Publishing Message Explanations [4]

This section describes the procedures and guidelines to follow when formatting and processing message explanations for publication in a message document. Following these procedures lets you produce a message document in the style of the Cray Research message documentation. If you do not want to document your messages in that style, you do not need to follow these procedures.

The following topics are discussed in this section:

- Summary of publication procedures
- Message style definition
- Message markup
- Header and trailer files
- Extraction and printing
- Testing online explanations
- Troubleshooting

4.1 Summary of publication procedures

This subsection briefly summarizes the general steps required to format a message text file into both an explanation catalog accessible by using the `explain`(1) utility and a printed document suitable for publication as part of a Cray Research manual. This summary refers you to more detailed information that is contained in later subsections. Use this summary as an overview of the message process or, after you are familiar with the procedure, as a quick reference to the required steps.

1. Edit the message text file for content and format.

   Make changes to the text of the messages and explanations to bring them into conformance with the message guidelines (see Appendix A, page 55).

   Be especially careful of changes that you make to the actual error messages (text tagged with `$msg`). Do not change the order or number of variables in the message without changing the code that passes parameters to the message routines.

   Format the message explanations in conformance with the message style using the macros described on the `msg`(7D) man page. Make sure that the text of each

explanation (text tagged with $nexp) contains a copy of the message it explains. For a procedural guide to the message style, see Section 4.3, page 41.

2. Create or edit the header file for the message section. Create the trailer file.

   Create a file named *group*.head that contains the manual title, manual number, center footer, section title, and introductory paragraphs for the printed message section. The last line of this file must be a .2S macro that begins the 2-column format. See Section 4.4, page 45, for a description and an example of a header file. If this file already exists, make any necessary editing changes.

   Create a trailer file named *group*.trail that contains a .2E macro. This macro must be present to end the 2-column formatting.

3. Extract the explanations from the message text file; place them in a separate file.

   Use the catxt(1) utility to extract the explanations (text tagged with $nexp) from the message text file. Place the resulting text in a file named *group*.nexp. For example, the following command extracts the explanations for the data migration messages (group code dm) from the message text file dm.msg and places the result in the dm.nexp file:

   ```
   catxt -n dm.nexp dm.msg
   ```

   If you use symbolic names, you must use catxt with the -s option. For example, the following command extracts the explanations for the data migration messages (group code dm) from the message text file dm.msg; replaces the symbolic names with numbers based on a list contained in an include file that is specified in the dm.msg file; and places the result in the dm.nexp file:

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

   See Section 2.2.1.1, page 7, for a discussion of working with symbolic message names.

4. Print the message section by using text processing utilities on the Cray Research system or on your front end. Print the head, explanation, and trail files by using one command.

   See Section 4.5.2, page 49, for a sample command line to print the message document.

5. Repeat steps 1 through 4 until you are satisfied with the output.

6. Build the explanation catalog from the message text file by using the -c and the -e options of the caterr(1) utility.

The `caterr -c` utility builds binary catalogs that the run-time parts of the message system use. Specifically, a command of the following form builds an explanation catalog called *group*`.exp` from a message text file named *group*`.msg`:

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

For more information about the `caterr` utility, see the man page for the `caterr` utility or Section 4.6, page 49.

7. Set your `NLSPATH` environment variable to point to the explanation catalog you created in the previous step.

   The `NLSPATH` environment variable gives the file name of the catalog that the message system uses to look up explanations. The last node of the path name you specify with this variable must be `%N.cat`.

   For example, if the explanation catalog `dm.exp` (created in the previous step by using the `caterr` utility) is in the `/home/messages/dm` directory, set your `NLSPATH` environment variable as follows:

   ```
   setenv NLSPATH /home/messages/dm/%N.cat
   ```

   For more information, see Section 4.6.2, page 51.

8. Test the explanation catalog by requesting to view the explanations with the `explain`(1) utility.

   For example, to view the explanation for the message with the ID `dm-100`, issue the following command:

   ```
   explain dm100
   ```

   For more information on the `explain` utility, see Section 4.6.3, page 51.

9. Correct any problems in the explanation catalog by editing the message text file and building a new explanation catalog.

10. Repeat steps 6, 8, and 9 until you are satisfied that the online and printed message information is complete, consistent, and correct.

4.2 Message style definition

The following subsections provide information about the style used in message documentation. The message style consists of the following elements:

- Page layout
- Section heading

- Font and point size usage

Section A.2.2, page 63, contains an example of a document section formatted according to the message style.

4.2.1 Page layout

Messages are printed in a 2-column format. The page is divided into 2 columns that are 3.3 inches wide. A 0.25-inch gutter separates the columns vertically. A message ID bar appears at the beginning of each message. This bar, set off by horizontal lines, gives the message ID. The message ID consists of the group code, a dash, and the message number. Message text is in 9-point type.

4.2.2 Section heading

The section title of a message section is formatted the same as a section title in most Cray Research manuals. Introductory information that appears after the title and before the first message begins 1.375 inches below the bar under the section title. This text is in 11-point type and spans the width of the page. (The message style does not use the standard Cray Research publications modified 2-column format.)

4.2.3 Font and point size usage

The message system uses fonts in a manner consistent with the style used in other Cray Research manuals. Different font sizes are used because of the two-column format. Fonts are used as follows:

Font	Description
New Century Schoolbook	The default font. The body of the message explanation is set in 9-point New Century Schoolbook.
`Courier`	Used for all literals, including the copy of the error message that appears in the message explanation. Literals in the body of the explanations (commands, options, file names, and so on) are also in Courier.

Italic Used to denote variables. Variables in messages are discussed in greater detail in Section 4.3.4, page 43.

4.3 Message markup

The message style is achieved by using a set of `nroff`(1) and `troff`(1) macros called *message macros*. The following subsection describes how to use these macros to mark up a section of message documentation correctly.

The markup of a message section is different from that of other `nroff` and `troff` documents, because the message source file is processed not only by `nroff` and `troff`, but also by the `caterr`(1) and `catxt`(1) utilities. Because one of these utilities always processes the message text file before it is piped to `nroff` or `troff`, you must mark up the file to be acceptable to these utilities. This markup is then changed by the utility to be acceptable to `nroff` and `troff`.

4.3.1 Message text file

The message text file contains the marked-up messages and explanations. This file consists of messages, explanations, and comments. Each type of information is denoted by the use of a tag in the file. Messages are tagged with the string `$msg`. Explanations are tagged with either string `$nexp` or `$exp`. Comments are tagged with a dollar sign ($), followed by a space, tab, or carriage return.

4.3.2 Messages

Each message in the file must conform to the following rules of formatting:

- Begins with the string `$msg` followed on the same line by the message number (or symbolic name).
- Appears as a single logical line of text. If the message occupies more than one physical line, each physical line except the last must end with a continuation character (\) to make the message one logical line.
- Conforms to the guidelines for good messages as outlined in Appendix A, page 55.

The following example shows a message in the message text file that is identified by the `msg` tag and a message number:

```
$msg 6 The daemon is unable to migrate the file.
```

4.3.2.1 Symbolic message names

Instead of using a number to identify each message, you can use a symbolic name. For example, the following message is written using the name DGR_UTM to identify the message in place of the number 6.

```
$msg DGR_UTM The daemon is unable to migrate the file.
```

A header file is used to create a mapping between the symbolic names and the message numbers. For example, the following line could be used in a header file to map the name DGR_UTM to the number 6. The comment indicates the content of the message.

```
#define DGR_UTM  6  /* unable to migrate file*/
```

The name of the header file that contains the mapping must appear in the message text file so that the C language preprocessor (cpp(1)) can replace the symbolic names with the associated numbers before the catalog is generated by the caterr utility. For example, the following line is needed in the dm.msg file to include the dm_msg.h header file:

```
#include "dm_msg.h"
```

For information about printing a message text file that uses symbolic names, see Section 4.5, page 48. For information about creating an explanation catalog from a message text file that uses symbolic names, see Section 4.6, page 49.

4.3.3 Explanations

Each explanation in the file must conform to the following formatting rules:

- Begins with the string $nexp, which must be followed on the same line by the message number (or symbolic name).
- Contains a copy of the message. Like the message itself, the copy of the message must appear as one logical line. If the message occupies more than one physical line, each physical line except the last must end with a continuation character (\) to make the message a single logical line.
- Contains a .PP macro after the copy of the message and before the body of the explanation.
- Contains any of the macros and strings defined on the msg(7D) man page for use within explanations.
- Uses fonts and point sizes as described in Section 4.2.3, page 40.
- Ends with a .ME (message end) macro.

The following example illustrates the markup of one message and explanation pair. A symbolic message name is used in this example.

```
$msg DGR_URH The specified file is not a migrated file.
$nexp DGR_URH
The specified file is not a migrated file.
.PP
The data migration daemon received a request that applies
only to migrated files, but the requested file is not
migrated.  This error usually indicates that the file's
status has been changed by a process other than data
migration.  Perform an \*Cls -l\fR command and examine
the first character in the entry for the file.  If that
character is an "m", the file is migrated.  Inform your
system support staff.
.ME
```

4.3.4 Variables

Many error messages contain variables that contain contextual information when they are issued to the user. For example, the following messages each contain a variable that is supplied at run time:

```
A .keep file is not present for user ID 'mike'.

An attempt to allocate 512 bytes has failed.

Required option -t not specified.
```

In the first message, the user name at the end of the message is a variable. In the second message, the number of bytes is a variable. The option in the third message is a variable.

These messages would appear as follows in the message text file:

```
$msg 100 A .keep file is not present for user ID '%s'.

$msg 200 An attempt to allocate %d bytes has failed.

$msg 300 Required option -%s not specified.
```

Use single quotation marks (' ')around user-supplied strings that are referred to as tokens. Examples of such strings include file names and user IDs. The use of quotation marks highlights the literal information specific to the situation and reduces the possibility of variables being interpreted with a literal meaning. The user

ID in the first example in the preceding messages is in quotation marks because it is a token that is read from the user's environment. The quotation marks help the user to understand that the value should not be confused with standard text.

Not all variables should be enclosed in single quotation marks. Numbers do not need to be quoted (see the second example in the preceding messages). Strings that are to be interpreted literally by the user do not need to be in quotation marks. For example, the option in the third example message does not require quotation marks.

In situations where single quotation marks are used, it is necessary to precede the leading quotation mark with the `troff` string \&. This code protects the string from interpretation by `troff` as the beginning of a comment.

In the explanation of messages that contain variables, it is not possible to show the string or value that users see when they receive the message from the program. That value is unknown until the error occurs.

When you mark up the explanation, choose a word that indicates the nature of the information to be supplied at run time. Put that word in the message in place of the C language variable designator. Knowing the type of the variable will help you to choose an appropriate variable name. The most common variable designators and their variable types are listed in the following table:

Table 3. C language variable designators

Character	Type
%d or %i	Signed decimal
%f	Floating point
%o	Unsigned octal
%s	String (character pointer)
%x	Unsigned hexadecimal

Place the variable name in italic font to indicate that it does not appear literally in the message. Use the *V string to change to italic font (instead of the \fI string). The *V string improves the spacing between the Courier and the italic words.

For example, the markup of the three messages shown previously and their explanations might appear as follows:

```
$msg 100 A .keep file is not present for \&'%s'.
$nexp 100
A .keep file is not present for \*Vuser\fC.
.PP
```

```
The \*Cdmlim\fR\*(11 command did not find a file named \*C.keep\fR in
the home directory of the specified user.  To exempt files from
migration, you must create a file named \*C.keep\fR in your home
directory.  It should contain the names of the files that you wish
to exempt from migration.  The file names in this file may contain
standard wildcard characters.
.ME
$msg 200 An attempt to allocate %d bytes has failed.
$nexp 200
An attempt to allocate \*Vnumber\fC bytes has failed.
.PP
The command was unable to allocate additional memory.  This message
indicates that your run-time memory allocation is too small to
process the command.  Your system support staff may be able to
increase your run-time memory limit.
.ME
$msg 300 Required option -%s not specified.
$nexp 300
Required option -\*Voption\fC not specified.
.PP
The \*C-t\fR and \*C-j\fR options to the \*Cdmdjournal\fR command
are required.  See the man page for a complete description of the
options to the \*Cdmdjournal\fR command.
.ME
```

4.4 Header and trailer files

Header and trailer files contain macros that are needed for the printed version of the messages, but not for the online version. Because no place exists for these macros in the message text file, they are placed in two special files. The first file is a header file that contains the macros and text that must be processed by `troff` before processing the text of the explanations. The second file is a trailer file that contains a single macro to end the 2-column formatting.

4.4.1 Header file

The header file must contain the following macros and text:

Item	Description
.MT macro	Specifies the title of the manual in which the messages are published. The manual title is printed as the inner page header.
.MN macro	Specifies the manual number of the manual in which the message are published. The manual number is printed as the inner page footer.
.CF macro	Specifies the text that you want to have appear as the center page footer.
.GC macro	Specifies the group code for your messages. The argument to this macro appears in the message ID bar as part of the message identifier.
.ST macro	Specifies the title of the section you are formatting. The section title appears in large type at the beginning of the section and as the outer page header on all succeeding pages of the section.
Intro text	Text that explains the content of the section. This text usually includes the following information: • List of the programs, commands, or routines from which the messages documented in the section are issued. • Significance of the message numbers. For example, if three commands share a catalog, a block of numbers may be assigned to each of the three commands. Thus, messages 1 through 999 are for command 1, messages 1000 through 1999 are for command 2, messages 2000 through 2999 are for command 3. If your messages numbers are divided in this or any other significant way, describe the division in the introductory text. • Sources of additional information about the product or feature.
.2S macro	Starts 2-column format. This macro must be the last macro in the header file.

The following example shows the source markup of a message header file.

```
.MN "SG\-9999"
.MT "\*u Message System Exmple Manual"
.CF "Cray Research, Inc."
.GC "dm"
.ST "\*(Cbdm\fR Messages"
This section documents all error messages issued by the data migration
```

```
feature of UNICOS.  The group code for this feature is \*Cdm\fR.
Each message is listed, along with an extended explanation.  The messages
are arranged by message number, in ascending order.
.PP
The message number helps to indicate the part of data migration that is
issuing the error.  The message numbers are assigned as follows:
.CH 12 "Range" "Message source"
.TL
1\-99
Data migration daemon
.TL
100\-199
\*Cdmget\fR(1) command
.TL
200\-499
\*Cdmput\fR(1) command
.TL
500\-999
\*Cdmlim\fR(1) command
.TL
1000\-1299
\*Cdmdalter\fR(8) command
.TL
1300\-1499
\*Cdmdjournal\fR(8) command
.TL
1500\-1599
\*Cdmdstat\fR(8) command
\&.TL
1600\-
Any data migration command
.PP
The explanation that accompanies each message describes the error in greater
detail and suggests actions for solving the problem.  The explanation may
refer you to documentation that discusses topics related to either the
problem or the solution.  You can also refer to \fIUNICOS System Administration\fR,
publication SG\-2113, for a description of the data migration feature, its
configuration, and its administration.
.2S
```

4.4.2 Trailer file

The trailer file must contain one 2E macro to end 2-column format.

4.5 Extraction and printing

When you have a properly marked-up message text file, you are ready to extract the explanations from it and print the resulting message section.

The following subsections provide detailed steps that you must perform to produce a hard-copy message section.

4.5.1 Extracting the explanations

The message text file contains messages and explanations. Because the documentation that is being produced includes only the explanations, a process is required for extracting the explanations from the message text file. Use the `catxt`(1) utility to perform this extraction.

The syntax of the `catxt` utility is as follows:

```
catxt -n outfile [-s[-P cpp_opts]] infile
```

For example, if you have a marked-up message text file called `dm.msg`, you could extract the explanations from that file and put them in a file called `dm.nexp`, using the following command:

```
catxt -n dm.nexp dm.msg
```

It is a convention to use the suffix `nexp` for explanation files.

If the message text file, `dm.msg` in this example, contains symbolic message names, use the `-s` option as shown in the following command to call `cpp`(1) to map those names to numbers.

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

For more information about using `catxt`, see the `catxt` man page.

In addition to extracting the explanations, the `catxt` utility also replaces the `$nexp` tag with a `.MS` macro (message start). While it is performing this replacement, `catxt` checks that every `.MS` macro has a corresponding `.ME` macro. This pairing is required to ensure proper printing.

If `catxt` finds a missing `.ME` macro, it issues a warning message. The following is an example of the warning message:

```
***WARNING: nexp number 38 does not
have an ending ".ME"***
```

If you receive this message, add a .ME macro to the indicated message in the input file and rerun catxt. Do not make the correction in the output file. All corrections must be made to the message text file so that they are propagated to the online version of the messages, as well as to the hard copy.

4.5.2 Printing the explanations

When you have created the following files, you are ready to print your hard-copy message section:

- Header file (see Section 4.4.1, page 45, for the recommended content of this file)
- Explanation file (see Section 4.5.1, page 48, for a discussion of the utility to create this file)
- Trailer file (see Section 4.4.2, page 47, for the recommended content of this file)

Use the troff(1) text formatting utility to print these files as a hard-copy message section. The troff utility on Cray Research systems is a device-independent text processor (ditroff) that produces output suitable for a PostScript laser printer.

Use the -msg option of the troff utility to use the message macro definitions (see msg(7D)) during text formatting. The troff output can be printed using the lpr(1B) utility with the -n option. The -n option identifies ditroff as the source of the input to lpr.

For example, to print the data migration messages from the dm.head header file, dm.nexp explanation file, and dm.trail trailer file, use the following command:

```
troff -msg dm.head dm.nexp dm.trail | lpr -n
```

Appendix B, page 65, contains an example of the data migration message section that is printed as the result of these commands.

4.6 Testing online explanations

To test that a message text file produces a working explanation catalog, perform the following steps:

1. Build the explanation catalog from the message text file by using the -c and -e options of the caterr(1) utility. (In addition, use the -s option if your message text file uses symbolic message names as described in Section 2.2.1.1, page 7.)

2. Set your NLSPATH environment variable to point to the directory that contains the explanation catalog that you created in the first step.
3. View the explanations in the catalog by using the explain(1) utility.

The following subsections describe these three steps in greater detail.

4.6.1 Building the explanation catalog

The online explanations that a user sees are drawn from a file called the *explanation catalog*, which is a binary file built from the message text file. The explanation catalog is in a form that can be read by the explain(1) utility, which users use to retrieve an online explanation.

The caterr utility builds the explanation catalog from the message text file. Use the -e and -c options of caterr to build an explanation catalog. The -e option specifies that you are building an explanation catalog and not a message catalog. The -c option lets you specify the name of the explanation catalog to be built.

In addition, use the -s option if the message text file contains symbolic message names. The -s option calls the C preprocessor (cpp(1)). The cpp utility, using the include file referenced in the message text file, replaces the symbolic names in the file with the appropriate numbers. (For more information on using symbolic names, see Section 2.2.1.1, page 7.)

The syntax of the caterr utility with these options is as follows:

```
caterr [-s] -e -c catfile infile
```

The *catfile* argument specifies the name of the catalog to be output, and *infile* specifies the name of the message text file to be read as input.

The following example builds an explanation catalog named dm.exp from the message text file dm.msg:

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

The following example builds an explanation catalog named dm.exp from the message text file dm.msg, which contains symbolic message names:

```
caterr -s -e -c dm.exp dm.msg
```

4.6.2 Setting the NLSPATH variable

After you have created an explanation catalog, you must tell UNICOS the location of the catalog by setting the NLSPATH environment variable. This variable gives the search path that the explain utility (used in the next step) uses to find an explanation catalog.

The NLSPATH environment variable must be of a very specific format to work correctly. The file name (last part of the path name after the final slash) must be (literally) %N.cat. (When UNICOS parses this string, the %N is replaced by the group code of the message.)

For example, to test explanations in a catalog named dm.exp in the /home/messages/dm directory, set the NLSPATH variable to the following value:

```
/home/messages/dm/%N.cat
```

In the standard shell, the following statements set NLSPATH to include this string and export the value of NLSPATH:

```
NLSPATH=/home/messages/dm/%N.cat
export NLSPATH
```

In the C shell, the following statement sets NLSPATH to include this string:

```
setenv NLSPATH /home/messages/dm/%N.cat
```

The explain utility reads the NLSPATH variable to determine the name of the message catalog for the file (of the form *group*.cat). It then substitutes the .exp suffix for the .cat suffix in the message catalog name to determine the explanation catalog name. The utility tries to open the file to retrieve the explanation.

4.6.3 Viewing the explanations

Verify that the explanations are displayed correctly for the user by viewing the explanations through the explain utility. Issue the explain utility once for each explanation you want to view.

For example, to view the explanation for the message with the ID dm-100, issue one of the following commands:

```
explain dm100

explain dm-100
```

The explain utility retrieves the explanation and displays it. The following example illustrates how such a session might appear on your screen.

```
% explain dm100
A .keep file is not present for 'user'.

The dmlim(1) command did not find a file named
.keep in the home directory of the specified user.
To exempt files from migration, you must create a
file named .keep in your home directory.  It should
contain the names of the files that you wish to
exempt from migration.  The file names in this file
may contain standard wildcard characters.
```

Issue successive `explain` commands until you have tested at least a representative sample of the explanations in the file. Always test the first explanation and the last explanation.

Check the output for correct line breaks, highlighting and underlining (if your terminal is enabled to display them), and completeness of the text (no text missing).

If you find a problem with the catalog, return to the message text file, edit it to eliminate the problem, regenerate the explanation catalog, and test the explanation again. Remember that any change to the message text file is propagated to both the online explanation catalog and the hard-copy message section. After you make a change, check both outputs. Repeat this process until you are satisfied that the message text file is producing usable online text and hard copy.

4.7 Troubleshooting

Table 4 lists common problems that you might encounter when working with the message macros and procedures. The cause of these problems is identified.

Table 4. Common formatting problems and their solutions

Problem	Cause
Output is formatted incorrectly; only one word appears on each line.	Verify that there is a `.2S` macro in the *group*`.head` file. The absence of this macro will cause this problem.
Only part of the copy of the message that appears in the explanation is in Courier font; the remainder is in New Century Schoolbook.	You have used a `\fR` font instead of a `\fC` font after changing fonts for a variable in the copy of the message. Change the `\fR` font code to `\fC`.
The message font changes from Courier to New Century Schoolbook, a blank line is inserted, and the point size gets smaller, even though no font code appears in the source file.	The copy of the message exceeds one logical line. Either join the lines into one or use a continuation character (\) at the end of each physical line except the last one.
A message or explanation is truncated where a variable occurs in the text. Some portion of the source text is missing in the output.	Check for an unprotected single quotation (`'`). Single quotation marks that appear after a space must be preceded by the characters backslash ampersand (\&) to protect them from interpretation by `troff` as the beginning of a comment.
No header bar is appearing on the first page of the section and many messages are being continued over columns and pages.	Either you do not have a header file or your header file does not contain a `.ST` macro. Create a header file and include a `.ST` macro or add a `.ST` macro to your existing header file.