



Electric Insight

Users Guide

version 3.2

Electric Cloud, Inc.
676 W. Maude Avenue
Sunnyvale, CA 94085
www.electric-cloud.com

Copyright © 2005 - 2009 Electric Cloud, Inc. All rights reserved.

Published October 2009

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

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

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

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

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

Trademarks

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

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

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

Contents

Chapter 1	ElectricInsight Introduction	
	About ElectricInsight	1-1
Chapter 2	Getting Started	
	Annotation.....	2-1
	Annotation File Splitting.....	2-1
	Starting ElectricInsight	2-2
	Navigating ElectricInsight	2-3
	Agents & Jobs	2-3
	Navigation.....	2-4
	Overview	2-5
	Legend.....	2-6
	Job Summary	2-7
	Replay Build	2-8
	Dependencies and Waiting Jobs Popup	2-9
	Zoom.....	2-9
Chapter 3	Build, Job, and Make Information	
	Build Properties	3-1
	Job Details.....	3-2
	Job Details Tab	3-2
	Job Path Tab.....	3-3
	Environment Tab	3-3
	Annotation Tab	3-4
	Submakes	3-5
	Viewing All Jobs.....	3-5
	Make Details	3-6
	Make Details Tab.....	3-6
	Make Path Tab	3-7
	Make Jobs Tab	3-7
	Searching for Jobs.....	3-8
	Supported Search Fields	3-9
Chapter 4	ElectricInsight Reports	
	About Reports	4-1
	Build manifest.....	4-2
	Cluster sharing	4-3
	Derived files analysis	4-4
	ElectricSimulator	4-5
	Files modified multiple times	4-6
	Job count by length	4-7

	Job time by length.....	4-8
	Job time by type.....	4-9
	Jobs by agent	4-10
	Jobs by file.....	4-10
	Longest jobs.....	4-11
	Longest serial chain.....	4-12
	Makefile manifest.....	4-13
	Root conflicts.....	4-14
	Serialization analysis	4-15
	Creating Custom Reports.....	4-16
Chapter 5	Using ElectricInsight to Increase Build Speed	
	Identifying Build Issues.....	5-1
Appendix A	Annolib Programmer's Reference	
	Annolib	A-1

ElectricInsight 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® removes the “black box” around software product builds and provides easy-to-understand performance data.

About ElectricInsight

ElectricInsight 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®, ElectricInsight mines information produced by Electric Make® to provide an easy-to-understand, graphical representation of the build structure for performance analysis.

ElectricInsight 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. ElectricInsight 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?

Annotation

ElectricInsight gathers information from build annotation files to create a picture of your build. To use ElectricInsight, you must run your builds with the `--emake-annodetail` parameter. For sophisticated analysis of builds, run Electric Make (eMake) with `file`, `history`, and `waiting` level annotation.

Set annotation levels using custom build classes or the eMake command line:

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

Collecting annotation information may cause your build to run approximately 5% slower.

Supported annotation detail levels:

<code>basic</code>	detailed information about each “job” in the build, including command arguments, output, exit code, timing, and source location
<code>env</code>	environment variable modification information (ElectricAccelerator 4.3 and later)
<code>file</code>	filesystem operations information, excluding lookups
<code>history</code>	information about implicit dependencies discovered by Electric Make
<code>lookup</code>	filesystem operations information, including lookups
<code>registry</code>	registry operations information (ElectricAccelerator 4.3 and later)
<code>waiting</code>	information required to reconstruct the complete dependency graph

Note: For additional information about annotation, see the “Annotation” chapter in the *Electric Cloud Electric Make Users Guide*.

Annotation File Splitting

Due to a gcc file size write limit, annotation file size is limited to 1.6 GB. Large annotation files are split to remain under this limit. This limit is not configurable.

ElectricInsight requires a single annotation file. To rejoin split annotation files:

On Linux:

```
cat file1 > result_file
cat file2 >> result_file
```

On Windows:

```
type file1 > result_file  
type file2 >> result_file
```

Note: Though there is no predetermined limit at which an annotation file cannot be loaded by ElectricInsight, the maximum size is limited by the size of the process because ElectricInsight is a 32-bit application.

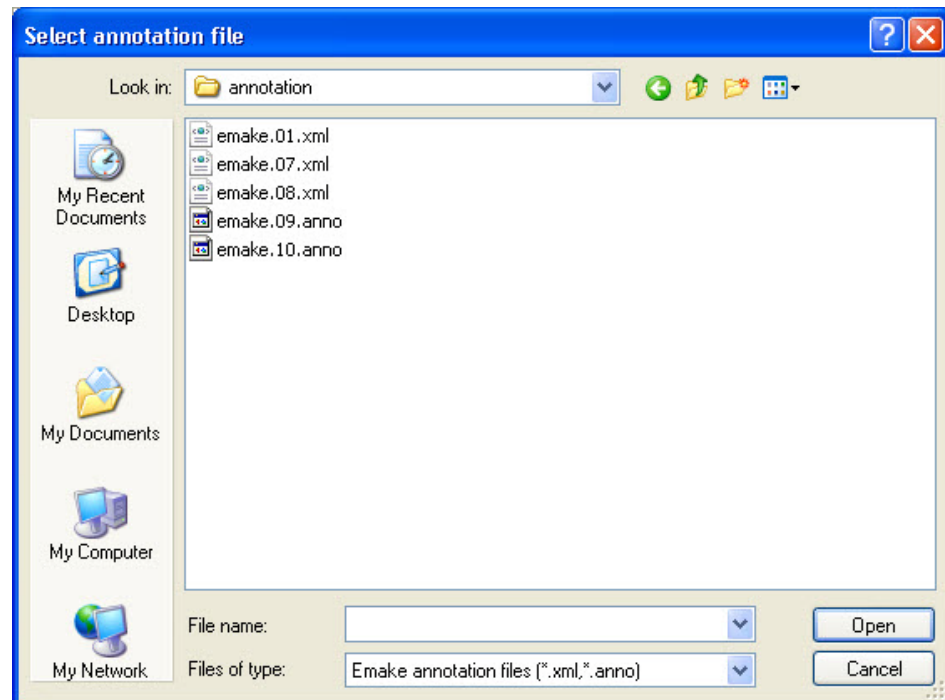
Starting ElectricInsight

To start ElectricInsight, go to Start > All Programs > Electric Cloud > ElectricInsight, or type “einsight” at the command prompt. You can specify the target build annotation file if you want.

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

When ElectricInsight starts, it displays your build information.

If you did not specify a build annotation file on the command line, browse for one to load. Click the folder icon on the toolbar or select File > Load annotation.



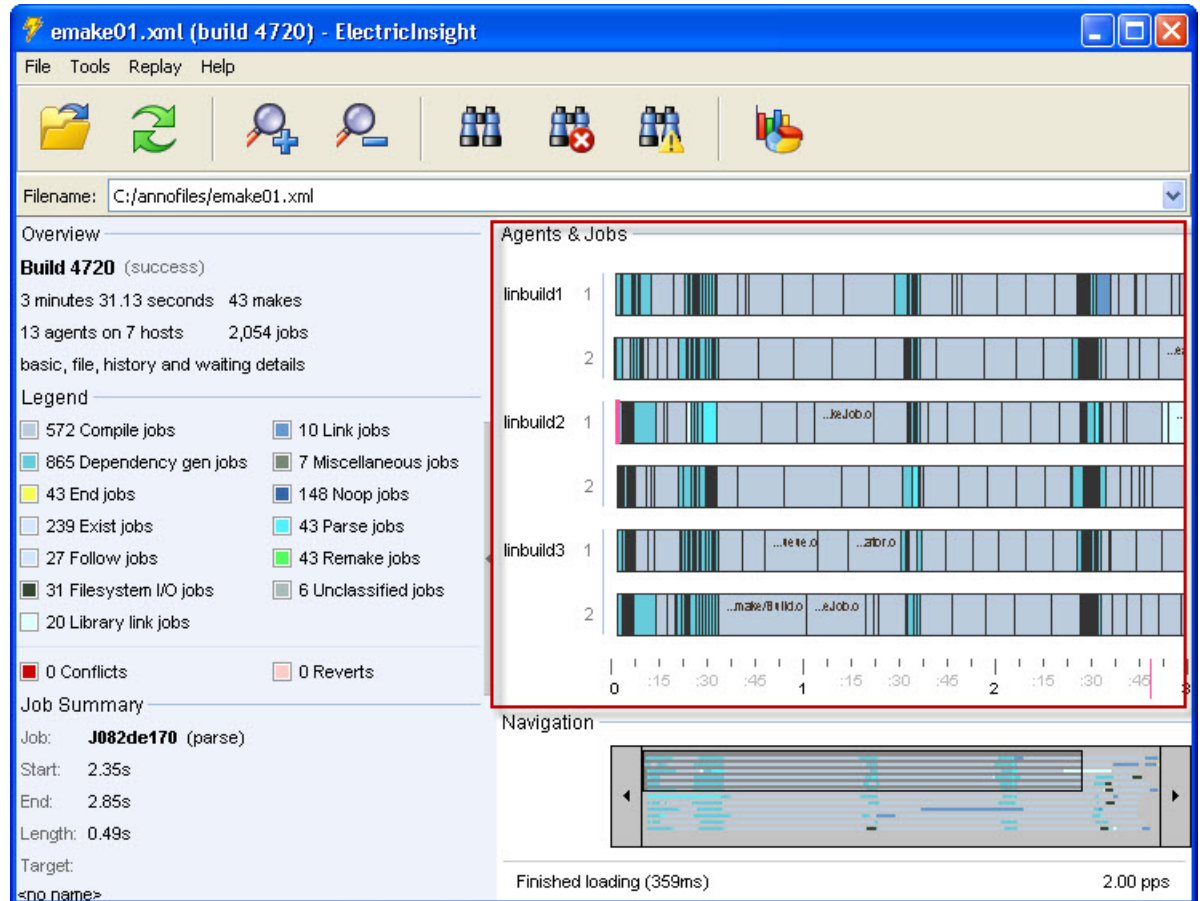
Navigate to the annotation file of a completed build you want to analyze and click **Open**. This displays your build.

Navigating ElectricInsight

The following sections will familiarize you with the ElectricInsight interface.

Agents & Jobs

In the Agents & Jobs section (outlined in the following screenshot), 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.



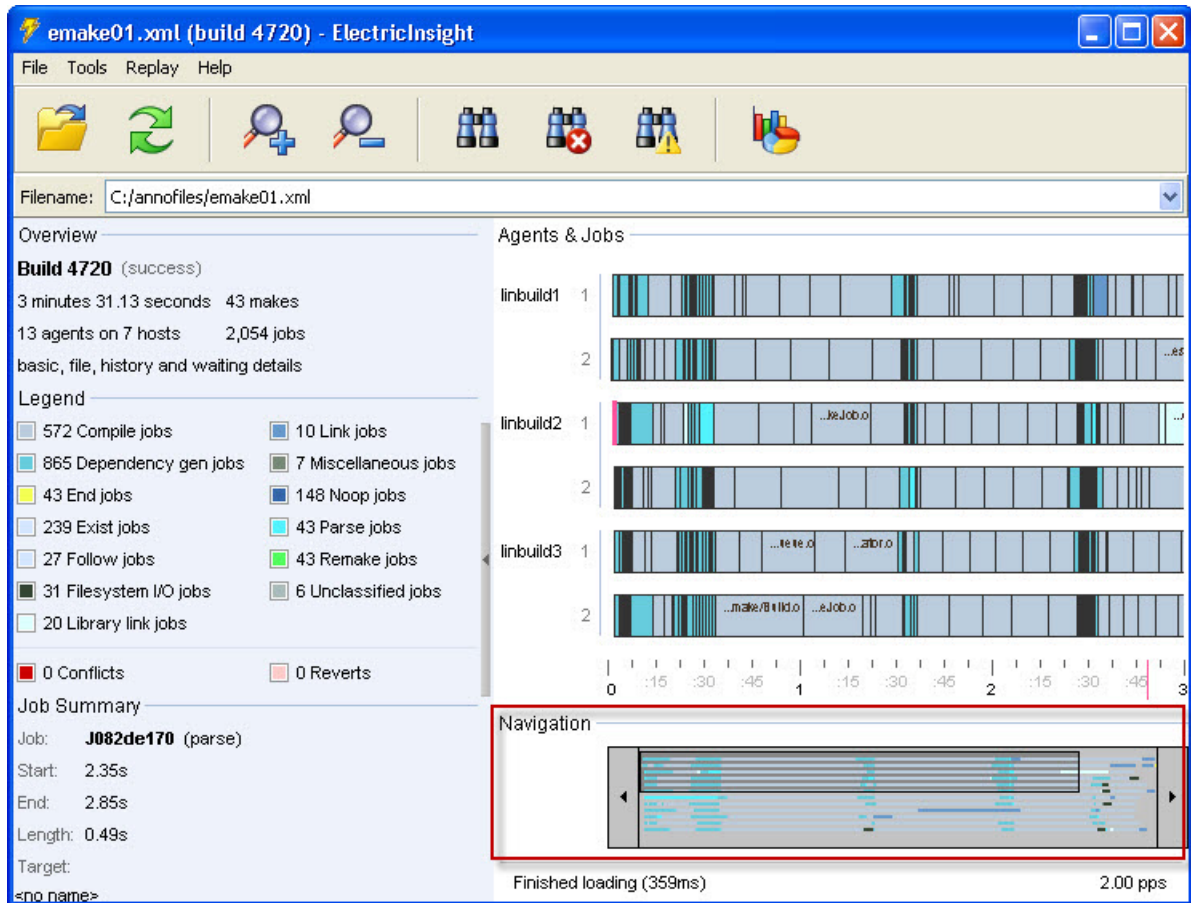
The section's x-axis is time, with the build starting at the left. The first job completed by each agent is at the extreme left of an agent's job bar. As you scroll to the right on an agent's job bar, you see the progress of jobs assigned to the agent.

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.

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.

Navigation

In the Navigation section, the view frame indicates which part of the build is displayed. 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.

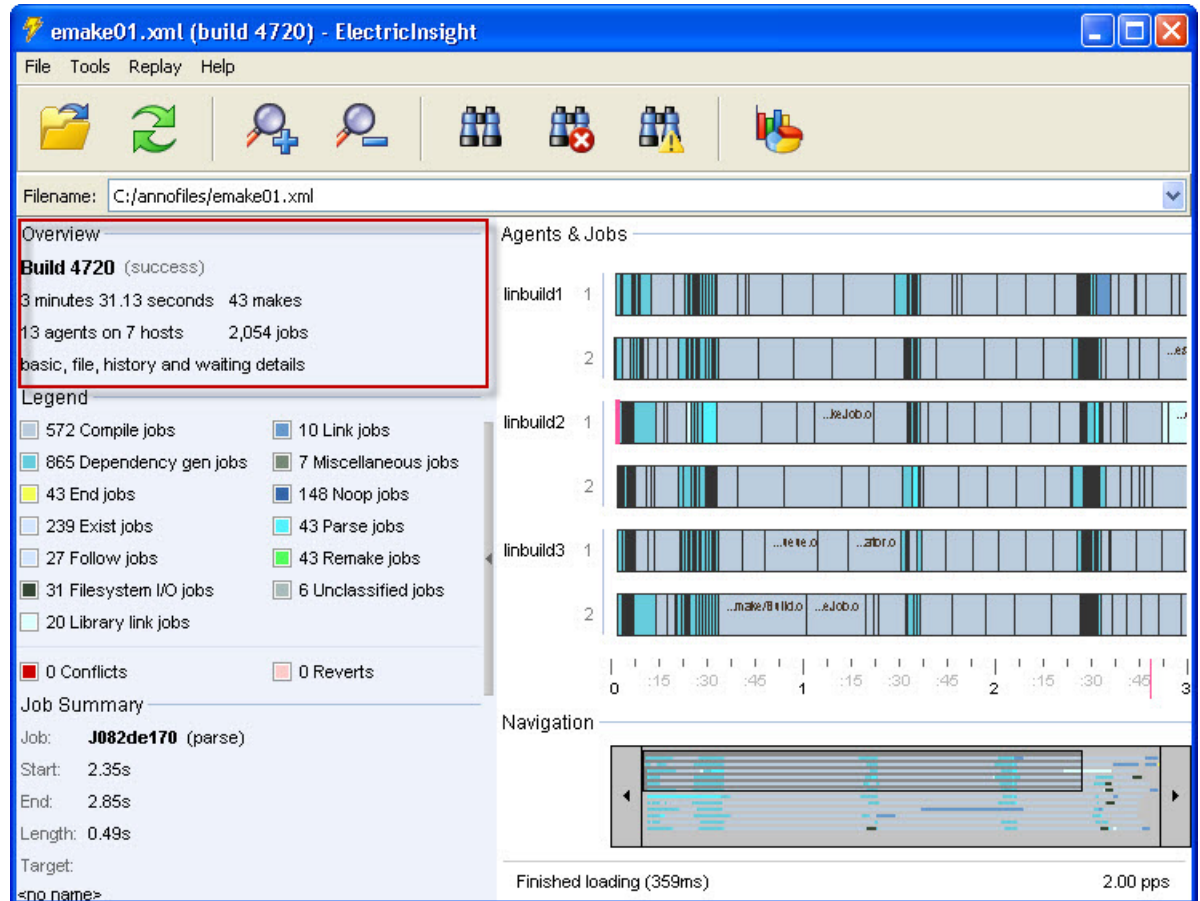


Below the Navigation section is a field showing how long it took ElectricInsight to load build annotation information.

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



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.
- Also, 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”](#) on page 3-9.

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

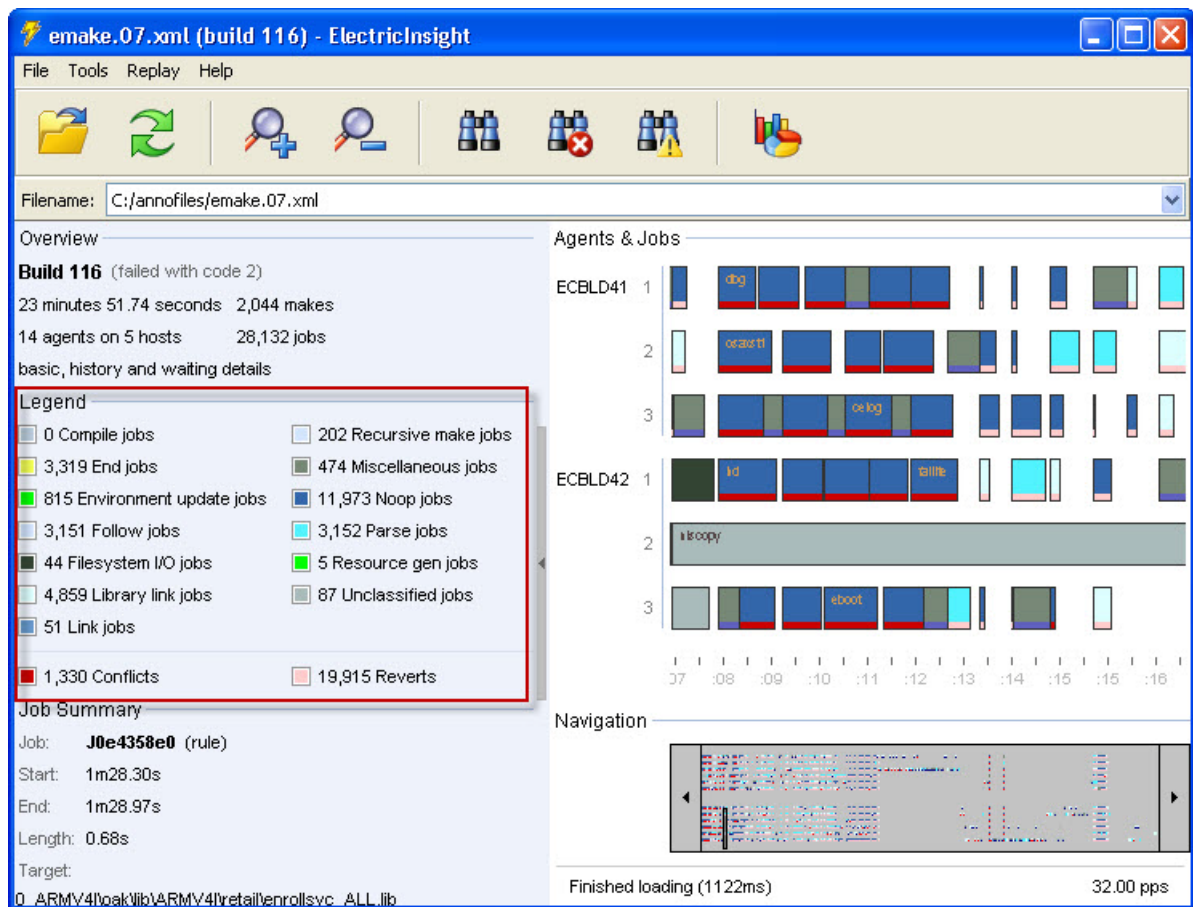
Legend

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.

The Agents & Job section displays two dimensions of job data—job status and job type. If the job status is not unusual (for example, the job was scheduled and ran normally), then the job type (such as parse or rule) only 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.

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 may 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.

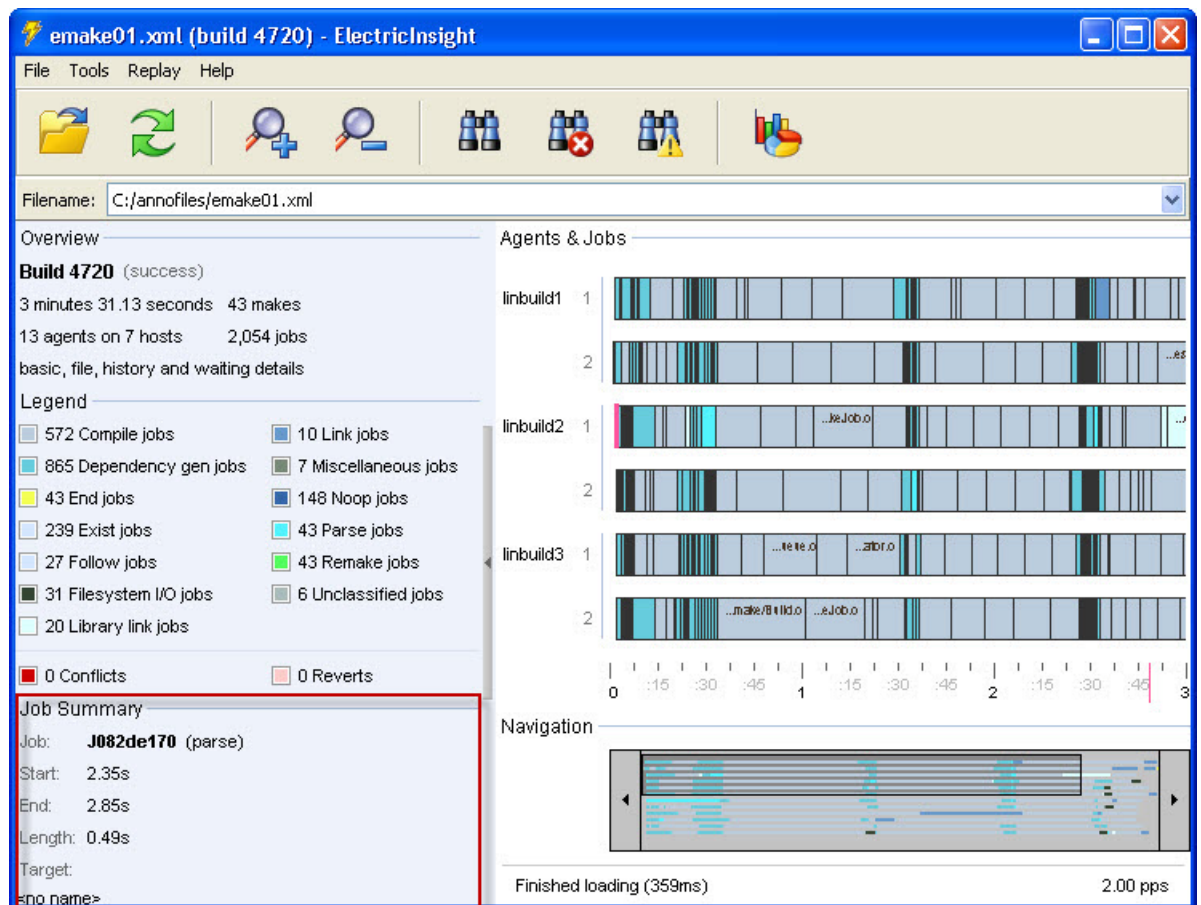


Job Summary

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.

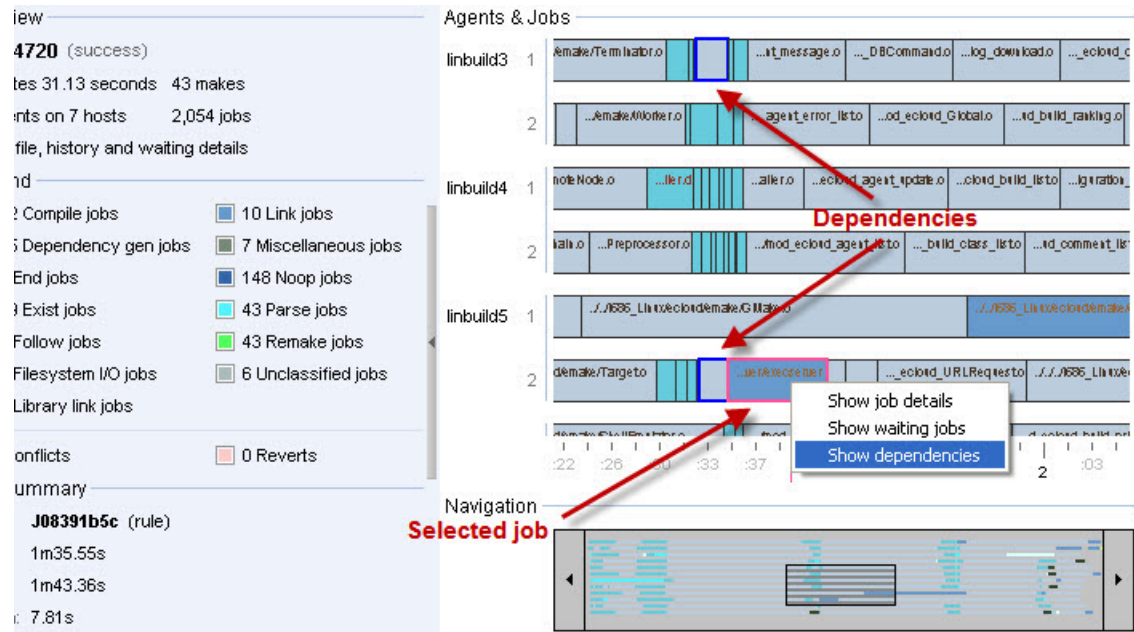
The Job Summary section displays the following information about the selected job:

- Job ID and job type or status (for additional information about job types and status, see [“Supported Search Fields” on page 3-9](#))
- Start time (as an offset from the build start time)
- End time
- Job length in hours/minutes/seconds
- Output target name



Dependencies and Waiting Jobs Popup

In the Agents & Jobs section, right-clicking a job that has dependencies or waiting jobs displays a popup menu. You can select **Show dependencies** or **Show waiting jobs**. Dependencies are outlined in blue. Waiting jobs (not shown in the screenshot) are outlined in red.



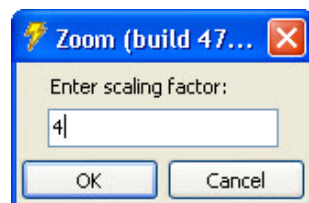
To find dependencies or waiting jobs, load an annotation file containing waiting and history level annotation and run the Serialization analysis report.

Zoom

The Tools menu contains the following magnification options:

- Zoom in
- Zoom out
- Zoom to...
- Zoom to fit

Zoom to... - this option allows you to specify a magnification factor for the display. To return to the initial, unzoomed view, select Tools > Zoom to... and type a zoom factor of 1.0.

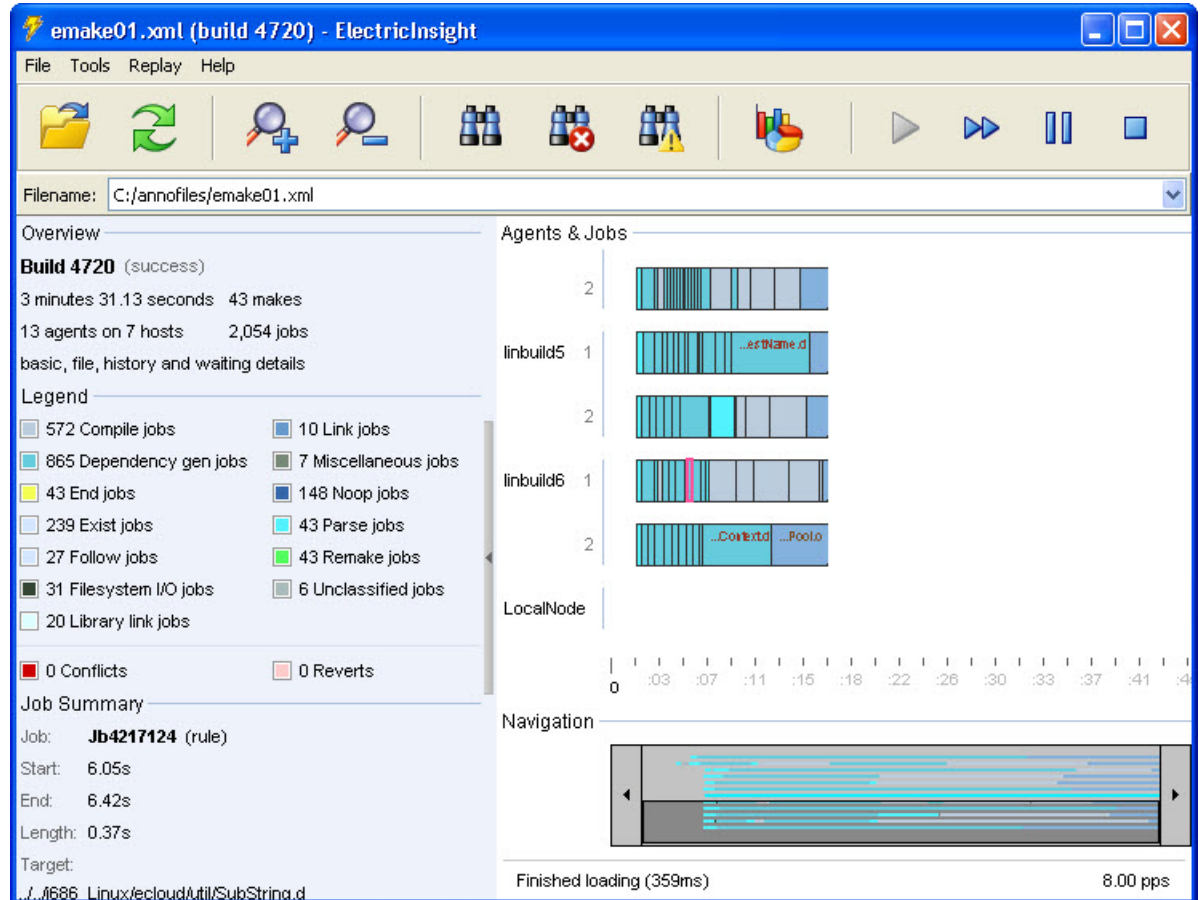


Zoom to fit - this option sizes the display to fit the available space. Doing this may leave empty space to the left of all agent job bars, which depicts the time when eMake was parsing the makefile, before agents were assigned jobs. It is also possible that agents you are not currently viewing were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

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].

Replay Build

ElectricInsight enables you to replay your build in the Agents & Jobs section, allowing you to observe its progress. You can display the replay toolbar by clicking **Replay > Show toolbar** in the main menu. The following screenshot shows the replay toolbar and a build replay in progress.

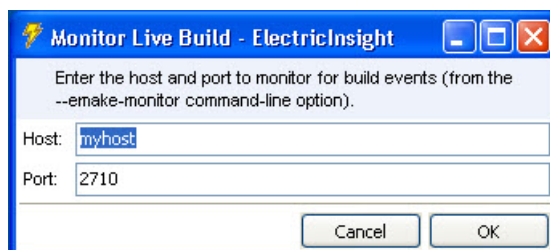


Replay controls:

- 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

ElectricInsight enables you to monitor a live build in the Agents & Jobs section. Select File > Monitor live build in the main menu to display the following dialog.

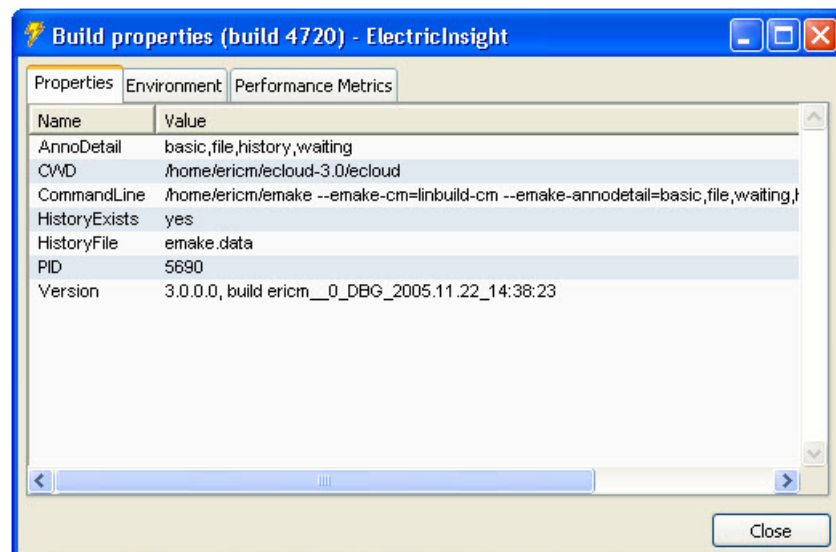


Provide the host and port information from the `--emake-monitor` command-line option and click **OK**.

Build, Job, and Make Information

Build Properties

To view overall build properties, click File > Build Properties. There are three tabs: Properties, Environment, and Performance Metrics.



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

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



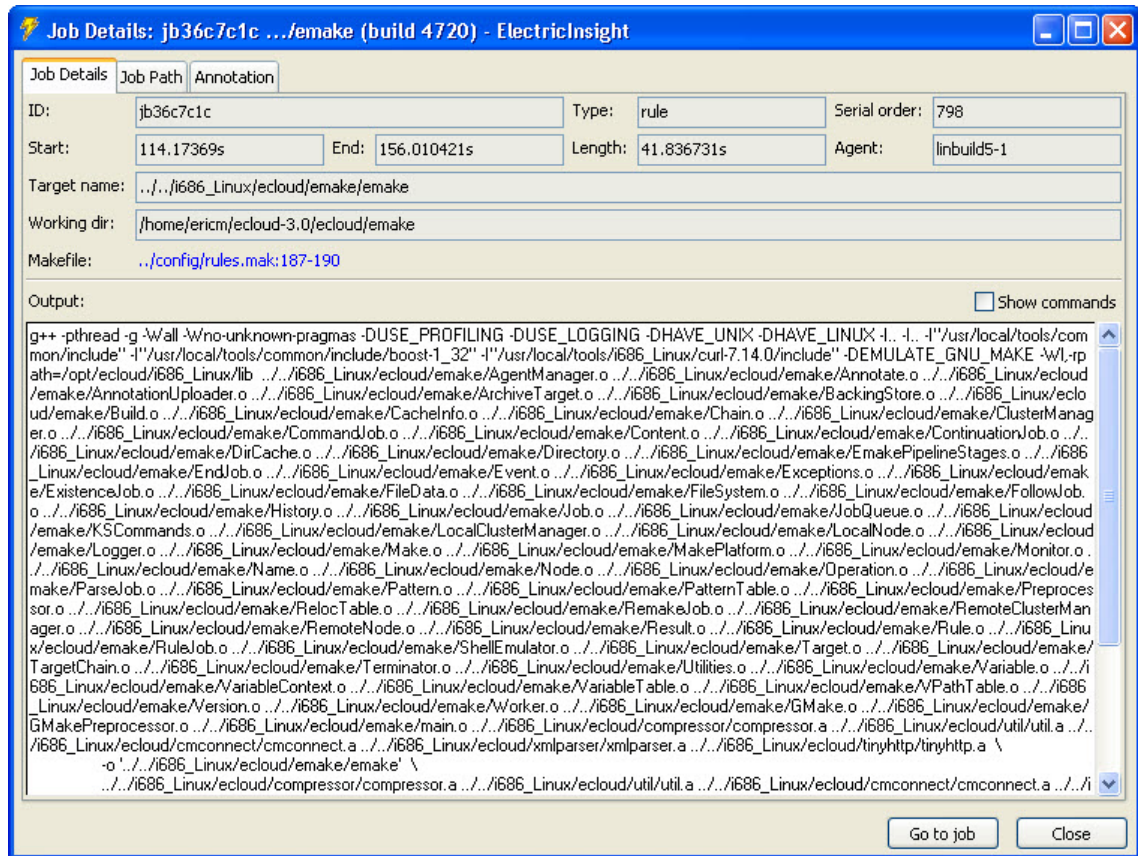
Job Details

This section provides additional information about jobs and the Job Details dialog.

To open the Job Details dialog, double-click a job in an agent job bar, or select a job and press CTRL-Enter. The Job Details tab is displayed by default.

Job Details Tab

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



For some types of jobs, the **Target name**, **Makefile**, and **Output** fields are populated. When the **Output** field is filled in, you can enable **Show commands** to see the makefile line(s) that correspond to the job.

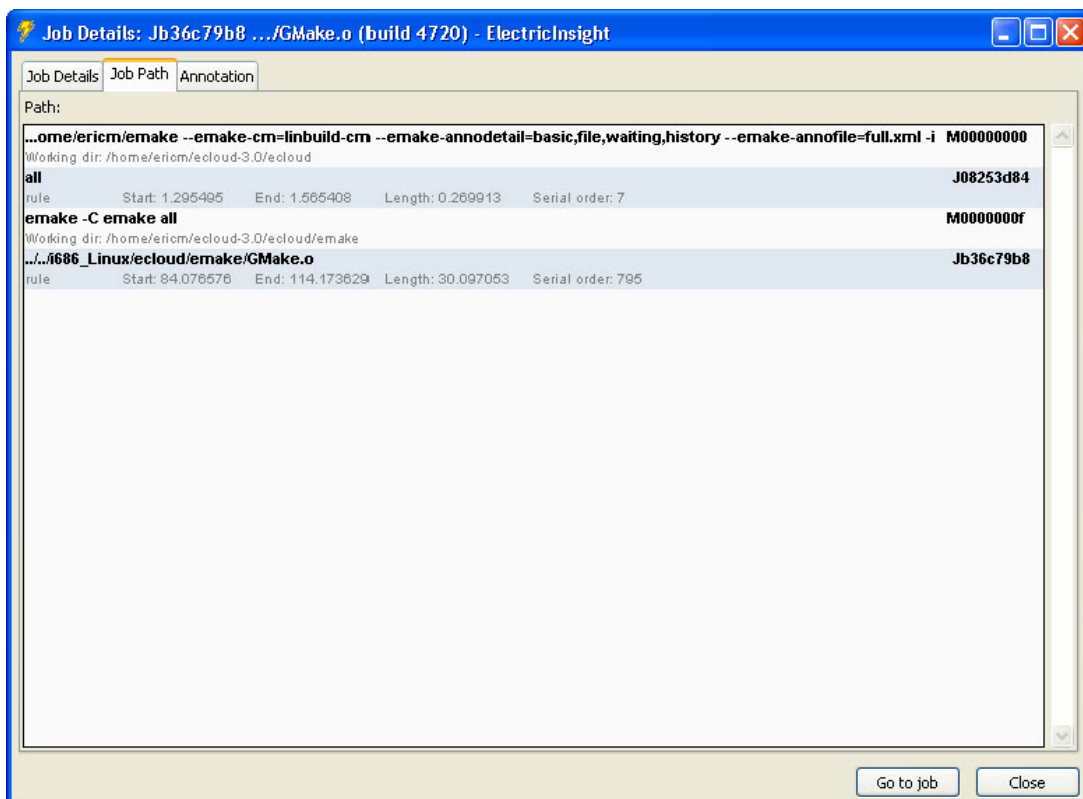
The **Makefile** field identifies the makefile and the relevant line in that makefile that created the job. You can click the makefile to open it in an editor. The **Makefile** field is empty for parse-type jobs because parse jobs are not created from a line in a makefile; for example, there is no rule 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.

For a job that started a recursive make, the Job Details dialog contains a Submakes tab listing recursive makes arising from the job. Double-click an entry in the list to display the job's detail dialog.

To search the Output field, press CTRL-F. CTRL-G finds the next occurrence.

Job Path Tab

The Job Path tab displays the path to the makefile line(s) that created the job. The first line contains the command invoking 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.



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.

Environment Tab

The Environment tab is available only if your annotation file includes env level annotation.

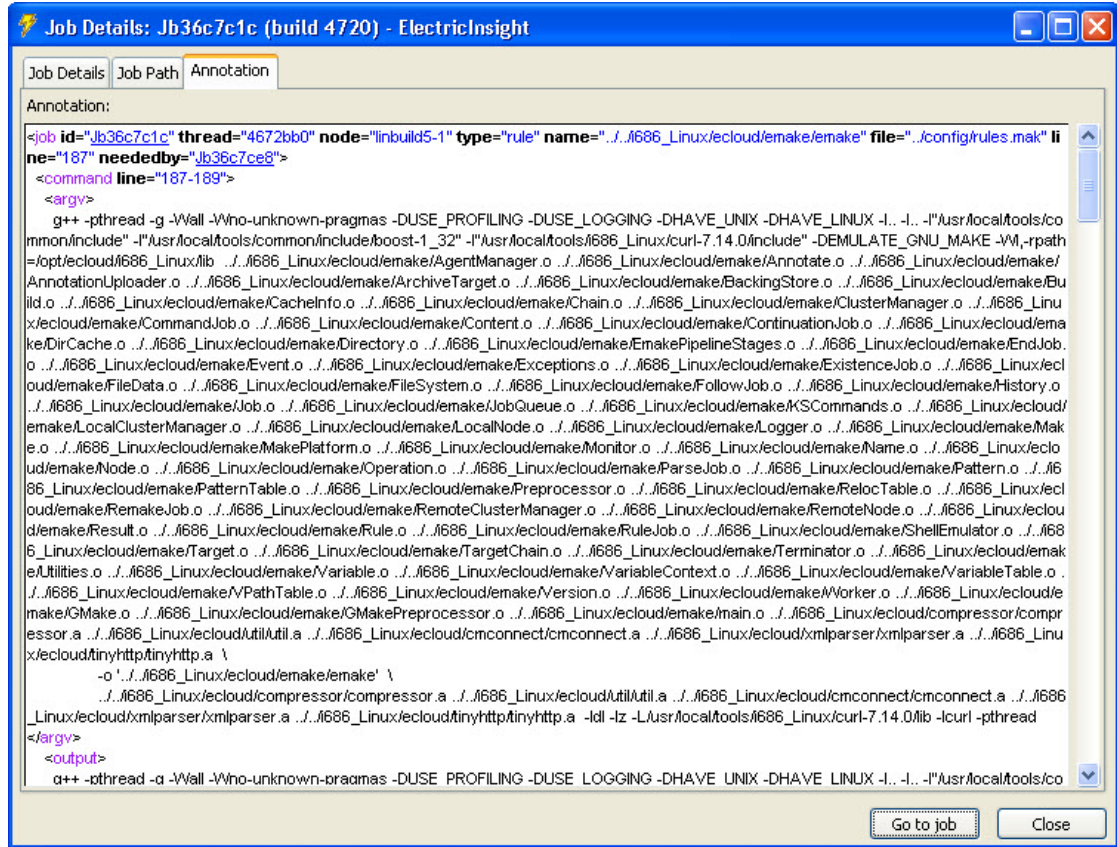
This tab displays environment information that was used to run the job. You can copy this information to the clipboard:

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

Annotation Tab

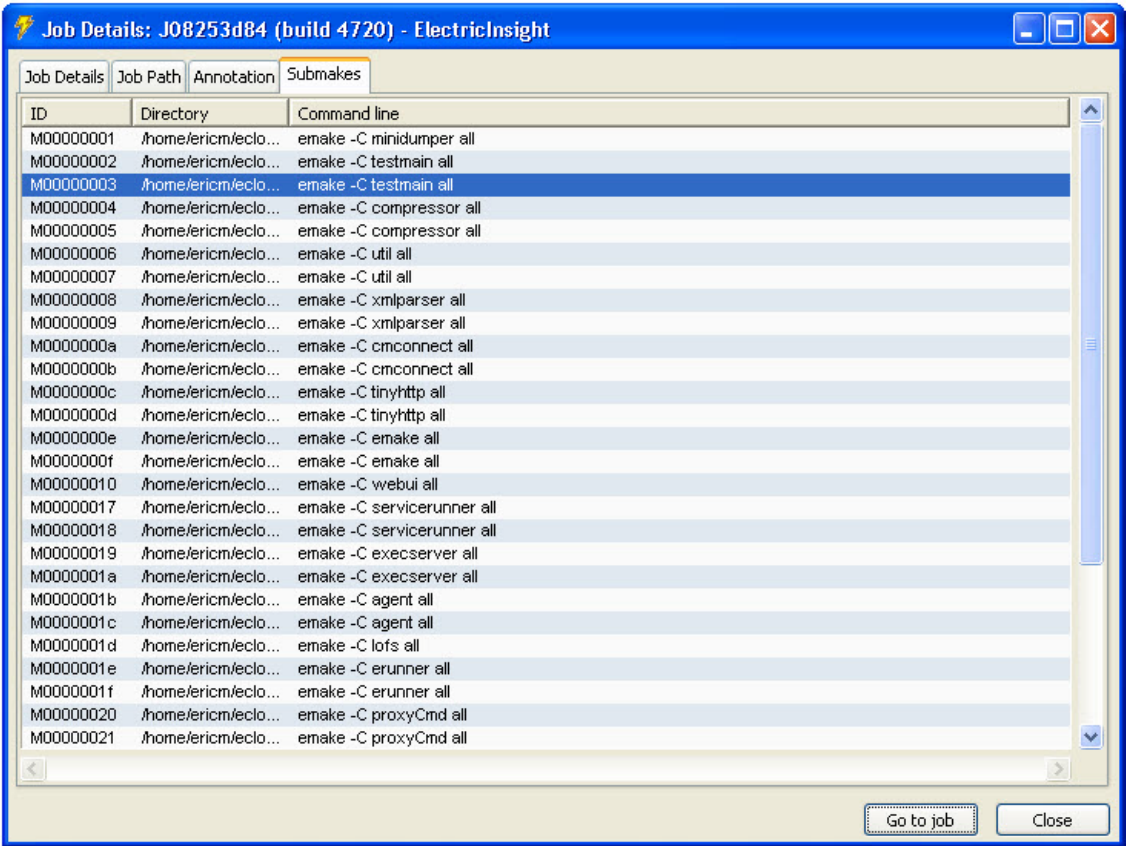
The Annotation tab displays the portion of the annotation file pertaining to the job.

Press CTRL-F to search for and highlight text. CTRL-G finds the next occurrence.



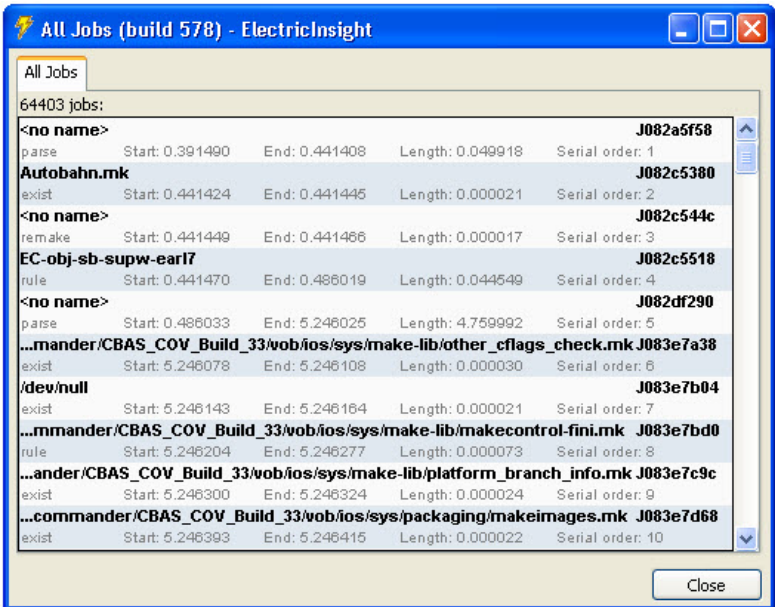
Submakes

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



Viewing All Jobs

To list all of the build's jobs from the main ElectricInsight window, select Tools > View all jobs....



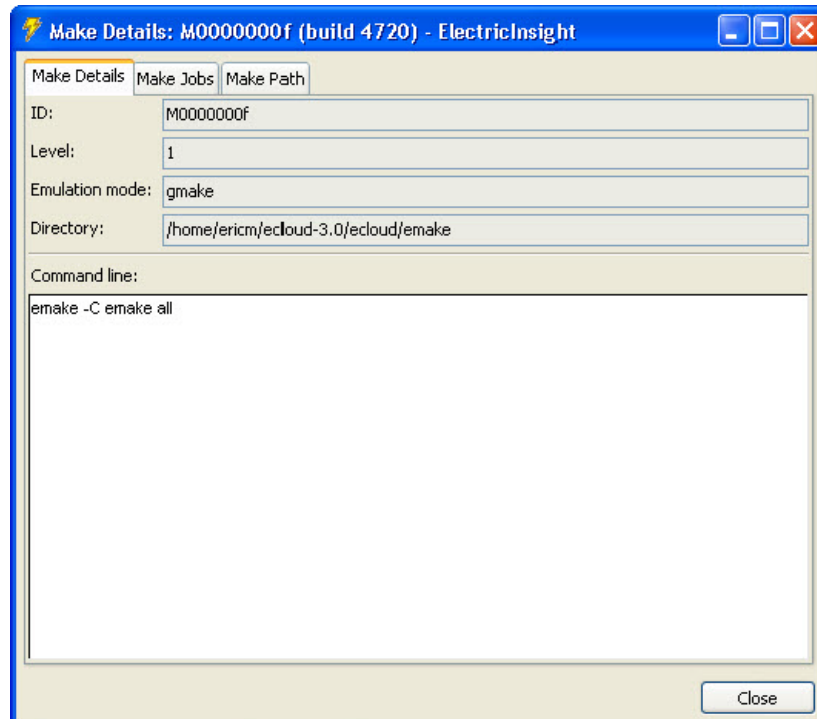
Make Details

This section provides additional information about the 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. The Make Details tab is displayed by default.

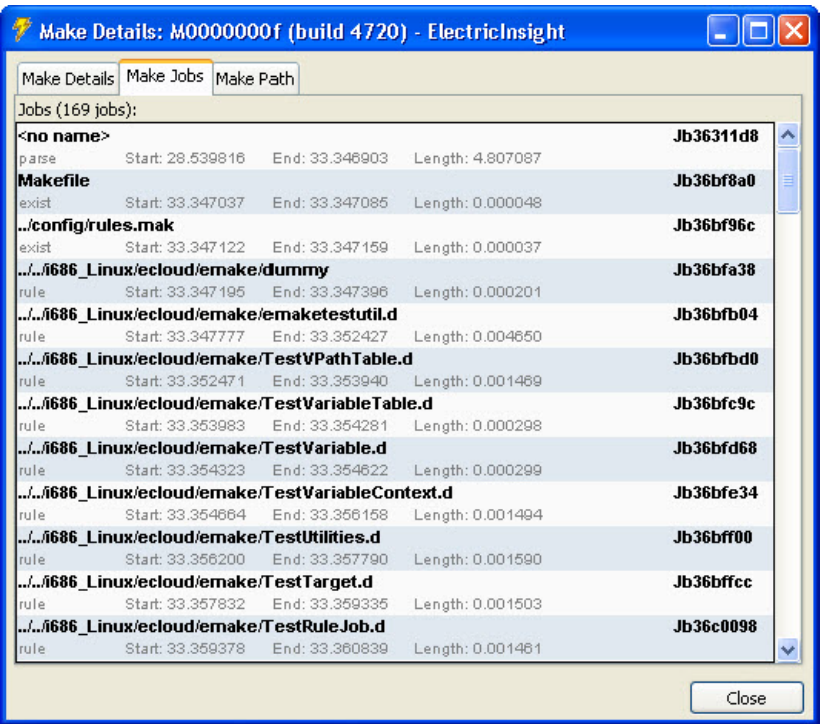
Make Details Tab

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



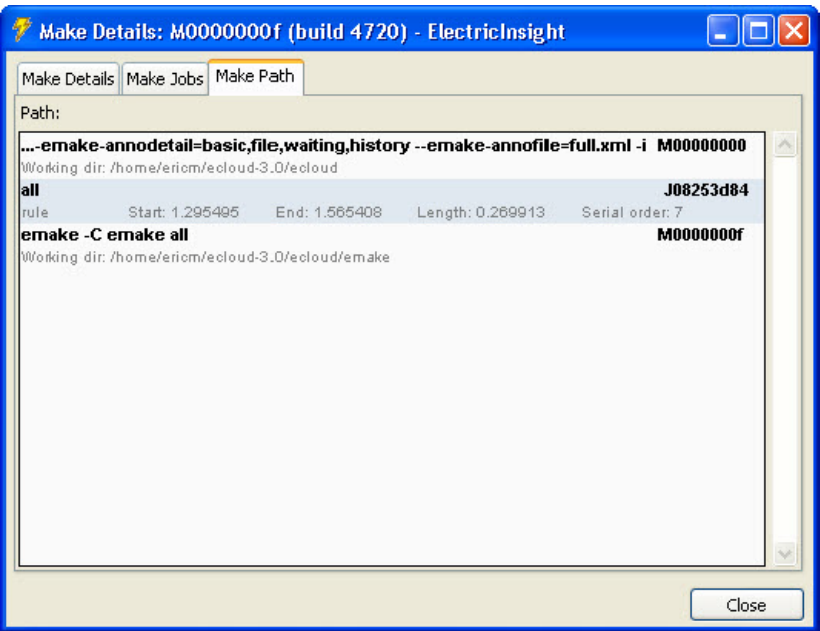
Make Jobs Tab

The Make Jobs tab lists all jobs that the makefile runs.



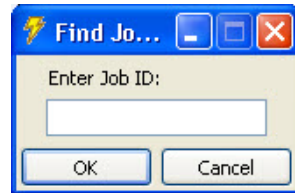
Make Path Tab

The Make Path tab displays the path to the makefile line(s) that created the make. The first line contains the command invoking 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.

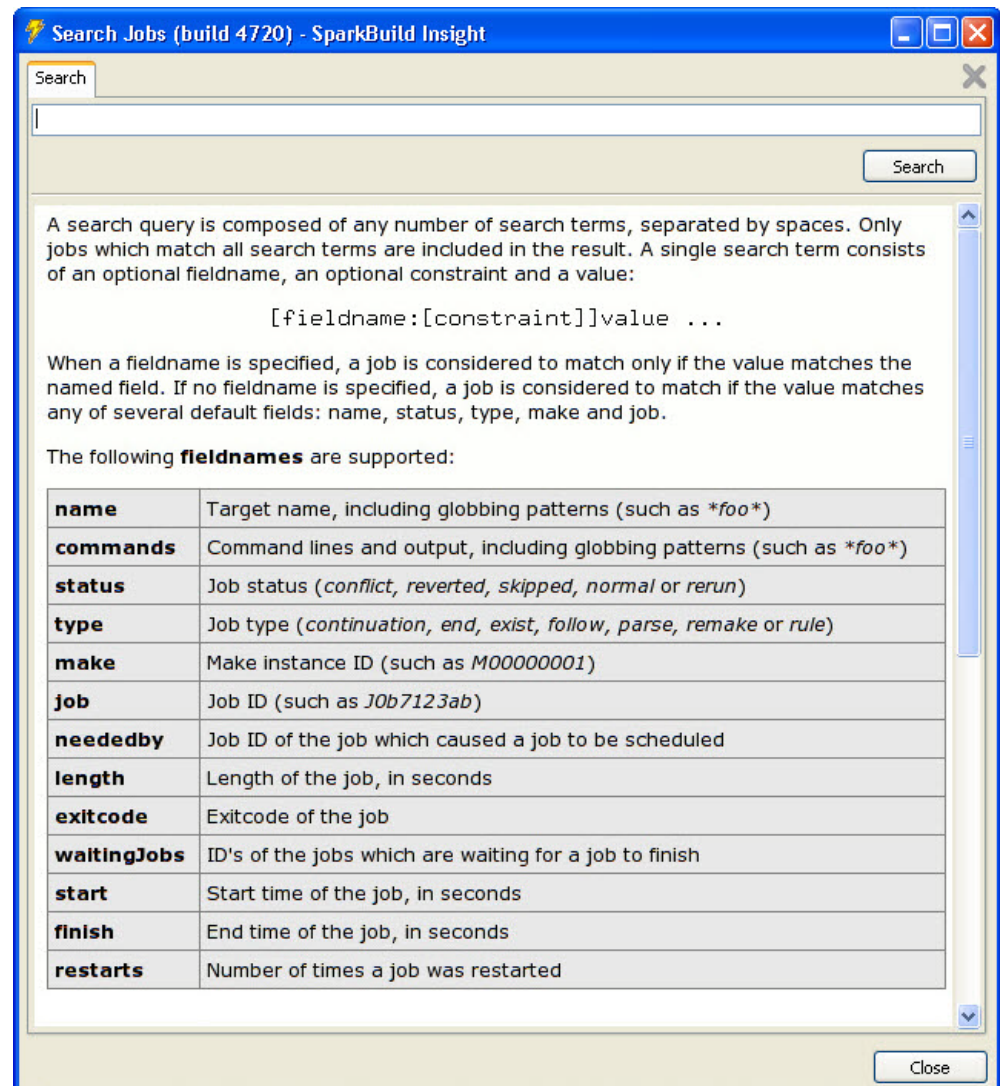


Searching for Jobs

If you know the job ID, use Tools > Find job by ID to go to job details.



If you do not know the job ID, you can search for jobs using several criteria. Select Tools > Search jobs, or click the binoculars icon on the toolbar to display the search dialog.



Type-in the strings you want to search for (for example, “gcc conflict”) and click **Search**. The values come from fields in the build annotation file. You can constrain your search by searching for specific values in specific fields.

To sort your results, select an option from the drop-down. Double-click a row to display a job’s Job Details dialog. In the dialog’s lower-right corner, click **Go to job**. The job is now highlighted in the Agents & Jobs section.

If you want to view all jobs and do not want to search, select Tools > View all jobs to display all build jobs sorted in serial order.

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 may 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 allows you to 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 get 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.

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

Search Examples

To find rule jobs involved in conflicts, search for

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

To find conflicts involving jobs named “export” search for

```
conflict name:export
```

To find rerun jobs with names including “port” search

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

To search for non-reverted jobs with names ending in “Makefile” search

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

About Reports

To display the Reports dialog, select Tools > Reports or click the reports icon on the toolbar. You can open one report at a time only for each ElectricInsight instance, but you can open several ElectricInsight instances.

You can run most reports by double-clicking the report name. Exceptions are Serialization analysis and ElectricSimulator, which have mandatory parameters.

The following reports are available:

- Build manifest
- Cluster sharing
- Derived files analysis
- ElectricSimulator
- Files modified multiple times
- Job count by length
- Job time by length
- Job time by type
- Jobs by agent
- Jobs by file
- Longest jobs
- Longest serial chain
- Makefile manifest
- Root conflicts
- Serialization analysis

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

Export... Run Command...

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

To run a shell command, click **Run Command**. Type a command and a pathname in the Run Shell Command dialog. The the shell command is run with each file in the filtered list as an argument. Results and errors are displayed under their corresponding tabs.

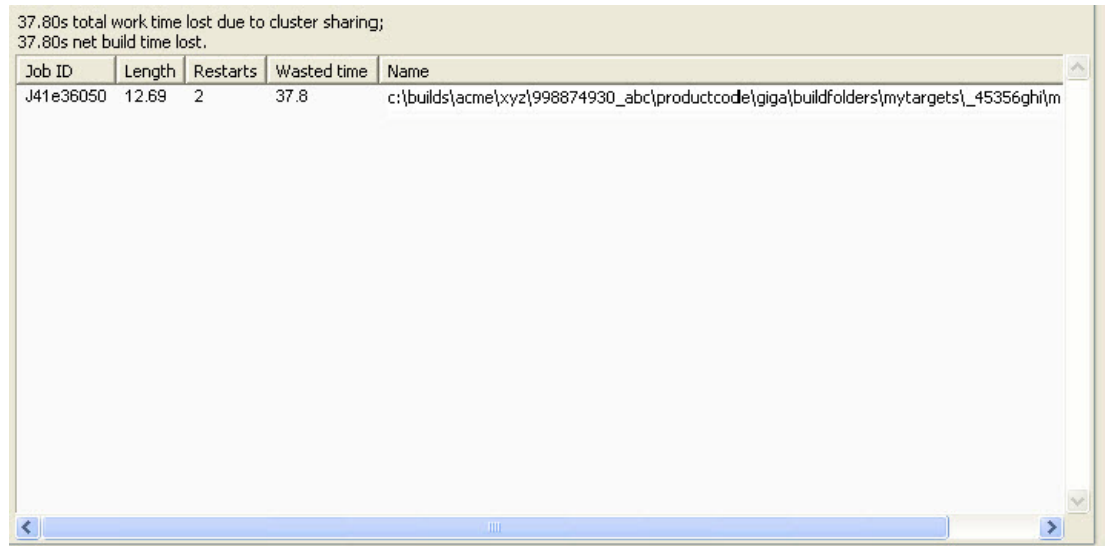
Cluster sharing

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.

For example: job A starts 5 seconds into the build and 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 15 seconds only because the build time lost by B overlaps the build time lost by A.

This report is useful if your build aborts jobs because an agent host is de-allocated from the build. The host may 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.



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

The screenshot shows a report window with a title bar and a scrollable area. The title bar contains the text '37.80s total work time lost due to cluster sharing; 37.80s net build time lost.' Below the title bar is a table with five columns: Job ID, Length, Restarts, Wasted time, and Name. The table contains one row of data for Job ID J41e36050. The Name column contains a long file path. The window has a standard Windows-style scrollbar at the bottom.

Derived files analysis

The Derived files analysis report allows you to 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`

To run the report, enter the file(s) (with full paths) in the input field and click **Run**.

Derived Files Report (5 operations)

Enter input file(s), one per line:

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

Clear Run

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

Copy Report

The middle display area lists the 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 this lower display area to display its Job Details dialog.

Click **Clear** to clear all fields.

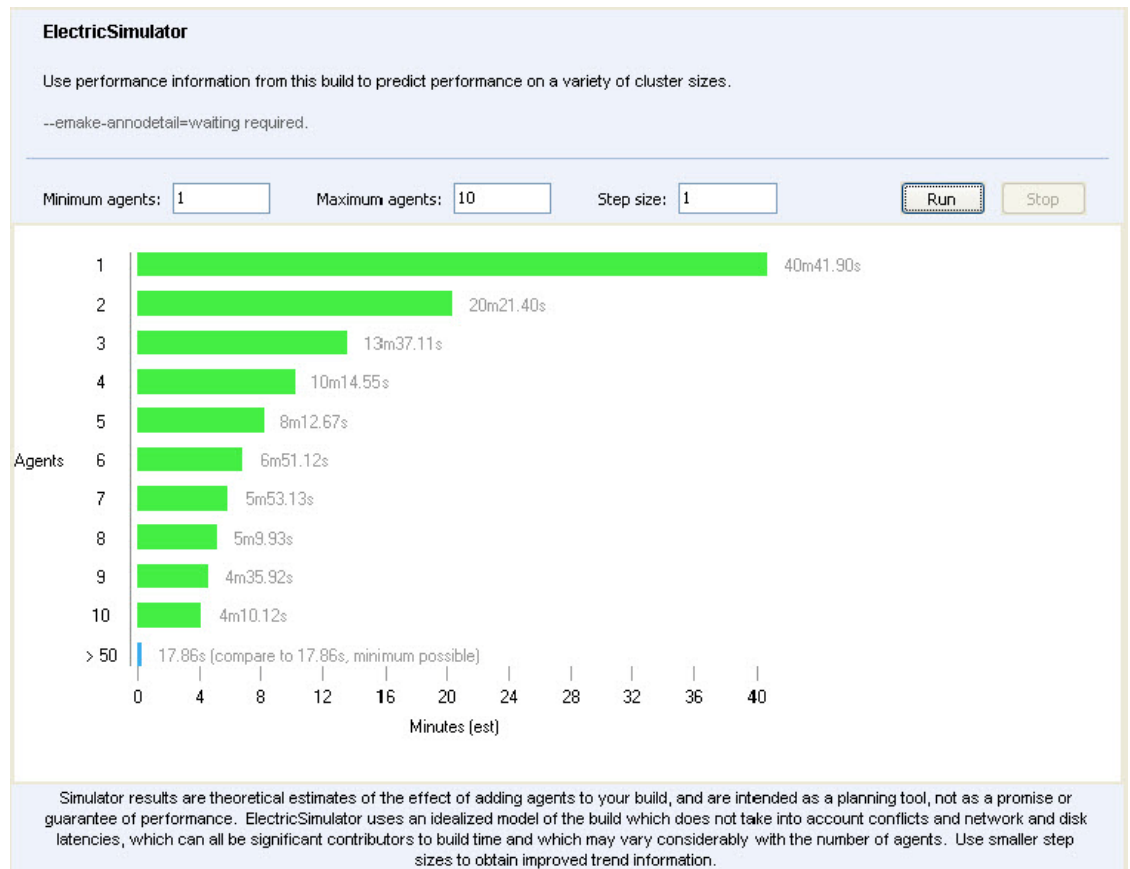
Click **Copy Report** to copy the displayed chain of operations (in the lower display area) to the clipboard.

ElectricSimulator

The ElectricSimulator report predicts build performance on a variety of cluster sizes.

This report requires

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



Fill in the fields and click **Run**.

Minimum agents - the minimum number of agents with which to run the simulation.

Maximum agents - the maximum number of agents with which to run the simulation.

Step size - the number of agents to increment the count each time.

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

The last time indicator (blue bar) is the minimum possible build time.

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

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

After you run the report, filter the results. In the Filter field, type the string you want to filter for and press Enter.

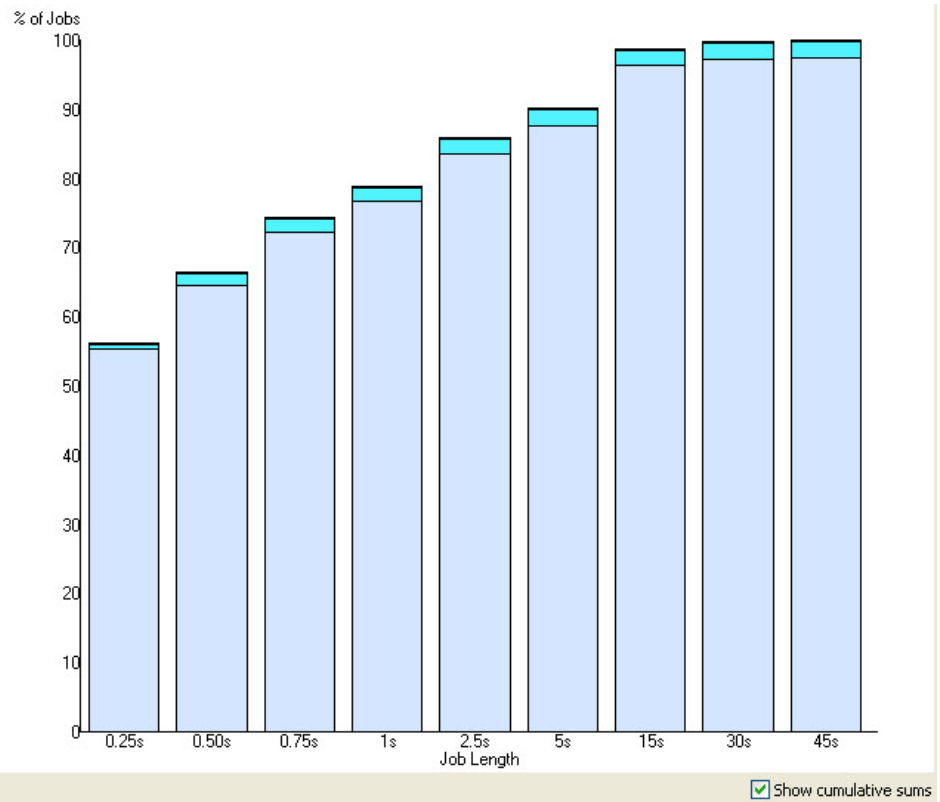
Use an asterisk to match any number of characters and use a question mark to match any single character.

Simple regular expressions are also supported. For example, `*[xz].o` and `*[x-z].o` are supported.

Filters are case sensitive.

Job count by length

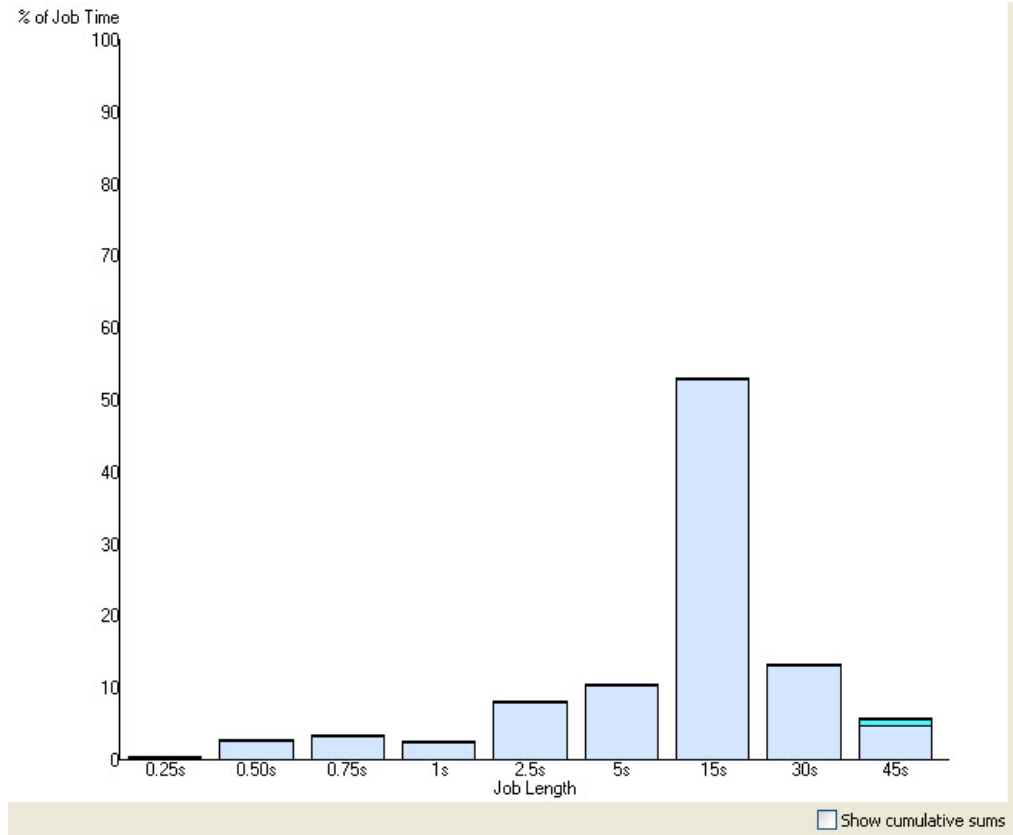
This bar chart shows jobs grouped into different length ranges. Each bar is color-coded to show the proportion of jobs for each job length.



By default, the second bar (and succeeding bars) count the unique jobs in that time range only. Therefore, the bar labeled 0.50s shows only jobs that took less than 0.50s but more than 0.25s. In the preceding chart, however, **Show cumulative sums** is enabled.

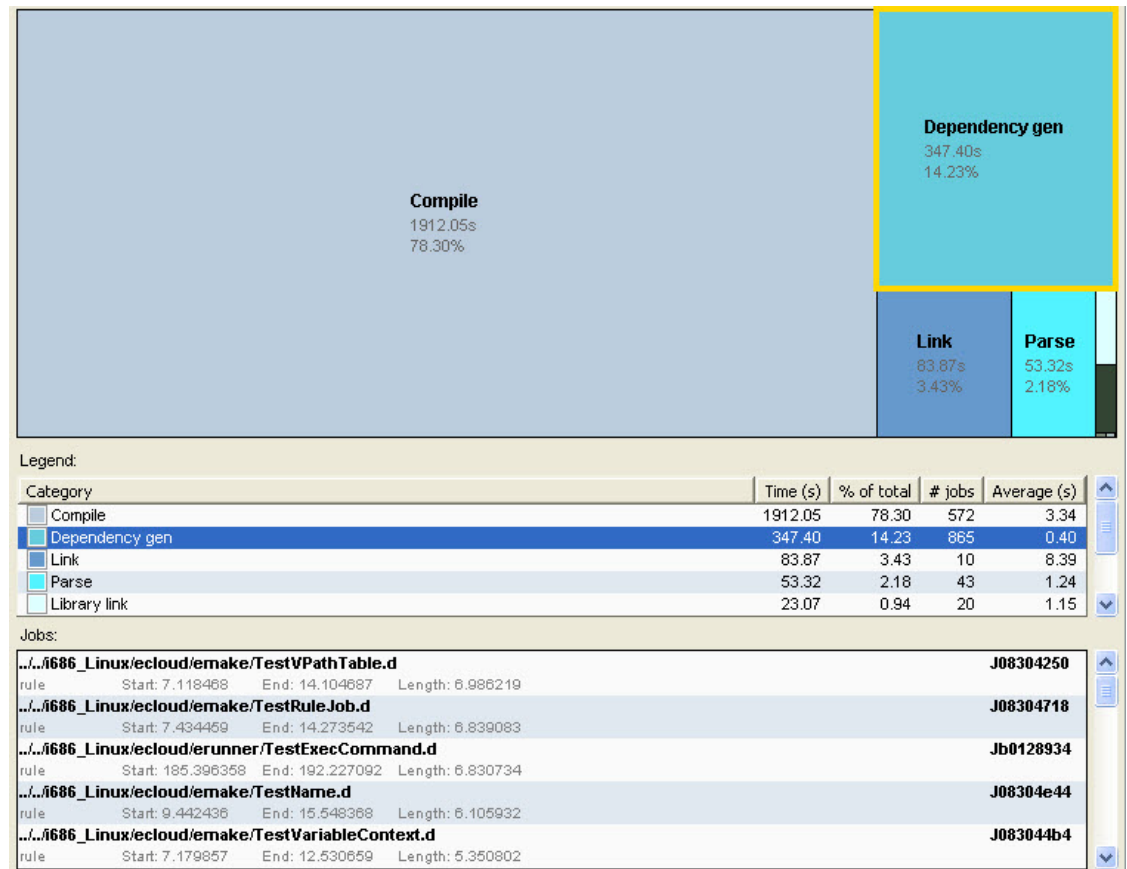
Job time by length

This bar chart shows how total build time is consumed by different length jobs. While the Job count by length report shows your build has a large number of very short jobs, this report shows which job length consumes the largest part of the build. Bars are color-coded to show which job types use the most time. The Show cumulative sum toggle allows you to see cumulative statistics.



Job time by type

This treemap 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.



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 the job's details. You can also display all jobs of a specific type by clicking an area of the treemap.

Jobs by agent

The Jobs by agent report lists the number of jobs run by each agent in the build. Click an agent to display the jobs that it ran. Double-click a job to display its details.

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	
././6886_Linux/eccloud/testmain/dummy				J0824a970
rule	Start: 1.811771	End: 1.831088	Length: 0.019317	
././6886_Linux/eccloud/testmain/TestTester.d				J0824aa3c
rule	Start: 1.831125	End: 3.285789	Length: 1.454664	
././6886_Linux/eccloud/util/TestTime.d				Jb421461c
rule	Start: 3.285834	End: 3.823118	Length: 0.537284	
././6886_Linux/eccloud/util/TestStringPool.d				Jb4214e14
rule	Start: 3.823151	End: 6.698036	Length: 2.874885	
././6886_Linux/eccloud/util/Exec.d				Jb4217c4c
rule	Start: 6.708147	End: 7.120006	Length: 0.411859	
././6886_Linux/eccloud/util/Pipeline.o				Jb421fa38
rule	Start: 12.640807	End: 19.538568	Length: 6.897761	
././6886_Linux/eccloud/util/System-unix.o				Jb4220c8c
rule	Start: 19.538635	End: 23.154224	Length: 3.615589	

Jobs by file

The Jobs by file 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/ecldoud/emade/emade			Jb36c7c1c
rule	Start: 114.173690	End: 156.010421	Length: 41.836731
.../i686_Linux/ecldoud/webui/mod_ecldoud/mod_ecldoud_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/ecldoud/emade/GMake.o			Jb36c79b8
rule	Start: 84.076576	End: 114.173629	Length: 30.097053
.../i686_Linux/ecldoud/emade/Build.o			Jb36c5048
rule	Start: 33.639376	End: 61.030681	Length: 27.391305
.../i686_Linux/ecldoud/emade/ShellEmulator.o			Jb36c70f4
rule	Start: 72.897960	End: 95.217902	Length: 22.319942
.../i686_Linux/ecldoud/emade/RemoteNode.o			Jb36c6dc4
rule	Start: 67.761896	End: 89.233741	Length: 21.471845
.../i686_Linux/ecldoud/agent/Session.o			J086bc1f8
rule	Start: 171.062765	End: 191.910578	Length: 20.847813
.../i686_Linux/ecldoud/emade/RelocTable.o			Jb36c6b60
rule	Start: 63.667325	End: 84.076530	Length: 20.409205
.../i686_Linux/ecldoud/emade/History.o			Jb36c5dd4
rule	Start: 47.863931	End: 67.761811	Length: 19.897880

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

Longest serial chain


This report displays the sequence of serialized jobs with the longest end-to-end runtime in the build. This 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.

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

This report requires

--emake-annodetail=waiting,history




☒ Find longest serial chain overall
☐ Find the longest serial chain leading to a specific job



Go

Longest serial chain overall is 1m24.81s

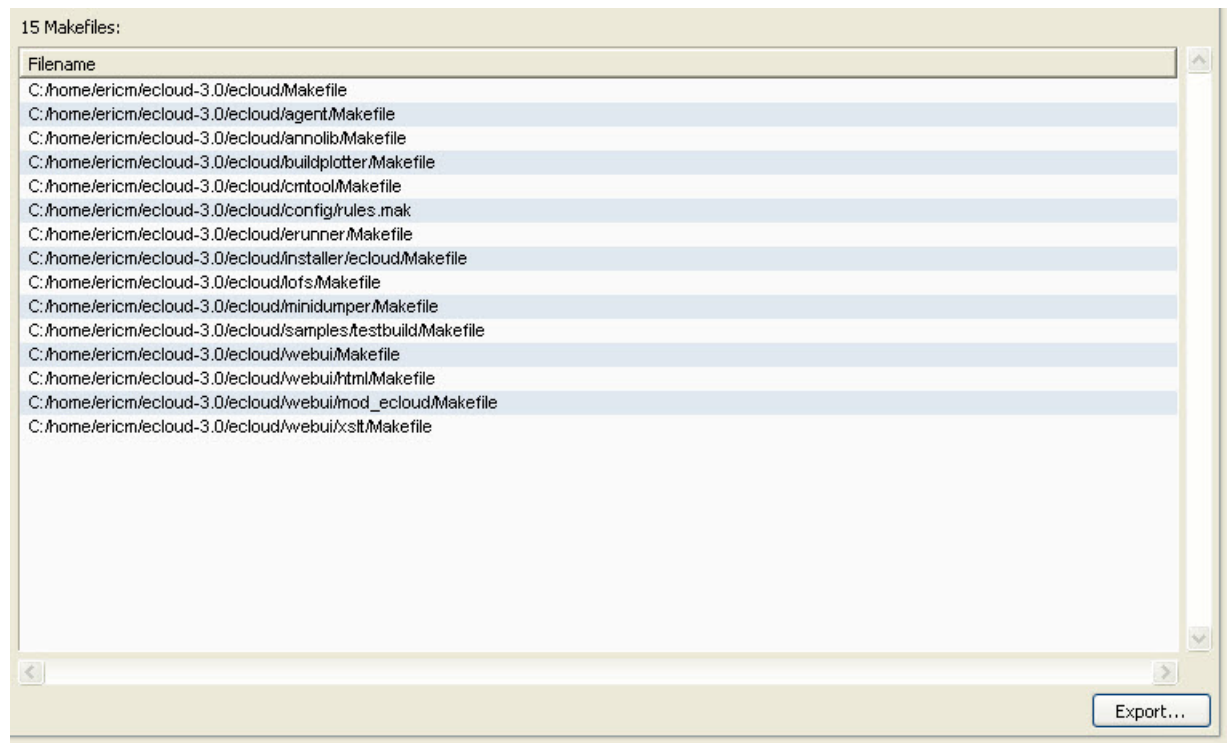
<no name>				J0821e0f8
parse	Start: 1.134574	End: 1.293646	Length: 0.159072	
Makefile				J08253988
exist	Start: 1.294643	End: 1.294699	Length: 0.000056	
<no name>				J08253cb8
remake	Start: 1.295344	End: 1.295381	Length: 0.000037	
all				J08253d84
rule	Start: 1.295495	End: 1.565408	Length: 0.269913	
<no name>				J0826f0d8
parse	Start: 2.026760	End: 2.641829	Length: 0.615069	
../i686_Linux/ecldoud/emake/dummy				J083040b8
rule	Start: 7.085405	End: 7.103173	Length: 0.017768	
../i686_Linux/ecldoud/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/ecldoud/emake/dummy				Jb36bfa38
rule	Start: 33.347195	End: 33.347396	Length: 0.000201	
../i686_Linux/ecldoud/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/ecldoud/emake/GMake.o				Jb36c79b8
rule	Start: 84.076576	End: 114.173829	Length: 30.097053	

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

Makefile manifest

This report lists all makefiles used in the build. To export the list to a file, click **Export**.



Root conflicts

This report lists all root conflicts in the build.

A root conflict is a conflict that was not caused by an earlier conflict. ElectricInsight divides all conflicts found in a build into two categories:

- root conflicts
- 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	
PClreg				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 can detail dependencies between a pair of jobs. For example, you want to know what jobs are waiting for after you look at the Annotation tab and see that the job has waitingJobs.

This report requires

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

and works best with

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

Type the ID of the job you started first in the first job field; type the ID of the waiting job in the second job field, and click **Locate Dependency**.

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

Locate Dependency

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_vobfile_v3_1.o	create	read

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

In some cases you can use this report to understand the longest serial chain, but dependencies may 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.

Creating Custom Reports

You can create custom reports and make them accessible in ElectricInsight.

Example: Creating a report to display the number of jobs run by each agent used in the build

A report typically consists of a single `.tcl` source file containing Tcl code required to perform build annotation analysis as well as the Tk code required to display the result in the ElectricInsight UI.

Directories

On startup, ElectricInsight scans the directories listed below for `.tcl` files defining new reports. You can save your report in either location and ElectricInsight automatically includes the report the next time it starts.

Linux directories:

- `/opt/eccloud/ElectricInsight/reports`
- `$HOME/.eccloud/ElectricInsight/reports`

Windows directories:

- `c:\eccloud\ElectricInsight\reports`
- `$USERPROFILE\Electric Cloud\ElectricInsight\reports`

Reports saved in the Electric Cloud directories listed above are available to all users running ElectricInsight. Reports in other locations are available to a single user only.

Scripts

1. The Tcl script must declare the report to ElectricInsight using the `DeclareReport` API, which takes three arguments:
 - The report name - the name displayed in the reports list in the ElectricInsight Reports dialog.
 - The name of the user-defined function invoked to create the report.
 - A report description - this description is displayed in the Description field of the ElectricInsight Reports dialog when the user clicks the report name.

For example:

```
DeclareReport "Jobs by agent" CreateJobsByAgentReport {
  Count the jobs run by each agent.}
```

2. The Tcl script contains the function to perform the analysis and display results. The function is invoked with two arguments:
 - The name of the “widget” the function needs to create to display results.
 - The name of the variable 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.

```
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 build annotation information currently loaded in ElectricInsight. To access this information in your report, import the `anno` variable: `global anno`

3. Write the Tcl code to perform the analysis on the annotation information:

```
# Iterate through all the jobs in the build, counting
# the number of jobs run by each agent.

array set count {}
$anno jobiterbegin
while { [$anno jobitermore] } {
    set job [$anno jobiternext]
    set agent [$anno job agent $job]
    if { $agent ne "" } {
        if { [info exists count($agent)] } {
            incr count($agent)
        } else {
            set count($agent) 1
        }
    }
}
```

4. Write the Tk code to display results. ElectricInsight 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
```

If you develop a report, you do not need to restart ElectricInsight to reload your report. Instead, select Tools > Reload reports.

Using ElectricInsight to Increase Build Speed

Identifying Build Issues

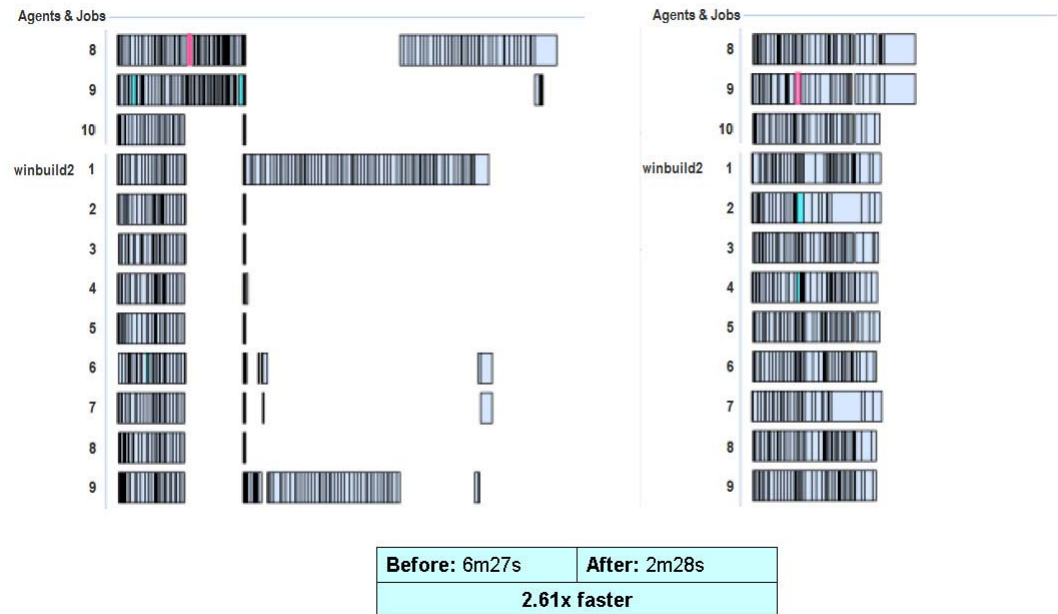
Using ElectricInsight to understand build performance can help you to identify and solve issues that provide the largest performance gains.

At a minimum, specify `annodetail` as `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 may affect performance slightly, specify `basic` level detail only, unless you are actively pursuing a performance issue.

When viewing an annotation file in ElectricInsight, 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 the Longest serial chain report to help estimate the best possible performance you can reasonably get from the build. Use the Serialization analysis report to 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 adjustment the build to eliminate serializations. The image on the right shows the result.



Additionally, when examining the Agents & Jobs section, watch for the following:

- one agent bar is longer than others
- if there are gaps when an agent is not running a job, carefully examine the running jobs while some agents are idle

Annolib

annolib is the library used to parse annotation files.

anno create	Create a new anno object that can be used to query information from an annotation file. The return value is a handle which can be used in subsequent [<i>\$anno ...</i>] call. For example: <pre>set anno [anno create] \$anno agents</pre>
<i>\$anno agents</i>	Retrieve a list of agents that participated in this build.
<i>\$anno comparejobs field jobId jobId</i>	Compare two jobs for sorting by the given field, returning -1, 0, or 1 if the first job is earlier, equal to, or later than the second job.
<i>\$anno destroy</i>	Release all resources associated with this anno instance and remove the command for controlling it from the interpreter.
<i>\$anno duration</i>	Retrieve the length of the build described in the anno object. This is the greatest time index seen in the annotation file.
<i>\$anno environment</i>	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.
<i>\$anno file command filename</i>	Query the anno object for information about a file referenced by the build.
<i>\$anno file isdirectory filename</i>	Return a boolean indicating whether the specified file is a directory or not.
<i>\$anno file operations filename</i>	Return a Tcl list of lists describing the operations that referenced this file in the build.
<i>\$anno file type filename</i>	Return the file type for the specified file.
<i>\$anno files</i>	Retrieve a list of the files referenced in Operations in this annotation file.
<i>\$anno filecount</i>	Retrieve a count of the number of files referenced in Operations in this annotation file.
<i>\$anno indexagents</i>	For each agent in the annotation, construct a list of jobs run on that agent, sorted in order of time they were invoked. This index is stored internally, for use by ElectricInsight when rendering the annotation.
<i>\$anno load channel ?count?</i>	Parse <i>count</i> bytes of data from <i>channel</i> , creating job objects as necessary. The return value is a boolean indicating whether or not parsing is complete. If <i>count</i> is not specified then all data will be read from <i>channel</i> .

<code>\$anno loadstring data ?done?</code>	Parse the data in <i>data</i> , creating job objects as necessary. There is no return value. If <i>done</i> is specified, it indicates whether or not the string represents the end of the XML data.
<code>\$anno job command jobId</code>	Invoke commands on a particular job from the annotation.
<code>\$anno job agent jobId</code>	Retrieve the name of the agent the job was run on.
<code>\$anno job annolength jobId</code>	Retrieve the length of the segment of the annotation file describing this job.
<code>\$anno job annostart jobId</code>	Retrieve the start of the segment of the annotation file describing this job.
<code>\$anno job commands jobId</code>	Retrieve a list of the commands associated with this job. The result is a list of lists with the form: <code>cmdlist = {[cmd]}</code> <code>cmdlist</code> is a list containing one or more <code>cmd</code> <code>cmd = {lines argv [output]}</code> <code>cmd</code> is a list containing <code>lines</code> , <code>argv</code> , and one or more <code>output</code> <code>lines</code> = the actual line(s) from the makefile from where the command was issued, e.g. "11-12" <code>argv</code> = the actual command issued to the system <code>output = {src out}</code> <code>output</code> is a list containing <code>src</code> and <code>out</code> <code>src</code> = the source of the output, e.g. "make" <code>out</code> = the actual string emitted by the component
<code>\$anno job conflict jobId</code>	Returns registry conflict information for the specified job.
<code>\$anno job conflictfile jobId</code>	If this job is a conflict job, retrieve the name of the file that it conflicted over.
<code>\$anno job conflicttype jobId</code>	If this job is a conflict job, retrieve the type of the conflict.
<code>\$anno job deps jobId</code>	Retrieve the list of history dependencies for this job. This may be empty if annotation did not include history-level detail, or if there were no history dependencies.
<code>\$anno job environment jobId</code>	Retrieve environment for the specified job.
<code>\$anno job exitcode jobId</code>	Retrieve the exit code for this job.
<code>\$anno job finish jobId</code>	Retrieve the end time of the job in seconds.
<code>\$anno job flags jobId</code>	Retrieve the flags for the job as a Tcl list.
<code>\$anno job isconflict jobId</code>	Returns a boolean indicating whether or not this job is a conflict job.
<code>\$anno job isrerun jobId</code>	Returns a boolean indicating whether or not this job is a rerun job.
<code>\$anno job isreverted jobId</code>	Returns a boolean indicating whether or not this job is a reverted job.
<code>\$anno job length jobId</code>	Retrieve the duration of the job in seconds. This is equivalent to <code>[\$anno job jobId finish] - [\$anno job jobId start]</code> , but it avoids using Tcl's <code>[expr]</code> command, so it is much faster.
<code>\$anno job make jobId</code>	Retrieve the Make instance ID for the Make containing the specified job.
<code>\$anno job makefile jobId</code>	Retrieve the name of the makefile containing the rule that produced this job, if any.
<code>\$anno job name jobId</code>	Retrieve the name for the specified job.
<code>\$anno job neededby jobId</code>	Retrieve the ID of the job that caused this job to be run.
<code>\$anno job operations jobId</code>	Retrieve the list of Operations performed by this job, in order, as a Tcl list of lists.

<code>\$anno job operations -type registry jobId</code>	Retrieve the list of registry operations for the specified job.
<code>\$anno job partof jobId</code>	Retrieve the ID of the job that this job continues. Only valid for continuation jobs; other jobs will return an empty string.
<code>\$anno job rerunjob jobId</code>	For conflict jobs, retrieve the job that was used to rerun the job.
<code>\$anno job serialorder jobId</code>	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.
<code>\$anno job start jobId</code>	Retrieve the start time of the job in seconds.
<code>\$anno job timing jobId</code>	Retrieve the full timing information for this job as a list of lists of the form {start finish agent} {start finish agent} Normally there will be only one entry in the list; if the job was restarted due to cluster sharing or agent failure, there will be additional entries. Jobs that never ran will return an empty list.
<code>\$anno job type jobId</code>	Retrieve the type of the specified job.
<code>\$anno job submakes jobId</code>	Retrieve the list of submakes performed by the specified job.
<code>\$anno job waitingjobs jobId</code>	Retrieve the list of jobs that had to wait for the job to complete before running. This may be empty if there are no waiting jobs, or if the annotation file did not include waitingJobs annotation.
<code>\$anno job writejob jobId</code>	For conflict jobs, retrieve the job that wrote the file that the job conflicted over.
<code>\$anno jobcount</code>	Return a count of the number of jobs in the annotation.
<code>\$anno jobiterbegin</code>	Initialize the job iterator to the head of the list of jobs in the anno object.
<code>\$anno jobitermore</code>	Return a boolean indicating whether the next call to [<code>\$anno jobiternext</code>] will return a valid job or not. This is used in conjunction with [<code>anno jobiterbegin</code>] and [<code>anno jobiternext</code>] to efficiently iterate through the list of jobs in the anno object, in serial order.
<code>\$anno jobiternext</code>	Retrieve the job with the next highest serial order in the anno object using an iterator initialized with [<code>anno jobiterbegin</code>]. 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.
<code>\$anno jobsearch attribute pattern</code>	Search the jobs in the annotation for jobs that match the given criteria.
<code>\$anno make command makeId</code>	Invoke commands on a particular Make from the annotation.
<code>\$anno make commandline makeId</code>	Retrieve the command line for the specified Make.
<code>\$anno make job makeId</code>	Retrieve the job ID for the job that spawned the given Make instance, or an empty string if no job spawned the Make.
<code>\$anno make level makeId</code>	Retrieve the level of the specified Make instance.
<code>\$anno make mode makeId</code>	Retrieve the emulation mode of the given Make instance.
<code>\$anno make workingdir makeId</code>	Retrieve the working directory for the specified Make instance.
<code>\$anno makecount</code>	Return a count of the number of Makes in the annotation.
<code>\$anno metrics</code>	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 [<code>array get</code>]. The list is unordered.
<code>\$anno parseoptions ?optionList?</code>	Query or set the anno object parse options. This controls which portions of the annotation file are processed when [<code>anno load</code>] is invoked.

<code>\$anno properties</code>	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.
<code>\$anno refcount</code>	For testing only. Retrieve the reference count from the anno object.
<code>\$anno rjobiterbegin</code>	Initialize the job iterator to the end of the list of jobs in the anno object.
<code>\$anno rjobitermore</code>	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.
<code>\$anno rjobiternext</code>	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.
<code>\$anno sortjobs ?options? jobList</code>	Sort the jobs in jobList according to the given criteria. WARNING: This modifies the list IN PLACE.
<code>\$anno type command type</code>	Query aggregate attributes of jobs by type.
<code>\$anno type conflicttime type</code>	Returns the amount of time spent on jobs of this type that were later determined to be in conflict.
<code>\$anno type jobcount type</code>	Returns the number of jobs of this type.
<code>\$anno type reruntime type</code>	Returns the amount of time spent on jobs of this type that were rerun jobs.
<code>\$anno type revertedtime type</code>	Returns the amount of time spent on jobs of this type that were later reverted.
<code>\$anno type time type</code>	Returns the amount of time spent on jobs of this type that were not conflict, reverted, or rerun jobs.
<code>\$anno types</code>	Retrieve a list of known types of jobs.