<u>ELROS</u>

Installation
Guide

Jimmy Guse
Linda Stanberry

<u>Last Update</u>: 12/4/92 @ 10:41 AM

Table of Contents

I Introduction

The ELROS system is distributed as a Unix tar file that contains sources and Makefiles to construct the runtime system. The ELROS Makefile structure makes use of an include file, Make.Defaults, that defines all the macros used to configure the ELROS environment. This enables configuration flags, options, and pathnames to be passed to nested Makes using this same file.

In order to have the Makefile structure work properly, Make.Defaults needs to be edited to provide installation-dependent options before trying to build ELROS. These options are described below in section *II: Configuring*.

Section *III: Making* describes the target options for the top-level make.

Section *IV: Packaging* describes the make target that builds an ELROS tar tape for distribution.

Section *V: Tailoring* describes how to tailor or fine tune your ELROS runtime system configuration for particular applications.

II Configuring

You will need to select the appropriate Make.Defaults from the **./defaults** subdirectory, and copy this file to the root ELROS directory, renaming the selected file (e.g., SUN.Make.Defaults) to Make.Defaults. Then you need to edit the copied Make.Defaults file to ensure that the configurable options are the way you want. This file consists of macro definitions that are used by all the Makefiles in the ELROS tree. They are explained below. Most are given default values that are appropriate for the particular machine configuration and for a basic ELROS configuration. Pathnames are not given as defaults, however, and you should review the other configuration options as well.

RO_DIR - set to the path name of the root directory of the ELROS tree to be constructed. This should be the path name to the directory in which Make.Defaults resides. This path name is used in turn to find the runtime, preprocessor, and driver branches of the ELROS tree. The driver (ecc) will use these path names when invoking the preprocessor and loading with the runtime library.

RO_TASKING_FLAG - Set to __RO_TASKING if you wish to use the PADS multi-tasking package available with the ELROS system; otherwise, set to NO__RO_TASKING.

RO_OPT_REAL - set to 1 if you want to enable attempts at REAL optimization for your specific architecture. If the code is not set up for your machine, it will use portable real number manipulation routines. If ELROS has difficulty determining the correct routines for your architecture, you may have to set this flag to 0.

RO_HAS_LONGLONG - set to 1 if your C compiler supports the syntax "long long", set it to 0 if not. Note that ANSI C compilers support this syntax, but not all ANSI C compilers treat this as a type distinguishable from long. Set to 1 only if your compiler supports a separate long long type.

RO_HAS_LONGDOUBLE - set to 1 if your C compiler supports the syntax "long double", set it to 0 if not. As with long long, set to 1 only if your C compiler supports a separate long double type.

RO_SWITCHES - set to reflect desired C compilation options, -g for development environment.

ELROS currently supports 2 communications platforms: ISODE and PURESOCKETS. Your system must support at least one of these platforms, and you indicate which platforms are supported by the following option.

RO_PLATFORMS - set to ISODE if only ISODE is supported; set to PURESOCKETS if only PURESOCKETS is supported; and set to ISODE_and_PURESOCKETS if both are supported.

If ISODE is supported, you must specify the following. If ISODE is not supported (i.e., not included in RO_PLATFORMS), these macros will not be used, but you can set them to null if desired.

ISODE_INCDIR - set this to the path containing the ISODE include files.

ISODE_LIBDIR - set this to the path containing the ISODE library, 'libisode.a'.

ISODE_BINDIR - set this to the path containing the ISODE utilities (rosy, posy, pepy), if supported; set to null otherwise.

ISODE_OTHERS - set this to the path containing the ISODE "others" subdirectory; the "lookup" service in this subdirectory is used by the ELROS test suite.

For your specific machine, you need to designate what file the preprocessor and library should use for determining ELROS type mapping to native types. This information is kept in the file **./parameters.dat**, which has to be configured for your machine. Possible configurations are given in the subdirectory **./params**. Use the following macro to choose one of the predefined files, or one of your own creation which you add to **./params**.

RO_PARAMS - set to designate which file in **./params** should be copied into **./parameters.dat**.

The Make.Defaults file also contains a section of macro definitions that are used by the various Makefiles to carry out system specific commands. These macros definitions should already be tailored correctly for each of the files in **./defaults**. No macro definitions after **RO_PARAMS** should require editing, and are therefore not described here, except for the following macros which are referenced in section IV of this document.

The following macros are used in the development environment to enable creation of distribution tar tapes. They are not intended for general use, however. See section *IV: Packaging* for more on their use.

RO_DIST_DIR - set to the path name of the directory in which to build a distribution subdirectory and tar file.

RO_INSTALL_DIR - set to the path name of the directory in which to install an ELROS system.

III Making

ELROS is compiled and linked using a single Makefile, which in turn invokes the Makefiles in each of the product branches of the

ELROS tree. The products of the Make are the ELROS preprocessor, library, driver, and utilities. In addition the Makefiles set up links to ISODE header files and libraries as needed.

Once the environment has been configured by editing the Make.Defaults file, the entire ELROS system can be built with the command:

make New

To make only a new version of the runtime system and test it use:

make Runtime Tests

The latter option will update the runtime system and execute the test suite. In general, to make a new version of one of the ELROS products, use **make** with the capitalized name of that branch of the ELROS tree (e.g., **make Preprocessor** will invoke the Makefile in the preprocessor branch, and update the preprocessor products as needed).

Invoking with no arguments:

make

in the top-level directory will run the Makefiles in each subdirectory of ELROS, updating all products as necessary.

To force all parts of the ELROS system to be rebuilt from scratch (e.g., after editing the Make.Defaults file), reuse the command:

make New

To force remaking just one of the ELROS products, use the same command as to update, but first do a **make clean** in the product directory.

To summarize, the following targets are available in the top-level Makefile for building and updating ELROS: New, Runtime, Preprocessor, Driver, Util, and Tests.

IV Packaging

This section describes **make** targets that are really for development use, and not needed for building an ELROS system from a distribution tar file. However, this describes how the tar files are made.

A built-in mechanism is available to build a distributable ELROS package. It is invoked using the following command, executed in the top-level directory of an ELROS "installation" :

make Distribution

This process builds the distribution in the "distribution" subdirectory and produces a tar file named "new-elros.tar" in the top-level ELROS directory. To build the distribution subdirectory, each subdirectory of the ELROS installation is visited, and **make package** is invoked in order to copy the needed files to the appropriate place in distribution. Each Makefile uses the macro RO_DIST_DIR to determine where to copy its files, and the macro DISTFILES to determine what files are copied.

An ELROS "installation" is a directory containing subdirectores for the sources and products of the ELROS system, representing snapshots of each product at a given time. In our development environment, different persons are responsible for maintaining different ELROS products, and there is one common installation that represents the latest versions of all products. The distribution tapes for ELROS are made in this common installation tree.

For each product, there is a **make install** target to update the common installation from the development directory, which is basically just a copy from the private directory to the shared directory. The macro RO_INSTALL_DIR determines where the files will be copied, and macros DISTFILES and INSTALLFILES determine which files will be copied.

In our development environment, we rely on SCCS to maintain a history of our source file modifications. There is a special target of the top level Makefile that will make sure all files have been delta-ed in to SCCS before installing new products:

```
make   delta-check
```

This will report any files that are currently being edited in any of the directories defined in the macro DIRS.

V Tailoring

ELROS can be tailored for different mappings for your hardware configuration. The file **./parameters.dat** contains information about the hardware configuration of your machine. It may be changed to utilize alternate data typing in your C compiler or for a particular application. A default version of this file is provided with the distribution, but as you become more acquainted with ELROS, you may wish to tune the mappings for your own needs.

As described in section II of this document, the default parameters file is selected via the RO_PARAMS macro. You can edit the resulting **./parameters.dat** file to produce new mappings. The information contained in this file is described below.

The parameters file contains lines each beginning with a keyword and followed by 0 or more fields. The "#" character, when appearing at the beginning of a line, indicates that the line is a comment. The keyword lines are as follows:

Types

You specify how ELROS types are mapped onto C types through the following. Each line specifies an ELROS type class (e.g., Boolean, Integer, or Real), a maximum size of the ELROS type class, and then the C type into which you want all ELROS objects of that type class and no larger than the indicated size, to map.

Boolean *size* *c-type*
Integer *size* *c-type*

Each of these lines specifies a C-type corresponding to a particular ELROS type class and length. Each type class may be mapped into different *c-type*s, depending on the ELROS type class length. The lines for an ELROS type class must appear in ascending order of the *size* value, although they need not be together. The *c-type* is the C declaration form used for the type. For each *c-type*, *size* should specify the **sizeof** that type on your machine.

Real *m-size* *e-size* *real-c-type*

This line specifies the parameters for Real types. The lexical conventions are the same as for the other types, except that *real-c-type* must be one of "float", "double", or "long double", *m-size* is the number of (binary) bytes of mantissa, and *e-size* is the number of (binary) bytes of exponent. Ordering is by the sum of *m-size* and *e-size*.

Buffer Sizes

You specify the size of buffers to be used when sending and receiving operations and/or data with the following.

SENDBUFFERSIZE *bytes*
RECEIVEBUFFERSIZE *bytes*

Each of these lines must appear exactly once. The values are used as hints to the runtime system as to what buffer sizes to use. If either or both of these lines is omitted,*bytes*=512 is assumed. Some research results indicate that an optimal size is 4096 bytes.

Endianness

BIG_ENDIAN
LITTLE_ENDIAN

One of these keywords must appear. They specify the endianness for certain non-native types which are stored in byte arrays.

Significant Characters

SIGNIFICANT_ID_CHARS *count*

Truncate names defined within interfaces to *count* significant characters. Errors are reported when the truncated names conflict within an interface. Use this option when the C-compiler or loader does not support arbitrary length names. By default *count*=16. *count*=0 means don't truncate.

SIGNIFICANT_INTERF_CHARS *count*

Truncate interface names to *count* significant characters,. Errors are reported when the truncated names conflict. Use this option when the C-compiler or loader does not support arbitrary length names. By default *count*=8. *count*=0 means don't truncate.

Bind Symbols

BIND *bind-symbol*

Any number of these lines may appear in the parameters file. *bind-symbol* is an identifier for a bind statement supported by the runtime library. Normally these lines should not be changed, since the runtime-library is not mutable.