

Subject: Aegis Frequently Asked Questions (FAQ)
From: Peter Miller <pmiller@opensource.org.au>
Newsgroups: comp.software-eng,comp.software.config-mgmt,
comp.software.testing,comp.sources.d,alt.sources.d

Submitted-by: Peter Miller <pmiller@opensource.org.au>
Archive-name: aegis.4.25/FAQ

This message contains answers to a number of frequently asked questions about Aegis. If you don't know what Aegis is, read on.

Suggestions for additions or improvements to this FAQ are most welcome.

Contents

Subject: 1. What is Aegis?

Aegis is a transaction-based software configuration management system. It provides a framework within which a team of developers may work independently, and Aegis coordinates integrating these changes back into the master source of the program, with as little disruption as possible.

For a more complete description, see the README file or the User Guide. Both are available from the anonymous FTP site (see below).

Subject: 1.1. What operating systems are supported?

Aegis will run on almost any version of UNIX. The distribution contains a configure script generated by GNU autoconf to adapt to your system.

There is no Aegis port to NT, OS/2, MS-DOS or VMS. The author has no expertise in these operating systems. If you do have such expertise, port Aegis, and I will try to include your work in the next release.

Subject: 1.1.1. Cygwin

There is a very capable UNIX emulation layer known as Cygwin available from Cygnus. Please see <http://sourceware.cygnum.com/cygwin/>. If you get too excited, see the next item.

Subject: 1.1.2. Windows NT

Aegis depends on the underlying security provided by the operating system (rather than re-invent yet another security mechanism). However, the POSIX *setuid* system call, which has no direct equivalent on Windows NT. This makes porting difficult. Single-user ports are possible, but not usually what folks want.

Compounding this is the fact that many sites want to develop their software for both Unix and Windows NT simultaneously. This means that the needs to be guaranteed to be handled in the same way by both operating systems, otherwise one can act as a "back door" into the repository. Users and permissions (sourced from the same network register of users) on both Unix and Windows NT, making the mapping almost impossible. It did actually correspond.

Most sites using Aegis and Windows NT together do so by running Aegis on the Unix systems, but building and testing on the NT system accessed via Samba or NFS.

Subject: 1.2. Where can I get Aegis?

Aegis is available by WWW from

URL:	http://aegis.sourceforge.net/	
File:	aegis.4.25.README	# the README file from the tar set
File:	aegis.4.25.faq	# Frequently Asked Questions
File:	aegis.4.25.lsm	# description in LSM format

File:	aegis.4.25.rm.pdf	# PDF of the Reference Manual
File:	aegis.4.25.tar.gz	# the complete source
File:	aegis.4.25.ug.pdf	# PDF of the User Guide

Subject: 1.3. Is Aegis actively being maintained?

Yes, Aegis is actively being maintained by the author. You can contact him by email
Peter Miller <pmiller@opensource.org.au>

Subject: 1.4. Is there an Aegis mailing list?

Yes. Discussion may include, but is not limited to: bugs, enhancements, and applications. The list is not moderated.

The address of the mailing list is
aegis-users@canb.auug.org.au

Do not send email to the list if you wish to subscribe.

To subscribe to this mailing list, send an email message to `majordomo@canb.auug.org.au` with a message body containing the
`subscribe aegis-users`

If you have an address which is not readily derived from your mail headers (majordomo is only a Perl program, after all) you will need
`subscribe aegis-users address`

where *address* is an email address to which you want messages sent.

The mailing list is archived at eGroups. The URL is <http://www.egroups.com/list/aegis-users/info.html> and then follow the links.

The software which handles this mailing list **cannot** send you a copy of *Aegis*. Please use FTP or ftp-by-email, instead.

Subject: 1.5. How does Aegis compare with program X?

For all X, "The author has no experience with X. If you wish to contribute a comparison between Aegis and X, it will be considered for

Subject: 2. Configuration and initial build problems

Aegis is accompanied by a set of regression tests, and the BUILDING instructions included in the distribution detail how to run these tests.

Subject: 2.1. Changing Makefile and common/config.h

It is a Bad Idea to change the Makefile or the common/config.h file by hand. This is because much of the work done by the configure script

In particular, if you change the C compiler (CC) you absolutely must do this with the involvement of the configure script.

Subject: 2.2. RCS

The tests distributed with Aegis use RCS as their history tool. It is important that you use version 5.6 or later.

There seems to be a problem with the version of RCS distributed with HP/UX, even though it claims to be RCS-5.6.0.1. You will need to install it on a HP box.

Subject: 2.3. SCCS

The tests do not automatically detect that you have SCCS. You will need to go out and grab RCS even if you already have SCCS installed.

Subject: 3. Testing

One of the things many sites like about Aegis, is that it provides mandatory testing. This enforcement of testing is optional, and is configurable.

Please note that even if Aegis' testing framework is not useful to your project, the other aspects of Aegis' process still are (e.g. concurrency, etc.).

Subject: 3.1. Can I use something besides a shell script?

Yes, the "test_command" field of the project configuration file may be set to whatever command you like, see *aeprconf(5)* for more information.

A shell script is the default, because you can run anything out of the shell script. In particular, you can set up a temporary directory with a cleanup script into this temporary directory when running tests, cleanup, no matter what the fallout, even a core dump, is thus a simple matter of "rm -rf".

Subject: 3.2. Testing curses programs

I don't have a curses program tester, nor do I know of one. It seems to me that the "screen" program contains all the necessary elements required to make it a suitable test harness.

If anyone has found suitable curses testers, even commercial ones, I would be happy to list (FTP) locations here.

Subject: 3.2.1. Expect

Some sites have been using "expect" to test curses programs. You need to be careful about terminal types. The author is Don Libes <libes@cs.unc.edu>

expect is available by anonymous FTP from

Host: sunsite.unc.edu
Dir: pub/Linux/system/Shells
File: expect-5.12.tgz

The author has not personally used this system, and so can make no comment on it.

Subject: 3.3. Testing X11 programs

I don't have an X11 program tester, however several commercial ones seem to be advertised heavily.

If anyone has found suitable X11 testers, even commercial ones, I would be happy to list additional (FTP) locations here.

Subject: 3.3.1. replayXt

ReplayXt is a library that allows Intrinsics (or Xt) based application to be executed from a script file. In particular, applications based on Intrinsics can be played. The author is Jan Newmarch <jan@pandonia.canberra.edu.au>

replayXt is available by anonymous FTP from

Host: ftp.x.org
Dir: /contrib/devel_tools
File: replayXt.README
File: replayXt.1.1.tar.z

The author has not personally used this system, and so can make no comment on it. This entry originated from Simon Pickup <simon@pandonia.canberra.edu.au>

Subject: 4.3. How do I enforce coding standards?

The “develop_end_policy_command” field of the project configuration file is for this purpose. If this command fails, the change may not be accepted.

For example, if you wanted to check that all files conformed to project coding standards, you could run the hypothetical “check_standards.sh” script (see the User Guide for more information on how to create a script like this), using

```
develop_end_policy_command = "sh ${source check_standards.sh}";
```

This script would use the *aelfc* command to obtain the list of files to be checked.

Subject: 4.4. How do I get dates printed before and after build commands?

The *build_command* field of the project configuration file can also be used to do more than just build:

```
build_command = "set +e; date; cook ...; x=$?; date; exit $x";
```

You need to be careful about the *-e* flag, because Aegis invokes the shell to run the commands with the *-e* option, to abort after the first error.

You may want to upgrade to Aegis 3.0 or later. In this version, the times are automatically printed as part of the verbose commencement of a build.

Subject: 4.5. How do I stop Aegis automatically merging?

The merging behavior is controlled by the “diff_preference” field of the user configuration file. See *aeuconf*(5) for more information. There are also options to the *aed*(1) command which can override the preferences, see *aed*(1) for more information.

To stop the automatic merging, add the line

```
diff_preference = no_merge;
```

to the *.aegisrc* file in your more directory. To specifically perform a merge after that, you will need to use the “*aed -merge_only*” command. For more convenience, see the User Guide.

Subject: 4.6. How do I modify which developer is assigned to a change?

There have been times when we have one developer working on a change and, for one reason or another, we need assign that change to another developer. This is at a point where it can be integrated. Is there a way to tell Aegis that a new developer has been assigned to the change so that it would work accordingly? (My thanks to Mark Meuer <markm@endo.com> for this question.)

Yes, you need to use the *aechown*(1) command. This command may be run by a project administrator to reassign a change to a different developer. It will not work over NFS, a new development directory owned by the new developer is created, and the change files are copied, before the old change is deleted.

Subject: 4.7. Do I have to type all the configuration examples in the User Guide?

No, you do not. The *lib/config.example* directory contains a number of files with the configuration command from the User Guide ready to be copied into your configuration file.

Subject: 4.8. Is there a way to recreate a previous baseline?

For example, let’s say we have shipped one version to a customer and since then made 30 changes to the baseline. When the customer can’t reproduce, how can I easily rebuild the baseline from 30 changes ago to help track down the error?

Yes, it is possible to recreate a previous baseline. You need to follow these steps:

1. Determine which delta was shipped to the customer. This is easiest if you embed the version supplied by Aegis into your executable files.
2. Create a change a change (you want to fix the bug, right?) and start developing it.
3. Copy all of the project files into the change, using the delta number determined in step 1. Use these commands:

```
aecd
```

```
aecp . -delta N
```

where N is the delta number from the first step. Files created since the delta will be copied into your change as empty files. leave them empty.

Note that you need Aegis 2.3 for directory copying to work correctly. Earlier versions will need to use

```
aecd
```

```
aecp `aegis -list project_files -terse` -delta N
```

Note the backward quotes. The above can be abbreviated, its just long so you can see what it is doing.

4. Build the change as normal. This will completely rebuild the version sent to the customer. Note that a number of things are beyond updates and compiler updates. These can have an effect in accurately reproducing what was sent to the customer.

5. Find the bug and fix it, as you would normally do.

5. Merge the change. This will bring the files up to the most recent version, and merge the bug fix with the current version.

6. You can now uncopy all of the files which are unchanged. Aegis provides a fast way to do this

```
aecpu -unchanged
```

This command behaves like aecpu, but it goes hunting for files which are the same between the baseline and the development directory, the merge.

Subject: 4.9. How do I stop it using the pager?

Aegis automatically redirects the output of listing and help into the pager program described in the \$PAGER environment variable, or *m* if aegis is running in the foreground and the output is a terminal.

The simplest way to stop the use of the pager is to pipe the output through cat. A more elegant method is to use the -No_Pager option.

If you want to permanently disable the pager, set the following line in your .aegisrc file:

```
pager_preference = never;
```

If you want to use the pager now, you will need to use the -PAGer option.

Subject: 4.10. Why does develop end complain about builds?

Why does aede complain that "this change must successfully complete an 'aegis -Build' before it can end development" but I have just

There are two possibilities: time stamps and architectures.

Subject: 4.10.1. Build Time Stamps

Aegis tracks time stamps on files and builds, etc, to determine when files are out of date. If time clocks on various networked machines conclude that a file or files are logically out of date.

This problem is mentioned in the User Guide. Essentially, you need to install NTP or XNTP. If you can't do that, run *rddate*(1) on your machines to sync their clocks with the file server's clock.

Subject: 4.10.2. Build Architectures

The "aede" can also complain about builds when you have changed the architecture field of the project configuration file, but have not yet built. This can also happen when a separate change updates the architecture field, and all existing changes suddenly get this mystery error.

It happens because the default architecture is "unspecified" in the change attributes (use "aeca -l" to take a look). When the project configuration file is updated to a new architecture, according to the change's attributes, is still "unspecified". To fix, use "aeca -e" and edit the change's attributes to match the new architecture in the configuration file.

You may wish to upgrade to version 3.0, which checks for this, and prints a suitable error message, with a suggestion to edit either the project configuration file or the change's attributes.

Subject: 5. Internationalization

This section discusses a number of issues surrounding internationalization, and the effect it has upon Aegis and its error messages.

Subject: 5.1. Say What?

Internationalization is the process of modifying code so that it may produce messages in the native language of the user. A data file is supplied containing the strings to be substituted for error messages and other text produced by a program.

Subject: 5.2. So why are the error messages So Bad, now?

There is a set of default error messages in the code itself. These error messages are used if the localization file cannot be found, or if the localization file for some reason. These messages tend to be in the “programmer cryptic” language, and the English localization has provided. The English localization *needs* to be installed.

There is also a small, slight, tiny (cough) possibility that the necessary code changes have introduced bugs into some error reporting. (I’m in a third arm pit.) Please report any problems immediately to the author.

Subject: 5.3. What controls which language is used?

There are a range of environment variables, depending on your operating system. The most common is the LANG environment variable. If nothing is specified.

Subject: 5.4. What localizations are available?

Only English is available at this writing.

More may become available if there are any generous bilingual Aegis users out there, who want to provide a localization file in another language.

Subject: 5.5. So can I change the text of error messages?

Yes, you can, but it is not advised. Seriously. (I’m going to regret admitting this, so I’m not going to tell you how.) If you have suggestions, please email them to the author.

Subject: 6. Branches

This section discusses a number of issues surrounding branching.

Subject: 6.1. What is a Branch?

Aegis treats branches as “big changes”. This allows much of Aegis’ development model to be applied unchanged, but still allow the development to be broken down into fine grained pieces, or aggregated into large “release sized” chunks.

Subject: 6.2. What is an Anabranh?

Aegis’ branches can end. This is a consequence of treating branches as “big changes”. Logical places to end branches are times like when a new version is released. The geographic term for a stream which breaks off and rejoins downstream is an anabranch.

Because branches can end, combined with the fact that branches are “big changes”, this means that you use “aede” to end development and to make any other change on its parent branch.

Subject: 6.3. Do branches have to end?

No. You are not required to end development of a branch, just as you are not required to end development of a change. In particular, the branch which never ends. How you manage your project is up to you.

Subject: 6.4. Why are branches called projects?

In order for the branches to be “big changes”, there are times when you will refer to the branch as a change to its parent project. In order there are times when you will refer to them as a project.

The conversion between the two is intended to be simple. When you need to refer to a branch in its project role, join the branch’s parent with a dot; *e.g.* “-p aegis.1.2”. When you need to refer to a branch in its change role, separate the branch’s project name from the branch name; *e.g.* “aegis.1 -c 2”.

Subject: 6.5. Why must aed be used when integrating a branch?

Because branches are “big changes” and follow *all* of the things a change must do, including aed. The only time a branch is writable is when it is a branch. Thus, the aed for the branch is done between aeib and aeipass. The trunk doesn’t require it, because you never say aede to the trunk.

Subject: 6.6. How come the branch baseline is incomplete?

The branch baseline is the development directory for the “big change” represented by the branch. Just as a developer’s development directory is the directory they worked on, and their implications, so does the branch baseline.

This means that wherever you specify a search path (-I, VPATH, *etc*) to look first in your development directory and then in the baseline of the branch’s ancestors’ baselines. Aegis provides a “\$search_path” substitution telling you where to search; this is usually used in the “build” file.

The “aefind” command may also be used to perform actions similar to use usual Unix find(1), but across the Aegis search path.

End of aegis.4.25.faq Digest
