Administration of Online Documentation [9]

This chapter describes the administrative procedures required for the online glossary and the UNICOS online message system. Also, it describes the Cray DynaWeb server (which replaces the Docview and CrayDoc documentation tools) and explains how to add local man pages.

9.1 Modifying online glossary files

The UNICOS `define`(1) utility allows quick, online retrieval of Cray Research technical terms and their definitions, as well as terms and definitions added by a local site, that match a specified search string. With the `define` utility, Cray Research provides a definitions file called `CRAYdefs_i`, which contains embedded keywords. Sites can modify this definitions file or can add local definitions files by using the `builddefs`(1) utility.

9.1.1 Modifying the Cray Research definitions file

The following procedure shows how to use `builddefs` to modify the Cray Research definitions file that the `define`(1) utility uses:

Caution: If you make changes to the `CRAYdefs_i` definitions file, the changes will be lost when a UNICOS revision or update is installed. In this case, back up the `CRAYdefs_i` file, do the installation, and then reapply the changes you backed up.

1. Copy the `CRAYdefs_i` definitions file, which has embedded keywords, from the default definitions directory, `/usr/lib/define`, to your working directory.
2. Edit the file to make the desired changes.
3. Run `builddefs` on the edited file, as follows (`CRAYdefs_i` is the input file, and `CRAYdefs` is the output file):

   ```
   builddefs CRAYdefs_i CRAYdefs
   ```

   This command produces a `CRAYdefs` file, which is a definitions file without embedded keywords and a `CRAYdefs_k` file, which is a keyword file.

4. Set the DEFINEDIR environment variable to the working directory (*directoryname*) that contains the CRAYdefs and CRAYdefs_k files. For the Korn and standard shells, set the variable, as follows:

   ```
   DEFINEDIR=directoryname
   export DEFINEDIR
   ```

   For the C shell, set the variable as follows:

   ```
   setenv DEFINEDIR directoryname
   ```

5. Test the modified file by using the define(1) utility on selected terms.
6. Install the CRAYdefs_i, CRAYdefs, and CRAYdefs_k files in the /usr/lib/define directory.
7. If necessary, reset the DEFINEDIR environment variable. For the Korn and standard shells, reset the variable as follows:

   ```
   unset DEFINEDIR
   ```

   For the C shell, reset the variable as follows:

   ```
   unsetenv DEFINEDIR
   ```

9.1.2 Creating a local definitions file

The following procedure shows how to use the builddefs(1) utility to create a local definitions file for use by the define(1) utility. When multiple definitions files are in the definitions directory, the define utility reads the files in alphabetical order; that is, it searches file aaa before file bbb.

Caution: If your site begins the installation process from a clean partition, your local definitions files installed in the default define directory, /usr/lib/define, might be removed during the installation of a UNICOS revision or update. In this case, back up your local definitions files, install the UNICOS revision or update, and then reinstall your local definitions files.

1. Prepare an input file that contains embedded keywords according to the keywording rules contained in the following section.
2. Run builddefs on the local file, as shown in the following example. *sitedefs*_i is the input file, and *sitedefs* is the output file. This command

produces a *sitedefs* file, which is a definitions file without embedded keywords, and a *sitedefs*_k file, which is a keyword file.

```
builddefs sitedefs_i sitedefs
```

3. If you want to test how your local file works, set the DEFINEDIR environment variable to the working directory (*directoryname*) that contains the new formatted definition file. If you do not want to test the local file, skip to step 6. For the Korn and standard shells, set the variable as follows:

```
DEFINEDIR=directoryname
export DEFINEDIR
```

For the C shell, set the variable as follows:

```
setenv DEFINEDIR directoryname
```

4. Change directories by entering the following command line:

```
cp sitedefs sitedefs_k directoryname/
```

5. Test the new file by using the define(1) utility on selected terms.

6. Install the *sitedefs* and *sitedefs*_k files in the /usr/lib/define directory. The define utility reads the files in this directory and searches them in sequence for search string matches.

7. If necessary, reset the DEFINEDIR environment variable. For the Korn and standard shells, reset the variable, as follows:

```
unset DEFINEDIR
```

For the C shell, reset the variable as follows:

```
unsetenv DEFINEDIR
```

9.1.3 Glossary keywording rules

The builddefs utility reads a definitions file that has embedded keywords to produce a keyword file and a definitions file without embedded keywords. A pair of keywords in the form ++*term* marks each definition in the input file. Synonyms for the term, if any, also begin with ++ and follow the keyword line. (Do not mark synonyms in pairs.) You should enter all keywords and synonyms into the definitions file and use the correct capitalization conventions. The keywords and synonyms are recorded in the keyword file in lowercase characters.

Example:

```
++access control list
++ACL
  The access control lists (ACLs) are an extension to the normal
  UNICOS file discretionary access control. ACLs support the ability
  to grant or deny access to a file on any user and/or group
  basis. For more information on ACLs, see the acl(1) man page.
++access control list
```

The `builddefs` utility checks all keywords for keywording errors. The following rules apply to keywording:

- The two ++ symbols that appear in columns 1 and 2 of the intermediate definitions file identify a keyword. The keyword immediately follows the ++ symbols, with no intervening blank spaces and tabs. Empty keywords (that is, ++ with no following text) are not allowed.
- A keyword can consist of up to 48 characters. If a keyword is longer than 48 characters, it will be truncated.
- Each definition must have two keywords (a matching pair). The first keyword indicates the start of the definition. The second keyword indicates the end of the definition.
- Synonyms for a keyword are in the form ++*synonym* and are limited to 48 characters. Do not mark synonyms in pairs.

9.2 Cray message system

The Cray message system (formerly called the UNICOS message system) consists of tools and procedures for issuing error messages to users from program code and delivering documentation on those messages. The message system is based on the X/Open Native Language System specification.

This section contains information that administrators need to install, maintain, and update message system files under the UNICOS operating system.

Warning: Sites using the Cray ML-Safe configuration of the operating system can use the information and procedures outlined in the following sections to change or add messages. However, for changed messages, you must not alter the original, underlying meaning of the message.

Message system files are easy to install. They are shipped in both source and catalog format, and they are ready to use after they are loaded in the proper

directories. This section focuses on message system terminology, the location of message system files, and procedures for rebuilding message catalogs when message information changes.

9.2.1 Overview

The message system includes the following features that aid in improving error reporting and problem resolution:

- Published guidelines for writing good messages and good message documentation
- Message catalogs, located separately from the program code, that contain the text of the messages issued at run time
- Explanation catalogs that contain a discussion of the error and suggest solutions
- Online user access to message documentation by using the `explain`(1) command
- User control of the message format through the `MSG_FORMAT` environment variable
- Message text source files distributed with the release

These features create the following advantages for products that use the message system:

- Messages are more informative and usable.
- Online and printed explanations are readily available to users and administrators.
- Messages are easier to trace to their source because they contain a unique identifier that includes their product of origin.
- Messages and explanations are centralized in catalogs. The text of both is readily accessible for update and translation.
- Users can change the message format by using the `MSG_FORMAT` variable.

These advantages are achieved through a design that removes error messages from program code and places them in a *message text file*, which also includes explanations for each message.

The message text file is processed into a catalog of messages and a catalog of explanations. Library calls in the program code access the *message catalog* at run time. An accompanying *explanation catalog* contains explanations of the messages in the message catalog. To access these explanations, use the `explain`(1) command.

The system administrator is responsible for installing, maintaining, and updating the message catalogs. The following sections explain how to work with message text files and message catalogs. They describe how to install the message system files so that UNICOS programs can access them. They also describe how you can update your message catalogs if your site wants to add or change message or explanation text.

See the *Cray Message System Programmer's Guide*, Cray Research publication SG–2121, for a complete description of the message system from a programmer's perspective.

The man pages for the message system routines contain descriptions and examples of the commands, routines, and environment variables that compose the message system. See the following man pages:

- `caterr`(1)
- `catxt`(1)
- `explain`(1)
- `gencat`(1)
- `whichcat`(1)
- `catgetmsg`(3)
- `catgets`(3)
- `catmsgfmt`(3)
- `catopen`(3) and `catclose`(3)
- `nl_types`(5)
- `msg`(7D)

9.2.2 Message system files

The message system uses the following three kinds of files:

- Message text file
- Message catalog
- Explanation catalog

The message text file is the source file for message system text. The message and explanation catalogs are binary files produced from the message text file by using the `caterr`(1) command. The release tape includes both the source and binary forms of the message system files.

9.2.2.1 File names

Each type of message system file has a name in the form *group.suffix*. The group code (*group*) identifies the product, and the suffix (*suffix*) identifies the file type.

The group code is any string that relates to the product or products that the file supports. For example, the `segldr`(1) and `ld`(1) loader commands use the `ldr` group code. The `explain`(1) man page lists all of the group codes used by Cray Research software.

Each type of message file has a different suffix after the group code. The message text file has the suffix `.msg`, the message catalog has the suffix `.cat`, and the explanation catalog has the suffix `.exp`.

Thus, the following three message system files are associated with the SEGLDR product:

File name	Description
`ldr.msg`	Message text file
`ldr.cat`	Message catalog
`ldr.exp`	Explanation catalog

9.2.2.2 File location

The location of a message system file is determined by its type, text or catalog, and its product.

The message text file (*group*`.msg`) is located in the source tree with other files in the product's program library (for example, the message text file used by the loaders is located in the `/usr/src/prod/segldr/ldr.msg` file).

Most message and explanation catalogs (*group*`.cat` and *group*`.exp`) are installed in the `/usr/lib/nls/En` directory. Some products must be available

when the /usr/lib file system is not mounted; the catalogs for these products are installed in the /lib/nls/En directory.

9.2.3 Installing message system files

The release media includes the message text file, message catalog, and explanation catalog for each product that uses the message system. Cray Research recommends that you install the message files initially without changes. The UNICOS installation process creates the /usr/lib/nls/En and /lib/nls/En directories on your system and copies the message system files to these directories. With these files in place, the message system functions correctly and issues messages from UNICOS software.

9.2.4 Changing the message text file

The message system lets you update the message and explanation catalogs with site-specific information. Situations in which you may want to add site-specific information to existing catalogs include the following:

- A local modification to the code has created the need to add a new message or to change an existing message.
- A particular error condition has a site-specific remedy that you want to describe in the explanation.
- You want to add names or phone numbers for persons or groups to be notified if certain errors occur.
- You are creating a new program and want to use the message system to issue the error messages.
- The site wants to translate the messages into a different native language; catalogs of messages in the target language must be created and installed.

Warning: Local modifications to message and explanation catalogs will be overwritten during the installation of the next UNICOS revision or update that contains those catalogs. Sites must back up their local modifications, install the new catalogs and message text files, and reapply their local modifications.

If catalogs for a product are mistakenly deleted from the system, you may have to rebuild them.

The following sections describe how to edit and rebuild message system files.

9.2.5 Editing the message text file

The message text file is the source file for messages and explanations. If you make changes to a product that have an impact on that product's messages, this is the file that you must change.

The message text file contains the following four types of information:

- Message text, preceded by the `$msg` tag
- Explanation text that contains `nroff` formatting codes, preceded by the `$nexp` tag
- Plain ASCII explanation text, preceded by the `$exp` tag
- Comments, consisting of `$<space><`*text*`>`, `$<tab><`*text*`>`, or `$<newline>`

Blank lines are also acceptable within a message text file, but they are ignored during text-to-catalog processing.

Edit the message text file to include new information. Ensure that the resulting file conforms to the format specified in *Cray Message System Programmer's Guide*, Cray Research publication SG–2121.

9.2.6 Rebuilding catalogs

After new information is incorporated into a message text file, you must rebuild the related catalogs. You can build catalogs in two ways:

- Use the makefile for the product to remake the catalogs
- Use message system commands to remake the catalogs

The first method, using the makefile, is simpler. It requires fewer steps and less intervention on your part. However, it is less flexible because it calls the message system commands in a specific way.

The second method, using message system commands to remake the catalogs, gives you more options, but it also requires that you understand more about the message system commands and how to use them.

The following sections describe the two methods of rebuilding catalogs.

9.2.6.1 Rebuilding with nmake

The makefile for each product builds a message and explanation catalog from the message text file. It places these catalogs in the current directory (usually within the source tree). The `nmake install` command places the catalogs in the proper subdirectory. (It also reinstalls other software in that directory.)

The makefile varies from product to product, but, basically, `nmake` calls the `caterr` command twice. The first call to `caterr` creates a message catalog from the `$msg`-tagged information in the message text file. This message catalog is placed in the message system directory. For a discussion of where catalogs are located in the directory structure, see Section 9.2.2.2, page 367.

The second call to `caterr` creates an explanation catalog from the `$nexp`- and `$exp`-tagged information in the message text file. The explanation catalog is placed in the same directory as the message catalog.

To create an explanation catalog from source material tagged with `$nexp`, `caterr` calls `nroff`(1) and a file of message macros. `nroff` processes the formatting codes embedded in the explanations and passes the formatted text back to `caterr`. `caterr` then completes its catalog creation process.

9.2.6.2 Rebuilding with message system commands

The `caterr`(1) command rebuilds message system catalogs from the message text file. You can use `caterr` to rebuild a message catalog or an explanation catalog. Rebuilding both catalogs for a product requires that you invoke `caterr` twice. See the `caterr`(1) man page for details of the syntax.

For example, if changes were made only to a product's messages (not to the explanations), use the `caterr` command to process the messages into an updated message catalog. Use the `-c` option to call `gencat`(1).

The following command rebuilds the `ldr.cat` message catalog from the `ldr.msg` message text file:

```
caterr -c ldr.cat ldr.msg
```

The `caterr` command processes the text file, then calls `gencat`, which creates the new message catalog.

If changes were made only to a product's explanations (not to the messages), use `caterr` to remake the explanation catalog. Use the `-e` option to produce an explanation catalog instead of a message catalog.

The following command rebuilds the `ldr.exp` explanation catalog from the `ldr.msg` message text file:

```
caterr -e -c ldr.exp ldr.msg
```

The `caterr` command processes the text file, then calls `gencat`, which creates the new explanation catalog.

9.2.7 Printing messages

The messages for any group code can be printed as a document. You might want to do this in either of the following cases:

- Local changes are made to the message file and the site wants to provide an updated message document to users
- Cray Research has provided the messages only online, but the site wants to provide a printed message document

Follow the steps below to print a message document locally. Throughout this procedure replace *group* with the group code for the product whose messages you want to print.

1. Locate the message text file in the source tree for the product. If you are not familiar with the structure of the source tree, use the following `find` command syntax to locate the message text file. This invocation of the command displays the path name of the file *group*`.msg`.

   ```
   find /usr/src -name group.msg -print
   ```

2. Locate or create a header file for use in printing. Check in the directory where the message text file was found for a file named *group*`.head`. If this file exists, proceed to the next step. If it does not, create a header file that contains at least the following macros. (The `msg`(7D) man page describes the text processing macros used in this header file.)

   ```
   .GC group
   .ST "group Messages"
   .2S
   ```

3. Extract the explanations from the message text file. Use the `catxt` command to perform this step. The following invocation of the `catxt` command extracts the explanations from the file *group*`.msg` and places them in the file *group*`.nexp`. Invoke this command from a directory in

which you have write permission. This may require that you copy the message text file (*group*`.msg`) to a directory outside of `/usr/src`.

```
catxt -n group.nexp group.msg
```

If the *group*`.msg` file contains `#include` directives, the files specified in those directives must also be available in your working directory. If `#include` directives appear in the file, the messages use symbolic names instead of literal message numbers. Use the following form of the `catxt` command (instead of the command shown previously) to extract the explanations. The `-s` option resolves the symbolic names into message numbers.

```
catxt -s -n group.nexp group.msg
```

4. Process the *group*`.nexp` file with a version of the `troff`(1) text processor and print the resulting file. The commands to perform this step depend on the target printer. (This procedure assumes that the target printer is connected to a UNIX system networked to the Cray system.)

   If the target printer is capable of printing files output from the device-independent version of `troff` (sometimes called `ditroff`), go to step a. If the target printer is not capable of printing these files, go to step b.

   If you are unsure of the capability of the target printer to accept device-independent `troff` input, check the `lpr`(1) command man page for the UNIX system to which the printer is connected (not the `lpr` man page on the Cray system). If the `lpr` command accepts the `-n` option, the printer is capable of handling output from device-independent `troff`.

   a. Device-independent `troff` procedure

      Cray systems include device-independent `troff` as part of the base software release. On your Cray system, use the following command line to process the header and explanation files with device-independent `troff`:

      ```
      troff -msg group.head group.nexp | lpr -n
      ```

   b. `troff` procedure

      Many UNIX systems other than Cray systems include `troff` (not device-independent `troff`) as part of the base software release. Copy the Cray message system macro file `/usr/lib/tmac.sg` to the UNIX system connected to the target printer. Also copy the *group*`.head` and *group*`.nexp` files from the Cray system to the UNIX system.

On this UNIX system, use the following command line to process the header and explanation files with `troff`:

`troff -t tmac.sg` *group*`.head` *group*`.nexp >` *outfile*

Use the following command to print the `troff` output (*outfile*):

`lpr -t` *outfile*

9.3 Cray DynaWeb server

The Cray DynaWeb server makes Cray Research documents available to World Wide Web browsers. The Cray DynaWeb server is based on DynaWeb software produced by Electronic Book Technologies, Inc. (EBT). DynaWeb is a commercial-grade Web server that serves documents marked up in Standard Generalized Markup Language (SGML) to Web browsers for rapid navigation and searching. Documents in DynaWeb are stored in SGML and converted to hypertext markup language (HTML) when a Web browser requests the document.

Like any Web server, a DynaWeb server listens for hypertext transfer protocol (HTTP) requests from browsers. Each request contains a uniform resource locator (URL) that identifies the server and a particular block of data, such as a home page, a section of text, or an image.

By default, the Cray DynaWeb server listens for incoming requests on port 8080, which does not require super user (root) privileges.

Warning: The Cray DynaWeb server uses a port number greater than 1024 (8080, by default), so that root privileges are not required to run the Cray DynaWeb daemon. However, if you have an Internet connection, ports greater than 1024 are accessible to users external to your site. This is a security risk for your site: it does not meet the requirements of a Cray ML-Safe configuration of the UNICOS operating system, and it is a violation of copyright and licensing restrictions under which Cray Research and its customers must operate. Consequently, you must disable external acess to port 8080 on the Cray DynaWeb server or employ other access control measures for port 8080 on that machine.

The server can handle up to 256 simultaneous requests. The connections are short-lived, lasting only long enough for the server to process the requested data and send it to the browser. After sending the requested information, the server terminates its connection.

Note: The Cray DynaWeb server has been tested using Netscape Navigator version 2.02 and 3.0. The server is also accessible using the Mosaic and Lynx browsers; however, tables are not fully supported. Other browsers may work but are not explicitly supported by the server.

For detailed information about installing and administering a DynaWeb server, see *Online Software Publications Installation Guide*, Cray Research publication SG–6105 and *Online Software Publications Administrator's Guide*, Cray Research publication SG–6104.

9.4 Local man pages

The `man`(1) command displays online man pages. The `man`(1) command used in the UNICOS release is compatible with the UNIX 4.3BSD `man` command. The following two differences affect how you can implement local man pages:

- The `man` command references unformatted man pages in a set of `man`*x* directories, and formatted man pages in a set of `cat`*x* directories (the source, or unformatted, man pages are not released with the UNICOS system). Directories follow the BSD organization.

Directory	Description
`cat1`	User commands
`cat2`	System calls
`cat3`	Library routines
`cat4`	Special files (devices)
`cat5`	File formats
`cat7`	Miscellaneous information and DWB macro descriptions
`cat8`	Administrator commands
`catl`	(letter l) Local man pages

- The `MANPATH` environment variable lets you maintain local man pages in the directory of your choice. The X Window System `xman` command also uses the `MANPATH` environment variable.

The `MANPATH` environment variable lists the directories that the `man` command should search for man pages. When the `MANPATH` environment variable is not set, the `man` command, by default, searches the `/usr/man` subdirectories for man pages. Users can set the `MANPATH` environment variable to find man pages in directories other than `/usr/man`.

You can install local man pages as either source (unformatted) man pages (in `man`*x* directories) or as formatted man pages (in `cat`*x* directories). When the source page is newer than the formatted page, or when the formatted page is not available, the man command uses the man page macros defined in `/usr/lib/tmac/tmac.uc` to format source man pages.

If you install your local man pages in directories that are not affected by UNICOS upgrades, you will not have to reinstall them after future upgrades; however, users must set the `MANPATH` environment variable to include the local man page directory.

Man pages installed in the `/usr/man` `manl` or `catl` subdirectories will have to be reinstalled after each UNICOS upgrade. Save your local man pages before beginning the upgrade, then restore those packages to the appropriate location.

Only when man page searches fail on the first specified path does the `man` command search the next path. For example, by setting the `MANPATH` environment variable to include the local man page directory before the Cray Research man page directory, the `man` command will display local man pages, rather than Cray Research man pages that have the same name.

For more information about the `man` command and the `MANPATH` environment variable, see the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011.

9.4.1 Examples

The following two examples illustrate ways of using the `MANPATH` environment variable to implement local man pages on your system.

9.4.1.1 Example 1

You can install local man pages in the same directory structure as the associated binary files, such as `/usr/local`. If `/usr/local/man` is used for local man pages, you would follow these steps:

1. Create the `/usr/local/man` directory, then create the `cat`*x* or `man`*x* directories under `/usr/local/man`.
2. Have users add the `MANPATH` environment variable to their `.cshrc` or `.profile` file, as follows:

   In `.cshrc`:

   ```
   setenv MANPATH /usr/man:/usr/local/man
   ```

In `.profile`:

```
MANPATH=/usr/man:/usr/local/man
export MANPATH
```

3. Add the system-wide definition of the `MANPATH` environment variable to `/etc/cshrc` and `/etc/profile`, as follows:

   In `/etc/cshrc`:

   ```
   setenv MANPATH /usr/man:/usr/local/man
   ```

   In `/etc/profile`:

   ```
   MANPATH=/usr/man:/usr/local/man
   export MANPATH
   ```

4. Install the local man pages in the `/usr/local/man/cat`*x* or `/usr/local/man/man`*x* directories. Alternative man directories must follow the subdirectory structure and the naming convention as used in `/usr/man`. The man page file name convention allows an extension that corresponds to the number of the directory, such as `l` for `catl` and `manl`, plus a one-letter extension, as follows:

   - *file* `.1` (`b`, `c`, `g`, `m`, or `X`)
   - *file* `.3` (`c`, `f`, `g`, `i`, `l`, `m`, `n`, `r`, `s`, `u`, `x`, or `X`)
   - *file* `.4` (`f`, `n`, or `p`)
   - *file* `.7` (`d` or `X`)
   - *file* `.8` (`c`, `v`, or `e`)

9.4.1.2 Example 2

Users can have private man pages, or man pages restricted to a specific group of users. To include private man pages in `$HOME/man` or restricted man pages in */restricted_man_dir*`/man`, follow these steps:

1. Add `$HOME/man` or */restricted_man_dir*`/man` to the `MANPATH` environment variable in the `.cshrc` or `.profile` file, as follows:

   In `.cshrc`:

```
setenv MANPATH /usr/man:/usr/local/man:$HOME/man:/restricted_man_dir/man
```

In `.profile`:

```
MANPATH=/usr/man:/usr/local/man:$HOME/man:/restricted_man_dir/man
export MANPATH
```

2. Install the private man pages in the alternative man directories specified in the `MANPATH` environment variable; those directories must follow the subdirectory structure and the naming convention as used in `/usr/man` (see step 4 in example 1).

9.4.2 Display order for same-name man pages

If a local man page has the same name as a Cray Research man page, the `man` command displays the man pages in the order found. The `man` command searches the subdirectories in numerical order, 1-8, and searches the l (local) subdirectory last. For example, `man1/cat1` are searched before `man2/cat2`, and `man8/cat8` are searched before `manl/catl`.

When the `MANPATH` environment variable is set, the `man` command searches the next path only when a search fails on the first specified path. The `man` command will not display all same-name man pages installed in separate search paths.

If you want to display both man pages, do the following:

- To display the Cray Research man page first, followed by the local man page, install the local man pages in `/usr/man/catl`.
- To display the local man page first, followed by the Cray Research page, do the following:

  1. Install the local man page in `/usr/man/cat`*x* by using the appropriate file extension or suffix for `cat`*x* with contents clearly marked "Local."
  2. Move the Cray Research original page to `/usr/man/catl` (or `/usr/man/manl` source pages), with the file extension `.l`, then modify it to mark its contents "Cray Research original."