the build’s 10 longest jobs.



```
Longest jobs:
../../i686_Linux/ecloud/emake/emake                                            Jb36c7c1c
rule          Start: 114.173690   End: 156.010421   Length: 41.836731
../../../i686_Linux/ecloud/webui/mod_ecloud/mod_ecloud_comment_update.o        Jb02b9168
rule          Start: 118.820799   End: 160.133297   Length: 41.312498
<no name>                                                                      J0826b560
parse         Start: 2.053661     End: 32.279670    Length: 30.226009
../../i686_Linux/ecloud/emake/GMake.o                                          Jb36c79b8
rule          Start: 84.076576    End: 114.173629   Length: 30.097053
../../i686_Linux/ecloud/emake/Build.o                                          Jb36c5048
rule          Start: 33.639376    End: 61.030681    Length: 27.391305
../../i686_Linux/ecloud/emake/ShellEmulator.o                                  Jb36c70f4
rule          Start: 72.897960    End: 95.217902    Length: 22.319942
../../i686_Linux/ecloud/emake/RemoteNode.o                                     Jb36c6dc4
rule          Start: 67.761896    End: 89.233741    Length: 21.471845
../../i686_Linux/ecloud/agent/Session.o                                        J086bc1f8
rule          Start: 171.062765   End: 191.910578   Length: 20.847813
../../i686_Linux/ecloud/emake/RelocTable.o                                     Jb36c6b60
rule          Start: 63.667325    End: 84.076530    Length: 20.409205
../../i686_Linux/ecloud/emake/History.o                                        Jb36c5dd4
rule          Start: 47.863931    End: 67.761811    Length: 19.897880
```

To view the Job Details dialog, double-click a job ID.

Command-Line Interface Report

Run this command:

```
einsight --report=LongestJobs <annotation>
```

Note: On Windows, use `einsight-cmd` instead of `einsight` when running reports from the command line.

Where *<annotation>* is the path of the annotation file.

Results format:

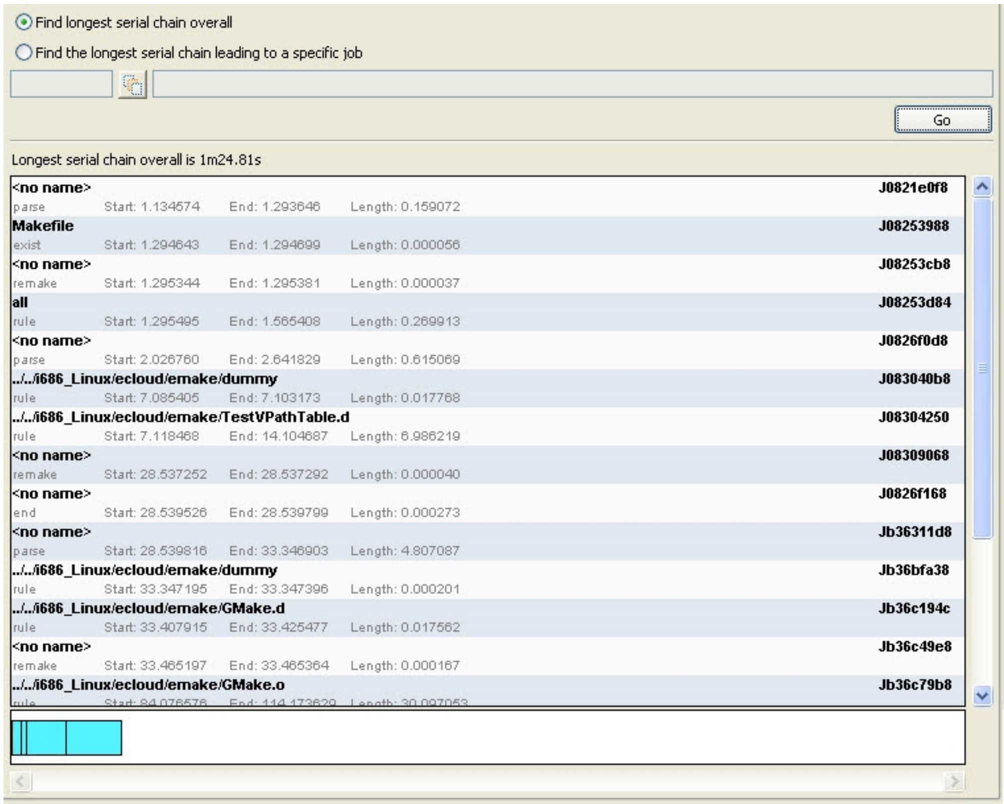
```
job id,duration,name
```

Longest Serial Chain

This report displays the sequence of serialized jobs with the longest end-to-end runtime in the build. The longest serial chain represents a lower bound on the build runtime. Without changing the structure or content of the build, the runtime cannot be less than the longest serial chain.

This report requires

```
--emake-annodetail=waiting,history
```



● Find longest serial chain overall

○ Find the longest serial chain leading to a specific job

Longest serial chain overall is 1m24.81s

```
<no name>                                                      J0821e0f8
parse      Start: 1.134574    End: 1.293646    Length: 0.159072
Makefile                                                       J08253988
exist      Start: 1.294643    End: 1.294699    Length: 0.000056
<no name>                                                      J08253cb8
remake     Start: 1.295344    End: 1.295381    Length: 0.000037
all                                                            J08253d84
rule       Start: 1.295495    End: 1.565408    Length: 0.269913
<no name>                                                      J0826f0d8
parse      Start: 2.026760    End: 2.641829    Length: 0.615069
../../i686_Linux/ecloud/emake/dummy                            J083040b8
rule       Start: 7.085405    End: 7.103173    Length: 0.017768
../../i686_Linux/ecloud/emake/TestVPathTable.d                 J08304250
rule       Start: 7.118468    End: 14.104687   Length: 6.986219
<no name>                                                      J08309068
remake     Start: 28.537252   End: 28.537292   Length: 0.000040
<no name>                                                      J0826f168
end        Start: 28.539526   End: 28.539799   Length: 0.000273
<no name>                                                      Jb36311d8
parse      Start: 28.539816   End: 33.346903   Length: 4.807087
../../i686_Linux/ecloud/emake/dummy                            Jb36bfa38
rule       Start: 33.347195   End: 33.347396   Length: 0.000201
../../i686_Linux/ecloud/emake/GMake.d                          Jb36c194c
rule       Start: 33.407915   End: 33.425477   Length: 0.017562
<no name>                                                      Jb36c49e8
remake     Start: 33.465197   End: 33.465364   Length: 0.000167
../../i686_Linux/ecloud/emake/GMake.o                          Jb36c79b8
rule       Start: 84.076576   End: 114.173629  Length: 30.097053
```

You can display the longest serial chain overall or the longest serial chain leading to a specific job.

To display the Job Details dialog, double-click a job.

Command-Line Interface Report

Run this command:

```
einsight --report="LongestSerialChain [job id]" <annotation>
```

Note: On Windows, use `einsight-cmd` instead of `einsight` when running reports from the command line.

Where:

[job id] is a job ID (optional). Specifying a job ID means that you want to find the longest serial chain leading to that job. If you don't specify a job ID, the report returns the longest serial chain overall.

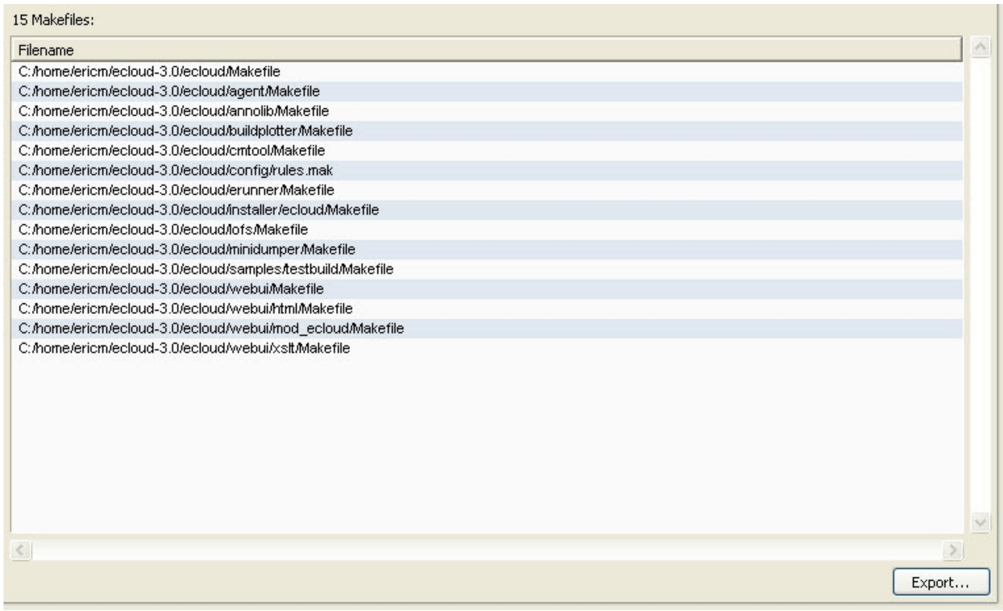
<annotation> is the path of the annotation file.

Results format:

```
job,type,length,name
```

Makefile Manifest

This report lists all makefiles used in the build.



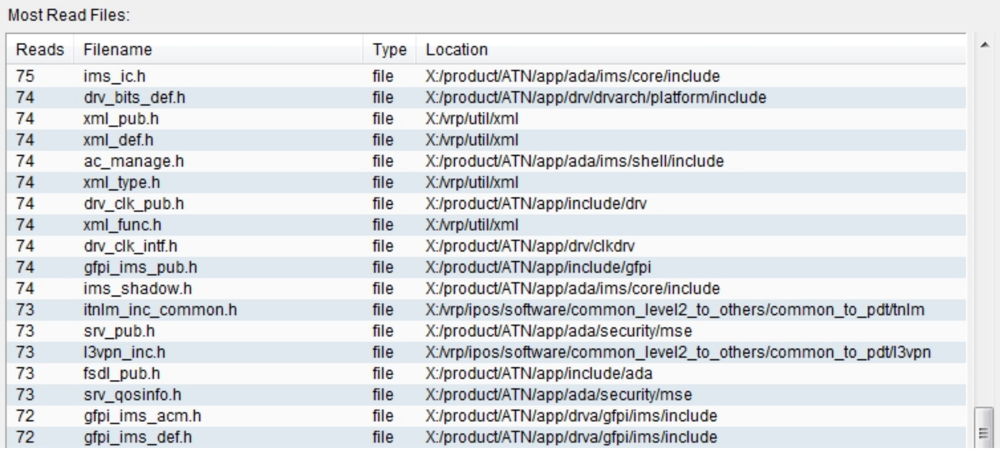
15 Makefiles:

Filename
C:/home/ericm/ecloud-3.0/ecloud/Makefile
C:/home/ericm/ecloud-3.0/ecloud/agent/Makefile
C:/home/ericm/ecloud-3.0/ecloud/annolib/Makefile
C:/home/ericm/ecloud-3.0/ecloud/buildplotter/Makefile
C:/home/ericm/ecloud-3.0/ecloud/cmtool/Makefile
C:/home/ericm/ecloud-3.0/ecloud/config/rules.mak
C:/home/ericm/ecloud-3.0/ecloud/erunner/Makefile
C:/home/ericm/ecloud-3.0/ecloud/installer/ecloud/Makefile
C:/home/ericm/ecloud-3.0/ecloud/lofs/Makefile
C:/home/ericm/ecloud-3.0/ecloud/minidumper/Makefile
C:/home/ericm/ecloud-3.0/ecloud/samples/testbuild/Makefile
C:/home/ericm/ecloud-3.0/ecloud/webui/Makefile
C:/home/ericm/ecloud-3.0/ecloud/webui/html/Makefile
C:/home/ericm/ecloud-3.0/ecloud/webui/mod_ecloud/Makefile
C:/home/ericm/ecloud-3.0/ecloud/webui/xslt/Makefile

To export the list to a file, click **Export**.

Most Read Files

This report lists the number of times a file was read, its name, type, and location.



Most Read Files:

Reads	Filename	Type	Location
75	ims_ic.h	file	X:/product/ATN/app/ada/ims/core/include
74	drv_bits_def.h	file	X:/product/ATN/app/drv/drvarch/platform/include
74	xml_pub.h	file	X:/vrp/util/xml
74	xml_def.h	file	X:/vrp/util/xml
74	ac_manage.h	file	X:/product/ATN/app/ada/ims/shell/include
74	xml_type.h	file	X:/vrp/util/xml
74	drv_clk_pub.h	file	X:/product/ATN/app/include/drv
74	xml_func.h	file	X:/vrp/util/xml
74	drv_clk_intf.h	file	X:/product/ATN/app/drv/clkdrv
74	gfpi_ims_pub.h	file	X:/product/ATN/app/include/gfpi
74	ims_shadow.h	file	X:/product/ATN/app/ada/ims/core/include
73	itnlm_inc_common.h	file	X:/vrp/ipos/software/common_level2_to_others/common_to_pdt/tnlm
73	srv_pub.h	file	X:/product/ATN/app/ada/security/mse
73	l3vpn_inc.h	file	X:/vrp/ipos/software/common_level2_to_others/common_to_pdt/l3vpn
73	fsdl_pub.h	file	X:/product/ATN/app/include/ada
73	srv_qosinfo.h	file	X:/product/ATN/app/ada/security/mse
72	gfpi_ims_acm.h	file	X:/product/ATN/app/drva/gfpi/ims/include
72	gfpi_ims_def.h	file	X:/product/ATN/app/drva/gfpi/ims/include

Command-Line Interface Report

Run this command:

```
einsight --report=MostReadFiles <annotation>
```

Note: On Windows, use `einsight-cmd` instead of `einsight` when running reports from the command line.

Where *<annotation>* is the path of the annotation file.

Results format:

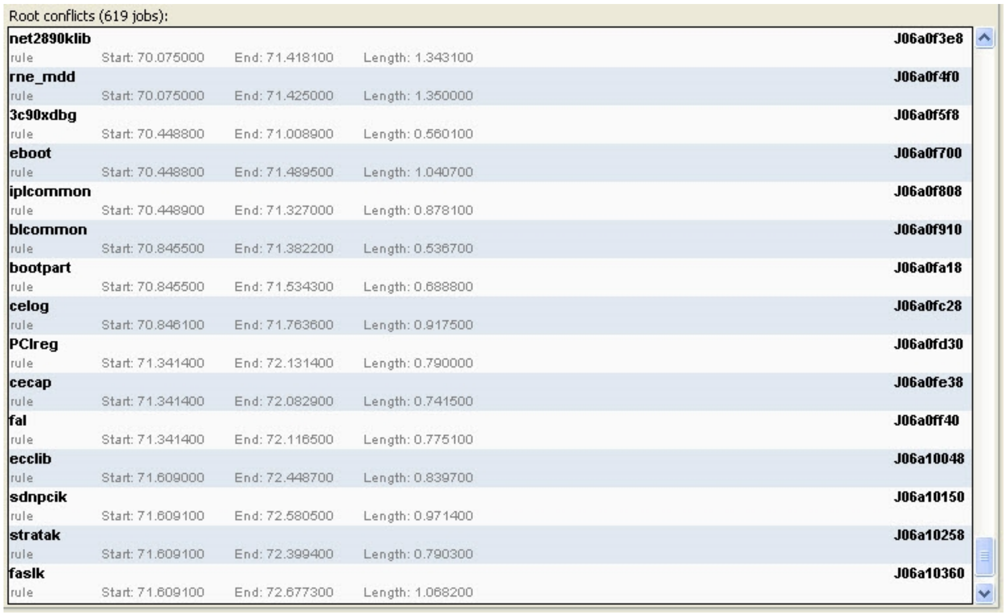
```
count,type,name,location
```

Root Conflicts

This report lists all root conflicts in the build.

Insight divides all conflicts found in a build into two categories:

- root conflicts, which are conflicts that are not caused by an earlier conflict
- conflicts resulting from earlier conflicts (for example, jobs that are in conflict with jobs that are rerun jobs)



```
Root conflicts (619 jobs):
net2890klib                                                     J06a0f3e8
rule      Start: 70.075000   End: 71.418100   Length: 1.343100
rne_mdd                                                         J06a0f4f0
rule      Start: 70.075000   End: 71.425000   Length: 1.350000
3c90xdbg                                                        J06a0f5f8
rule      Start: 70.448800   End: 71.008900   Length: 0.560100
eboot                                                           J06a0f700
rule      Start: 70.448800   End: 71.489500   Length: 1.040700
iplcommon                                                       J06a0f808
rule      Start: 70.448900   End: 71.327000   Length: 0.878100
blcommon                                                        J06a0f910
rule      Start: 70.845500   End: 71.382200   Length: 0.536700
bootpart                                                        J06a0fa18
rule      Start: 70.845500   End: 71.534300   Length: 0.688800
celog                                                           J06a0fc28
rule      Start: 70.846100   End: 71.763600   Length: 0.917500
PCIreg                                                          J06a0fd30
rule      Start: 71.341400   End: 72.131400   Length: 0.790000
cecap                                                           J06a0fe38
rule      Start: 71.341400   End: 72.082900   Length: 0.741500
fal                                                             J06a0ff40
rule      Start: 71.341400   End: 72.116500   Length: 0.775100
ecclib                                                          J06a10048
rule      Start: 71.609000   End: 72.448700   Length: 0.839700
sdnpcik                                                         J06a10150
rule      Start: 71.609100   End: 72.580500   Length: 0.971400
stratak                                                         J06a10258
rule      Start: 71.609100   End: 72.399400   Length: 0.790300
faslk                                                           J06a10360
rule      Start: 71.609100   End: 72.677300   Length: 1.068200
```

Serialization Analysis

This report details dependencies between a pair of jobs. For example, you see in the Annotation tab that a job has waitingJobs and you want to know why the jobs are waiting.

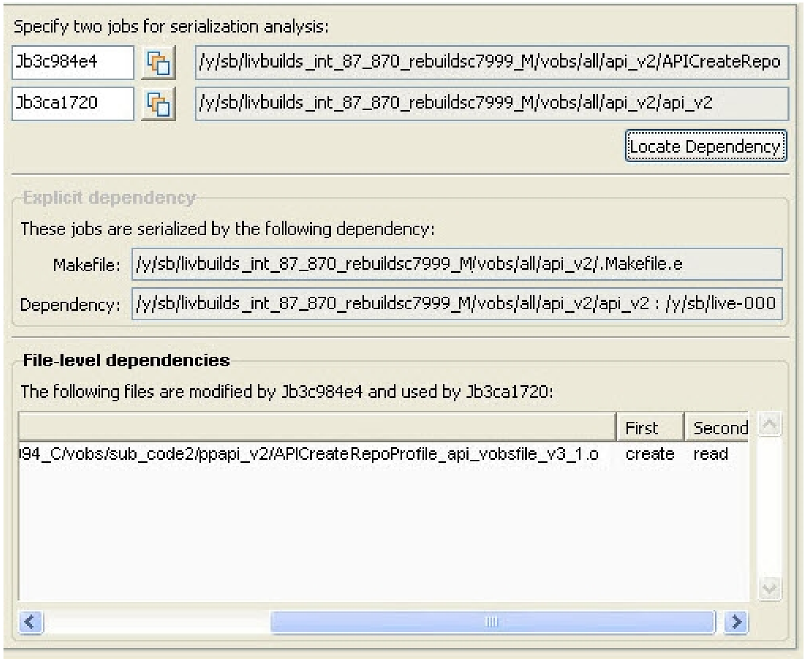
In some cases you can use this report to understand the longest serial chain, but dependencies might exist that are not within the scope of the Serialization analysis report. For example, it is common for a rule job to be preceded by a parse job. This type of logical dependency is not included in the Serialization analysis report.

This report requires

```
--emake-annodetail=waiting,history
```

but works best with

```
--emake-annodetail=waiting,history,registry
```



Specify two jobs for serialization analysis:

Jb3c984e4 /y/sb/livbuilds_int_87_870_rebuildsc7999_M/vobs/all/api_v2/APICreateRepo

Jb3ca1720 /y/sb/livbuilds_int_87_870_rebuildsc7999_M/vobs/all/api_v2/api_v2

Explicit dependency

These jobs are serialized by the following dependency:

Makefile: /y/sb/livbuilds_int_87_870_rebuildsc7999_M/vobs/all/api_v2/.Makefile.e

Dependency: /y/sb/livbuilds_int_87_870_rebuildsc7999_M/vobs/all/api_v2/api_v2 : /y/sb/live-000

File-level dependencies

The following files are modified by Jb3c984e4 and used by Jb3ca1720:

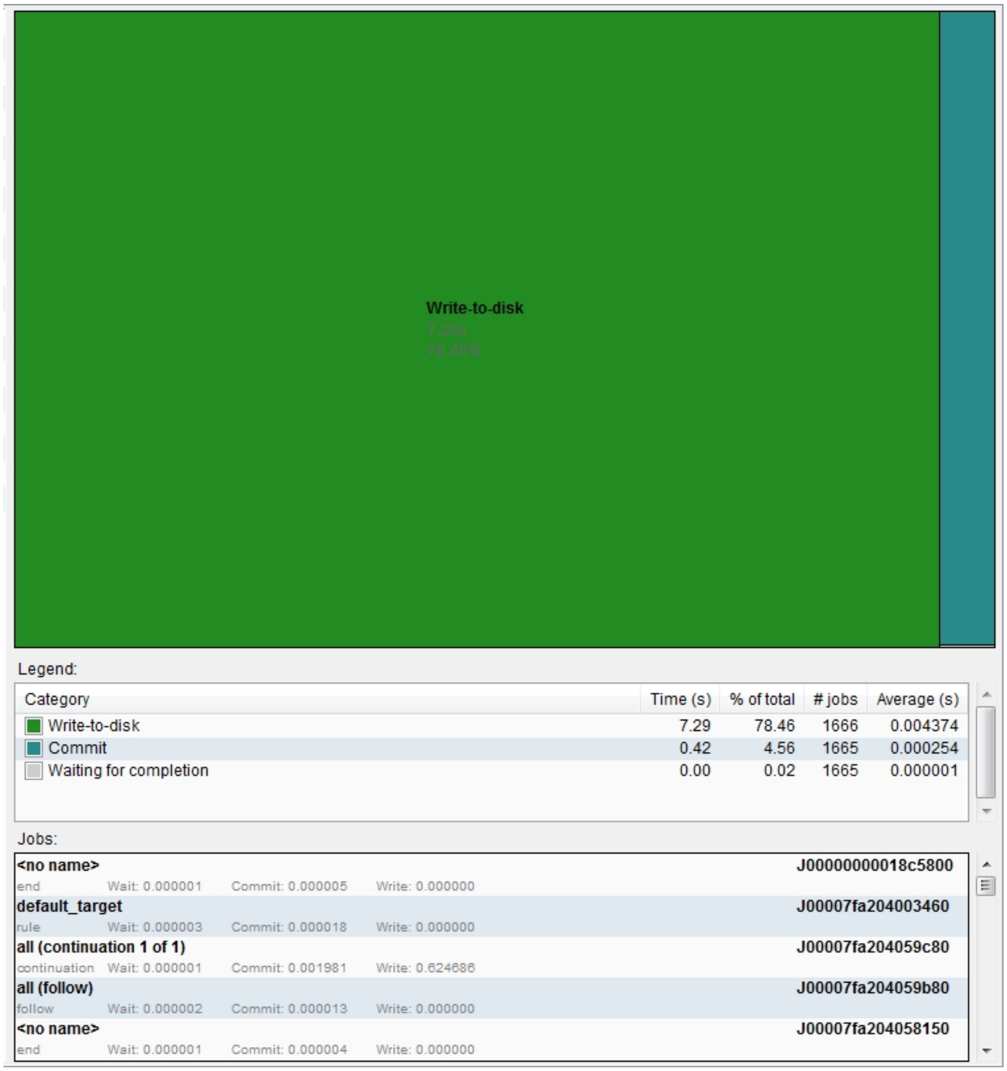
	First	Second
)94_C/vobs/sub_code2/ppapi_v2/APICreateRepoProfile_api_vobsfile_v3_1.o	create	read

1. Type the ID of the job you started first in the first job field.
2. Type the ID of the waiting job in the second job field.
3. Click **Locate Dependency**.

Note: If you know where the jobs of interest appear in the main jobs display, you can use the buttons next to the job fields to select the jobs.

Terminator Lag

Terminator lag time is the time that the terminator is active after the completion of the last job in the build. The Terminator Lag report shows the portions of terminator lag time that are used by terminator activities such as commit, write-to-disk, and wait-for-completion.



Legend:

Category	Time (s)	% of total	# jobs	Average (s)
🟩 Write-to-disk	7.29	78.46	1666	0.004374
🟦 Commit	0.42	4.56	1665	0.000254
□ Waiting for completion	0.00	0.02	1665	0.000001

Jobs:

```
<no name>                                                          J00000000018c5800
end            Wait: 0.000001   Commit: 0.000005   Write: 0.000000
default_target                                                     J00007fa204003460
rule           Wait: 0.000003   Commit: 0.000018   Write: 0.000000
all (continuation 1 of 1)                                          J00007fa204059c80
continuation   Wait: 0.000001   Commit: 0.001981   Write: 0.624686
all (follow)                                                       J00007fa204059b80
follow         Wait: 0.000002   Commit: 0.000013   Write: 0.000000
<no name>                                                          J00007fa204058150
end            Wait: 0.000001   Commit: 0.000004   Write: 0.000000
```

To see the list of jobs that are associated with a category of terminator activities, click the category. To see the details that are associated with a job, double-click the job to view the Job Details window. For a description of the information that appears in the Job Details window, see the Job Details Dialog section.

Command-Line-Enabled Reports

This section lists all command-line-enabled reports and their associated commands. For all reports, *<annotation>* is required and represents the path of the annotation file.

> **IMPORTANT:** For Windows, use `einsight-cmd` instead of `einsight` when running reports from the command line.

Build Metrics

```
einsight --report=BuildMetrics <annotation>
```

Results format:

```
name,value
```

Build Summary

```
einsight --report=BuildSummary <annotation>
```

Results format:

```
name,value
```

ElectricSimulator

```
einsight --report="ElectricSimulator [<minagents>] [<maxagents>] [<stepsize>]" <ann
otation>
```

Where:

[<minagents>] (optional) is the minimum number of agents to simulate. Default is 4.

[<maxagents>] (optional) is the maximum number of agents to simulate. Default is 32.

[<stepsize>] (optional) is the number of agents to increment the count each time. Default is 4.

Results format:

```
agents,duration
```

Export Timeline

```
einsight --report=ExportTimeline <filename> [<mode>] [<scale>] [<rowheight>] [<rows
pace>] <annotation>
```

Where:

- *<filename>* is the name of the PNG file to write to. (Required)
- *[<mode>]* indicates if you want the image in color or black and white. Valid values are `color` or `bw`. Default is `color`. (Optional)
- *[<scale>]* is the number of seconds per pixel. Default is `2`. (Optional)
- *[<rowheight>]* is the number of pixels per row of jobs. Default is `1`. (Optional)
- *[<rowspace>]* is the number of pixels separating rows. Default is `1`. (Optional)

Jobs by Agent

```
einsight --report=JobsByAgent <annotation>
```

Results format:

Table that contains the job count and total time for each agent.

Job Stats

```
einsight --report=JobStats <annotation>
```

You can also limit results to a job class, such as "compile" or "link". To limit your results, run this command:

```
einsight --report="JobStats <jobclass>" <annotation>
```

Where *<jobclass>* is the only job class you want to see, which can be link, compile, parse, etc.

Results format:

```
duration,count,time,count percent,time percent
```

Job Time by Type

```
einsight --report=JobTimeByType <annotation>
```

Results format:

```
class,count,percent,seconds,average
```

Longest Jobs

```
einsight --report=LongestJobs <annotation>
```

Results format:

```
job id,duration,name
```

Longest Serial Chain

```
einsight --report="LongestSerialChain [<job id>]" <annotation>
```

Where *[<job id>]* is a job ID (optional). Specifying a job ID means that you want to find the longest serial chain leading to that job. If you don't specify a job ID, the report returns the longest serial chain overall.

Results format:

```
job,type,length,name
```

Most Read Files

```
einsight --report=MostReadFiles <annotation>
```

Results format:

```
count,type,name,location
```

Terminator Lag

```
einsight --report=TerminatorLag <annotation>
```

Results format:

```
activity,count,percent,seconds,average
```

Creating a Custom Report

You can create custom reports and make them accessible in Insight. A report typically consists of

- a single .tcl source file containing Tcl code required to perform build annotation analysis
- the Tk code required to display the result in the Insight UI

For example, you can create a report to display the number of jobs that were run by each agent used in the build. Use the following steps to create a custom report.

1. Create a Tcl script that uses CreateReport and ConfigureReport to declare and set attributes for a report. Use this format:

   ```
   CreateReport name ?-<option> <value> ...?
   ```

   Use the following options:

`-command`	Generates a text-only version of the report
`-guicommand`	Generates a GUI version of the report

`-desc`	Specifies a report
`-requires`	Specifies a list of anno detail levels required for the report. The list must be space-separated
`-cleanup`	Cleans up the state associated with the report. For example, when a new anno file is loaded

You must use at least one `-command` option and at least one `-guicommand` option. For example:

```
set report [CreateReport "MyCustomReport"]
ConfigureReport $report -desc "The report description \
a continuation of the report description."
ConfigureReport $report -requires {basic file waiting history}
ConfigureReport $report -command RunCustomReport
ConfigureReport $report -uicommand MakeCustomReportUI
```

Note: Make sure the console version of the report returns the report results as a text string (do not merely print directly with [puts]).

If you are running the Insight GUI and you invoke a custom report that has no `-guicommand`, Insight will run the `-command` version, capture the output, and display it as raw text in the GUI.

2. Add the function to perform the analysis and display results to the Tcl script. The function is invoked with two arguments:

   - The name of the widget that the function needs to create to display results
   - The name of the variable that the function needs to update with progress information, from 0.0 to 1.0. This information controls the progress bar when the report is generated. The `progressVar` argument is optional

   ```
   proc CreateJobsByAgentReport {w progressVar} {
   ```

   In addition to function arguments, a global variable `anno` is available for reports. This is a handle for the `annolib` object containing the build annotation information that is currently loaded in ElectricInsight. To access this information in your report, import the `anno` variable:

   ```
   global anno
   ```

3. Add Tcl code to perform the analysis on the annotation information.

   - Following is an example that shows how to forward iterate through all the jobs in the build, counting the number of jobs run by each agent:

     ```
     array set count {}
     set end [$anno jobs end]
     for {set j [$anno jobs begin]} {$j != $end} {set j [$anno job next $j]} {
        set agent [$anno job agent $j]
        if { $agent ne "" } {
           if { [info exists count($agent)] } {
              incr count($agent)
           } else {
              set count($agent) 1
           }
        }
     }
     ```

   - Following is the same example using reverse iteration:

     ```
     array set count {}
     set rend [$anno jobs rend]
     for {set j [$anno jobs rbegin]} {$j != $rend} {set j [$anno job prev $j]} {
        set agent [$anno job agent $j]
        if { $agent ne "" } {
           if { [info exists count($agent)] } {
              incr count($agent)
           } else {
              set count($agent) 1
           }
        }
     }
     ```

4. Write Tk code to display results.

   Insight includes the `[Tile]` widgets and the `[Tablelist]` widget. For this report, a simple tablelist can display the results:

   ```
   # The “count” array contains the number of jobs run by
   # each agent.
   # We create a simple tablelist (multi-column listbox)
   # to display the results.

   frame $w
   ttk::label $w.label -text “Jobs by agent:” -anchor w
   tablelist::tablelist $w.results -columns {
      0 “Agent”
      0 “Jobs run by this agent”
   }  -height 10 -width 80 -borderwidth 1 -stretch end \
      -font TkDefaultFont -background gray98 \
      -stripebackground \#e0e8f0 \
      -labelcommand tablelist::sortByColumn
   $w.results columnconfigure 1 -sortmode integer
   grid $w.label -sticky ew
   grid $w.results -sticky nsew
   grid columnconfigure $w 0 -weight 1
   grid rowconfigure $w 1 -weight 1

   foreach agent [lsort [array names count]] {
      $w.results insert end [list $agent $count($agent)]
   }
   return $w
      }
      # end of procedure CreateJobsByAgentReport
   ```

5. Save the .tcl files that define your new report in one of the following directories.

   - Linux:
     - /opt/ecloud/ElectricInsight/reports
     - $HOME/.ecloud/ElectricInsight/reports
   - Windows:
     - c:\ecloud\ElectricInsight\reports
     - $USERPROFILE\Electric Cloud\ElectricInsight\reports

   At startup, Insight scans the above directories for .tcl files defining new reports and automatically includes the reports the next time that Insight starts. Reports that are saved in the above Electric Cloud directories are available to all users running Insight. Reports in other locations are available to a single user only.

6. Click **Tools > Reload reports**.

Chapter 6: Understanding Build Performance

This section provides guidance for using Insight to help you understand build performance issues.

- Specify the right annotation detail level

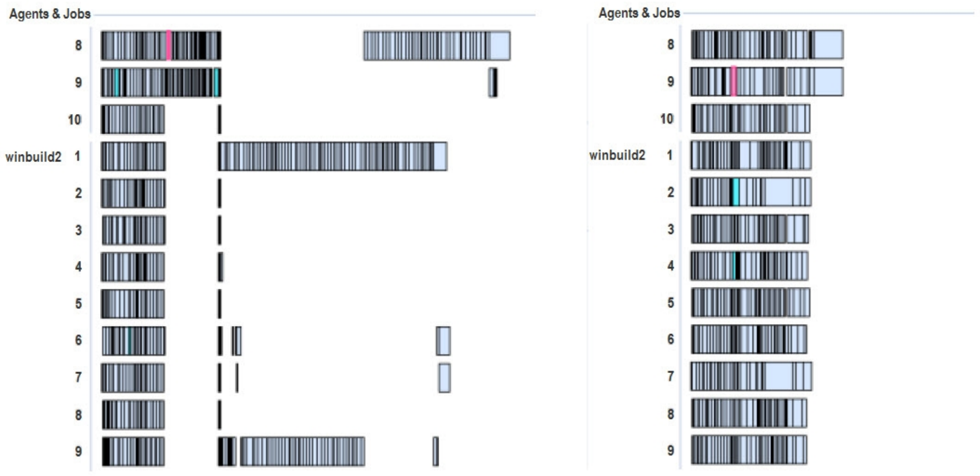
  At a minimum, specify annodetail to basic. Additional detail levels enable more sophisticated analysis of the build. For maximum performance analysis, set annodetail to “basic,lookup,waiting,history,registry”. Because additional information might affect performance slightly, specify basic detail level only, unless you are actively pursuing a performance issue.

- Use job details

  When viewing an annotation file in Insight, you can see if your build is slow due to excessive conflicts, over serializations, or insufficient decomposition of build steps. You can then drill-down into job details by double-clicking individual jobs in the Agents & Jobs section.

- Use reports
  - The Longest serial chain report can help estimate the best possible performance you can reasonably get from the build.
  - The Serialization analysis report can help you understand why jobs are serialized. In an optimal parallel build, all agents are busy at the same time.

    For example, the image on the left shows a build that suffers from over serialization. Using the Serialization analysis report, you can determine why the build is serialized and adjust the build to eliminate serializations. The image on the right shows the result of making the correct adjustments.



Before: 6m27s	**After:** 2m28s
2.61x faster	

- Watch for the following in the Agents & Jobs section:
  - One agent bar is longer than others
  - If you see gaps where an agent is not running a job, carefully examine the running jobs while some agents are idle
- Conflicts and reverted jobs

  The presence of conflict and reverted jobs means the build was slower than necessary. Reverted jobs are symptoms of job conflicts. To eliminate reversions, you must eliminate conflicts. Usually, running a build with a history file eliminates conflicts. Typically, the most expensive job conflicts are those involving parse jobs.

  If your build ran without a history file, you might have conflicts. If you had a pre-existing history file, your build should not contain conflicts unless the build changes affected job order and inter-dependencies.

Chapter 7: Annolib Programmer's Reference

Overview

annolib is the library used to parse annotation files. Required values are enclosed in `< >`(for example, *`<filename>`*). Optional values are enclosed in `?` (for example, `?count?`).

Note that in ElectricInsight 5.0, several new commands are introduced to allow any number of simultaneous iterators. The corresponding commands in earlier versions allow only one iteration to be “active” at a time; those commands are deprecated as indicated below.

API Commands

- `anno create`—Create a new anno object that can be used to query information from an annotation file. The return value is a handle that can be used in a subsequent `[$anno ...]` call. For example:

  ```
  set anno [anno create]

  $anno agents
  ```

- `$anno agents`—Retrieve a list of agents that participated in this build.
- `$anno comparejobs` *`<field> <jobId> <jobId>`*—Compare two jobs for sorting by the given field and return -1, 0, or 1 if the first job is earlier, equal to, or later than the second job.
- `$anno destroy`—Release all resources associated with this anno instance and remove the command for controlling it from the interpreter.
- `$anno duration`—Retrieve the length of the build that is described in the anno object. This is the greatest time index seen in the annotation file.
- `$anno environment`—Dump the contents of the environment table, which was built out of the section of the annotation file. The result is a list of name/value pairs similar to `[array get]`. The list is unordered.
- `$anno file` *`<command> <filename>`*—Query the anno object for information about a file referenced by the build.
- `$anno file isdirectory` *`<filename>`*—Return a Boolean indicating whether the specified file is a directory or not.
- `$anno file operations` *`<filename>`*—Return a Tcl list of lists describing the operations that referenced this file in the build.
- `$anno file type` *`<filename>`*—Return the file type for the specified file.

- `$anno filecount`—Retrieve a count of the number of files referenced in Operations in this annotation file.
- `$anno files`—Retrieve a list of the files referenced in Operations in this annotation file.
- `$anno indexagents`—For each agent in the annotation, construct a list of jobs that were run on that agent, sorted in order of time in which they were invoked. This index is stored internally for use by ElectricInsight when rendering the annotation.
- `$anno load` *`<channel> ?count?`*—Parse `?count?` bytes of data from *`<channel>`* and create job objects as necessary. The return value is a Boolean indicating whether parsing is complete. If `?count?` is not specified, all data is read from *`<channel>`*.
- `$anno loadstring` *`<data> ?done?`*—Parse the data in *`<data>`* and create job objects as necessary. There is no return value. If *`?done?`* is specified, it indicates whether the string represents the end of the XML data.
- `$anno jobagent` *`<jobId>`*—Retrieve the name of the agent on which the job ran.
- `$anno jobcommand` *`<jobId>`*—Invoke commands on a particular job from the annotation.
- `$anno job annolength` *`<jobId>`*—Retrieve the length of the segment of the annotation file that is describing the job.
- `$anno job annostart` *`<jobId>`*—Retrieve the start of the segment of the annotation file that is describing the job.
- `$anno job commands` *`<jobId>`*—Retrieve a list of the commands that were associated with the job. The result is a list of lists with the following form.
  - `cmdlist = {[cmd]}`
    cmdlist is a list containing one or more cmd
  - `cmd = {lines argv [output]}`
    cmd is a list containing lines, argv, and one or more output
    lines is the actual line(s) from the makefile from where the command was issued, for example “11-12”
    argv is the actual command issued to the system
  - `output = {src out}`
    output is a list containing src and out
    src is the source of the output (for example, “make”)
    out is the actual string emitted by the component
- `$anno job conflict` *`<jobId>`*—Return registry conflict information for the specified job.
- `$anno job conflictfile` *`<jobId>`*—If the job is a conflict job, retrieve the name of the file that it conflicted over.
- `$anno job conflicttype` *`<jobId>`*—If the job is a conflict job, retrieve the type of the conflict.
- `$anno job deps` *`<jobId>`*—Retrieve the list of history dependencies for the job. This might be empty if annotation did not include history-level detail or if there were no history dependencies.
- `$anno job environment` *`<jobId>`*—Retrieve the environment for the job.
- `$anno job exitcode` *`<jobId>`*—Retrieve the exit code for the job.
- `$anno job finish` *`<jobId>`*—Retrieve the end time of the job in seconds.
- `$anno job flags` *`<jobId>`*—Retrieve the flags for the job as a Tcl list.
- `$annojob isconflict` *`<jobId>`*—Return a Boolean indicating whether the job is a conflict job.

- `$anno job isrerun` *`<jobId>`*—Return a Boolean indicating whether the job is a rerun job.
- `$anno job isreverted` *`<jobId>`*—Return a Boolean indicating whether the job is a reverted job.
- `$anno job jobcache diffs` *`<jobId>`* —Retrieve a list of differences that caused a job cache miss. Each entry in the list is itself a list that gives the name of the file, the original state, the new state, and whether the differences were ignored by eMake.
- `$anno job jobcache length` *`<jobId>`* —Retrieve the duration of the original uncached run of the job.
- `$anno job jobcache status` *`<jobId>`* —Retrieve the job cache status, one of “hit”, “miss”, “newslot”, “rootschanged”, “uncacheable”, or “unneeded”.
- `$anno job jobcache type` *`<jobId>`* —Retrieve the type of job caching applied to the job, such as “gcc” or “parse”.
- `$anno job length` *`<jobId>`* —Retrieve the duration of the job in seconds. This is equivalent to `[$anno job jobId finish] - [$anno job jobId start]`, but it avoids using Tcl's `[expr]` command, so it is much faster.
- `$anno job make` *`<jobId>`*—Retrieve the Make instance ID for the Make containing the specified job.
- `$anno job makefile` *`<jobId>`*—Retrieve the name of the makefile containing the rule that produced the job, if any.
- `$anno job name` *`<jobId>`*—Retrieve the name for the specified job.
- `$annojob neededby` *`<jobId>`*—Retrieve the ID of the job that caused the job to be run.
- `$anno job operations` *`<jobId>`*—Retrieve the list of Operations performed by the job, in order, as a Tcl list of lists.
- `$anno job operations -type registry` *`<jobId>`*—Retrieve the list of registry operations for the specified job.
- `$anno job partof` *`<jobId>`*—Retrieve the ID of the job that the job continues. This is valid only for continuation jobs; other jobs will return an empty string.
- `$anno job rerunjob` *`<jobId>`*—For conflict jobs, retrieve the job that was used to rerun the job.
- `$annojob serialorder` *`<jobId>`*—Retrieve the serial order of the job relative to the other jobs in the build. The first job in the build has serial order 1; the final job in the build has serial order N for a build with N jobs.
- `$anno job start` *`<jobId>`*—Retrieve the start time of the job in seconds.
- `$anno job submakes` *`<jobId>`*—Retrieve the list of submakes performed by the specified job.
- `$anno job timing` *`<jobId>`*—Retrieve the full timing information for the job as a list of lists of the form `{start finish agent} {start finish agent} ....` Normally there is only one entry in the list; if the job was restarted because of cluster sharing or agent failure, there are additional entries. Jobs that never ran will return an empty list.
- `$anno job type` *`<jobId>`*—Retrieve the type of the specified job.
- `$anno job waitingjobs` *`<jobId>`*—Retrieve the list of jobs that waited for the job to complete before running. This might be empty if there are no waiting jobs or if the annotation file did not include waitingJobs annotation.
- `$anno job writejob` *`<jobId>`*—For conflict jobs, retrieve the job that wrote the file that the job conflicted over.

- `$anno jobcount`—Return a count of the number of jobs in the annotation.
- `$anno jobiterbegin`—(Deprecated as of version 5.0) Initialize the job iterator to the head of the list of jobs in the anno object.
- `$anno jobitermore`—(Deprecated as of version 5.0) Return a Boolean indicating whether the next call to `[$anno jobiternext]` will return a valid job or not. This is used in conjunction with `[anno jobiterbegin]` and `[anno jobiternext]` to efficiently iterate through the list of jobs in the anno object (in serial order).
- `$anno jobiternext`—(Deprecated as of version 5.0) Retrieve the job with the next highest serial order in the anno object using an iterator initialized with `[anno jobiterbegin]`. If the iterator has reached the end of the list of jobs, an empty string is returned. Otherwise, the job ID for the next job is returned, and the iterator is advanced one step.
- `$anno jobs begin`—Return an iterator for the first job in the build.
- `$anno jobs end`—Return an iterator referring to the “past-the-end” job in the build—not an actual job, but a hypothetical placeholder for iteration.
- `$anno jobs next` *`<jobId>`*—Return the next job in the build after the job given by *`<jobId>`*.
- `$anno jobs prev` *`<jobId>`*—Return the previous job in the build before the job given by *`<jobId>`*.
- `$anno jobs rbegin`—Return an iterator for the last job in the build.
- `$anno jobs rend`—Return an iterator referring to the “before-the-first” job in the build—not an actual job, but a hypothetical placeholder for iteration.
- `$anno jobsearch` *`<attribute pattern>`*—Search the jobs in the annotation for jobs that match the criteria.
- `$anno make` *`<command> <makeId>`*—Invoke commands on a particular Make from the annotation.
- `$anno make commandline` *`<makeId>`*—Retrieve the command line for the specified Make.
- `$anno make job` *`<makeId>`*—Retrieve the job ID for the job that spawned the given Make instance or an empty string, if no job spawned the Make.
- `$anno make level` *`<makeId>`*—Retrieve the level of the specified Make instance.
- `$anno make mode` *`<makeId>`*—Retrieve the emulation mode of the given Make instance.
- `$anno make workingdir` *`<makeId>`*—Retrieve the working directory for the specified Make instance.
- `$anno makecount`—Return a count of the number of Makes in the annotation.
- `$anno metrics`—Dump the contents of the metrics table, which was built out of the section of the annotation file. The result is a list of name/value pairs similar to `[array get]`. The list is unordered.
- `$anno parseoptions` *`?optionList?`*—Query or set the anno object parse options. This controls which portions of the annotation file are processed when `[anno load]` is invoked.
- `$anno properties`—Dump the contents of the properties table, which was built out of the section of the annotation file. The result is a list of name/value pairs similar to `[array get]`. The list is unordered.
- `$anno refcount`—Retrieve the reference count from the anno object. For testing only.
- `$anno rjobiterbegin`—(Deprecated as of version 5.0) Initialize the job iterator to the end of the list of jobs in the anno object.

- `$ann rjobitermore`—(Deprecated as of version 5.0) Return a Boolean indicating whether the next call to `[anno rjobiternext]` will return a valid job or not. This is used in conjunction with `[anno rjobiterbegin]` and `[anno rjobiternext]` to efficiently iterate through the list of jobs in the anno object, in reverse serial order.
- `$anno rjobiternext`—(Deprecated as of version 5.0) Retrieve the job with the next lowest serial order in the anno object using an iterator initialized with `[anno rjobiterbegin]`. If the iterator has reached the end of the list of jobs, an empty string is returned. Otherwise, the job ID for the next job is returned, and the iterator is advanced one step.
- `$anno sortjobs` *`?options? <jobList>`*—Sort the jobs in *`<jobList>`* according to the given criteria.

  Note: This modifies the list *in place*.
- `$anno type` *`<command> <type>`*—Query the aggregate attributes of jobs by type.
- `$anno type conflicttime` *`<type>`*—Return the amount of time spent on jobs of this type that were later found to be in conflict.
- `$anno type jobcount` *`<type>`*—Return the number of jobs of this type.
- `$anno type reruntime` *`<type>`*—Return the amount of time spent on jobs of this type that were rerun jobs.
- `$anno type revertedtime` *`<type>`*—Return the amount of time spent on jobs of this type that were later reverted.
- `$anno type time` *`<type>`*—Return the amount of time spent on jobs of this type that were not conflict, reverted, or rerun jobs.
- `$anno types`—Retrieve a list of known types of jobs.