Eli System Administration Guide

$Revision: 1.34 $

Compiler Tools Group
Department of Electrical and Computer Engineering
University of Colorado
Boulder, CO, USA
80309-0425

Table of Contents

This manual gathers together information that may be needed by an administrator of an Eli installation.

In some cases more up to date information can be obtained by consulting the information at Eli pages on the World Wide Web. Also, new system releases or patches will be made available via these pages. At present the following Web pages of interest exist:

```
(United States)
       http://www.cs.colorado.edu/~eliuser
(Australia)
       http://www.cs.jcu.edu.au/~tony/eli.html
(Germany)
       http://www.uni-paderborn.de/fachbereich/AG/agkastens/eli_home.html
```

1 Making Eli Executable

Odin is an expert system that manages build processes and storage for intermediate files. What build processes Odin manages are determined by the set of packages that are used. Eli is a set of Odin packages designed for compiler construction.

A suitable version of Odin is distributed with Eli. While Odin and Eli can be built, installed, and used separately, the easiest way to install both is to run the '`INSTALL`' script in the top-level directory of the distribution. The sole argument to the script should be the absolute path name of the directory where everything should be installed. This directory must already exist. If the argument is omitted, your home directory is used instead. For example, the following installs the '`eli`' and '`odin`' scripts in '`/usr/local/bin`', the default Odin packages in '`/usr/local/lib/Odin`', the Eli packages in '`/usr/local/lib/Eli`', and man pages in '`/usr/local/man`'.

```
INSTALL /usr/local
```

The '`INSTALL`' script will interactively guide you through the configuration, construction, and installation of Odin and Eli. The '`README`' file in the top-level directory gives a few hints about running the '`INSTALL`' script. The top-level directory also has a '`CLEAN`' script that will remove all of the built components from the distribution directory. It does not remove any installed versions.

Odin and Eli may be built installed separately as well. The rest of this document talks about the individual steps required to build and install Eli. The '`INSTALL`' script will guide you through them and perform some of them automatically. If problems occur, you may want to perform some of the steps by hand.

After doing the installation, make sure to add the '`bin`' directory installed above to your path.

The Eli distribution itself consists of a single directory, '`Eli`'. To create an executable version of the system, first go to that directory and copy the file '`sysdata.sh`' to '`sysdata`'. The file '`sysdata`' contains shell script variable definitions which are used to describe your system. You will need to edit '`sysdata`' to reflect the characteristics of your system.

In some case, specific installation notes have been compiled for Eli on some systems. For example, the file '`README.Linux`' in the top-level of the Eli distribution contains tips for installing Eli on Linux systems. If present, these notes should be read before attempting to install Eli.

Warning: Eli uses the C pre-processor extensively to configure source code, scripts etc. Strange things can happen if the path where you are building Eli contains a component that matches a pre-processor symbol. For example, the pre-processor on Linux defines the symbol `linux`. If you try to build Eli in, say, `/local/linux/opt` the build process will fail. The symbols to look for vary from system to system but usually the manual page for the pre-processor lists which symbols are defined by default.

The first set of definitions in '`sysdata`' need to reflect the operating system you are running on. To do so comment out all of the declarations for the variable `os` except the one which describes your system by placing a `#` at the beginning of the line. Installation of Eli has been tested on a variety of machine/os combinations (as reported by "uname"). The '`sysdata`' file contains a complete list of these combinations which usually include SunOS, Solaris, Ultrix, AIX, HP-UX, IRIX, OSF, and Linux.

The `C_COMPILER` should be set to the name of the compiler you wish to use to build the system. This compiler is also used to compile generated source files when Eli is used, but can be overridden by users setting the `ELI_CC` environment variable.

Eli can also be configured to use a C++ compiler when compiling generated source files. To do this you must set `C_COMPILER` to the name of the C++ compiler. Even though Eli itself will use the C++ compiler, you must *build* Eli with a C compiler. See below for instructions on how to do this.

The next two variables in '`sysdata`', `DEBUG_OPTION` and `PROFILE_OPTION`, must be set to the compiler options used to obtain debugging and profiling information, respectively.

The next variable in '`sysdata`', `CPP`, needs to reflect the absolute path of your system's C pre-processor. Also, the `CPPFLAGS` variable can be set if any options must be provided to the C pre-processor. If the GNU pre-processor is used, `CPPFLAGS` must be set to `-traditional`.

Some Eli tools use mathematical functions from the standard Unix math library. To enable Eli to reliably find the library, the variable `MATHLIB` should be set to the directory in which the library resides.

The remaining variables in '`sysdata`' refer to X-windows related header files and libraries. If you do not have X-windows installed on your system, leave the remaining definitions commented out as they are not necessary to install Eli. To utilize the Eli tools which use X-windows, however, you must set these variables. `XINC` and `XLIB` specify the absolute pathnames of your system's X include files and libraries, respectively. Also, one of the Eli tools requires the Athena Widgets libraries; `XAWINC` and `XAWLIB` specify the absolute pathnames of the include and library directories containing these libraries. If you don't have the Athena Widgets libraries, just leave the variable definitions commented out.

One of the Eli documentation browsers and the monitoring system make extensive use of the Tcl extension language and its Tk windowing library. If you want to use these components you must first install Tcl and Tk on your system. Versions of Tcl and Tk that are compatible with Eli are available from the place from which you obtained Eli.

Set the `TCLINC` and `TCLLIB` variables to the directories in which the Tcl include files and library, respectively, can be found. Similarly, the `TKINC` and `TKLIB` variables give the relevant directories for Tk. If these four variable are left commented out, the Tcl/Tk documentation browser used will not be built (but another one will be available) and the monitoring system will not be available.

`XVIEWINC` and `XVIEWLIB` specify the absolute pathnames of your system's include files and libraries for the Xview package. If you have X-windows installed on your system, but do not have the Xview package, leave `XVIEWINC` and `XVIEWLIB` commented out.

Finally, it is possible to build an Eli documentation viewer using the sunview library (in contrast to an X version). By default this is not done; to enable the sunview version set the variable `SUNTOOLDIR` to the location of your suntools include files.

Once you have made the necessary changes to the '`sysdata`' file, run the command `Configure` at the top-level of the Eli distribution.

After the system is done with the configuration, you can execute the command `make` to build the system. This will take anywhere from 20 minutes to an hour or more depending on the speed of your system.

If you have configured Eli to use a C++ compiler (`C_COMPILER` above), you must specify a C compiler to the `make` command you use to build the system (Eli cannot currently be built using a C++ compiler). For example, if you have the GNU C compiler (gcc) on your system a command like the following is sufficient to build Eli.

```
make CC=gcc
```

The biggest problem with porting Eli to a new system involves the details of the representation of symbol information in the a.out(5) format. Eli requires this information to determine which of the generated modules will actually be used in a particular compiler. Only one routine, '`pkg/eli/src/rdsymb.c`', is affected. Generally, the problem will manifest itself by compilation errors in the routine `read_symbols`.

The purpose of `read_symbols` is to extract information about global symbols from an object file. An *external* is a global symbol that is used in the current object file but defined in another object file; an *entry* is a global symbol that is defined in the current object file and used in another object file. The routine `read_symbols` scans a specified object file, calling `SymbolInfo` for each global symbol that is either an external or an entry. It passes `SymbolInfo` the symbol's string and a Boolean that is true if the symbol is an external.

Thus the routine `read_symbols` must embody an understanding of the object code format for the machine on which you are running. You will need to dig up the necessary information by looking at the a.out(5) manpage, and probably one or more system include files.

2 Installing Eli

When Eli is installed, an Odin package directory is created. This directory contains all of the tools and auxiliary files needed to run Eli as a set of packages managed by the Odin system. Odin provides the environment variable, `ODINPATH`, which specifies directories with the packages to use. The '`INSTALL`' script in the top-level directory will install an '`eli`' invocation script that sets `ODINPATH` to the location of the Eli packages and invokes Odin. Note that if you install Eli manually, you will have to provide this '`eli`' script yourself. The script looks as follows if the Eli packages were installed in '`/usr/local/lib/Eli`' and the '`odin`' script was installed in '`/usr/local/bin`'.

```
#! /bin/sh

ODINPATH=/usr/local/lib/Eli; export ODINPATH
/usr/local/bin/odin $*
```

To install the Eli packages manually, let '`dir`' be the full path name of the desired directory ('`/usr/local/lib/Eli`' in the above example) and execute the following command in the distribution directory:

```
make install DESTDIR='dir'
```

Once Eli has been installed you can remove all files from the distribution directory except the original source text by executing the command

```
make clean
```

When Eli has been made, extraneous files are present in the build directory. The following command can be used to remove all of these unnecessary files but retain all of the files necessary to install Eli without recompilation of source code.

```
make tidy
```

See Section 'Top' in Eli User Interface Manual, for details on how to use Eli.

Some extra steps are required to support printing the online documentation on PostScript printers. If your site wishes to support this, you will need to obtain a PostScript tar file from the same place you received your Eli distribution. You can then extract this tar file into the '`info/1.0.0/manuals`' directory of your installed Eli package directory. The files can then be printed directly from the documentation browser.

The Info online documentation system used in Eli has the capability to mail a message to the maintainer of the Eli system. (See Section 'Complaining' in Info.) The default value for the mail address used by this facility is the mailing list for reporting Eli bugs (see Chapter 3 [problem reporting], page 9). If many people are using Eli at a particular site, the responsible person at that site might want to receive that mail.

To change the mail address edit the file '`pkg/info/texinfo/Makefile.cpp`'. You should change the value of the `make` variable `MAINTAINER` to a more suitable value. Eli must be reconfigured (run `Configure`) and the documentation browsers must be recompiled and installed for the change to take effect. For example, if Eli were installed in '`/tools/Eli`', the following commands would be sufficient:

```
cd pkg/info/texinfo
'Edit Makefile.cpp'
make
make install DESTDIR=/tools/Eli/info/1.0.0/texinfo
```

3 Procedure for reporting a problem with Eli

Should you have a problem with Eli please E-mail a note describing the problem to:

```
elibugs@cs.colorado.edu
```

When reporting problems, please include the Eli version number (printed when Eli starts up, or available in the distribution in 'pkg/version/version.dg') and the output from running the shell command `uname -a` on your system. Also, please give *complete* output illustrating the problem. If necessary, please include relevant specification files.

In some cases it will also be helpful to identify the version of the software in question. Most distributed files have RCS tags embedded in them. Most executables are generated from sources that have an initialized variable *RCSid* such as:

```
char rcsid[] = "$Id: misc.c,v 1.2 88/11/30 12:57:39 bob Exp $";
```

It is easy to identify the source versions from which binaries have been generated by running the RCS `ident` utility on them. For example, running `ident` on 'pkg/gla/gla_fe' yields the following information:

```
gla_fe:
     $Id: driver.c,v 1.15 1996/03/22 19:28:48 kadhim Exp $
     $Id: dfltclp.c,v 1.8 1994/12/08 03:09:52 tony Exp $
     $Id: source.c,v 2.5 1995/01/16 02:19:21 waite Exp $
  ...
```

Sites that do not have `ident` can extract this information by one of the following commands:

Source
: `grep '\$Id:'` 'file-name'

Executable
: `strings` 'file-name' `| grep '\$Id:'`

4 Saving space

If you want to use Eli on more than one architecture (eg. Sun 4 Sparc or DEC Alpha) then you will need more than one installed version of Eli. However it is possible to share many of the files between these installed versions. The script '`elilink`' in the top-level of the Eli distribution can be used to set up this sharing by using file system links. Typically about half of an installed Eli system can be shared. This implies, for example, that you can have three shared versions but only consume disk space equivalent to two separate versions.

'`Elilink`' is used as follows:

`elilink` *working-directory master-directory*

Master-directory is a directory containing an installed version of Eli which is to be used as the complete master copy. It will be unchanged by `elilink`. *working-directory* is *another* installed version of Eli which is to share files with the master directory. That is, files in *working-directory* which are identical to their counterparts in *master-directory* will be linked to the master copy. '`Elilink`' produces messages to let you know what it is doing as it performs the link operations.

'`Elilink`' calls itself recursively. This means that you will either need to put the script in a directory that is on your `PATH` or change the script so that it knows where to look. If you want to do the latter, change the `ELILINK=` line near the top of the script to point to where the script resides.

Index