Guidelines for Messages and Explanations [A]

A message consists of two parts: the message that is printed for the user each time the error occurs, and an expanded explanation of the error condition that appears in the message documentation, both online and in the printed manual.

The message and the explanation should both be clear, concise, and focused; however, the message and the explanation may speak to different audiences.

The message is directed to any user who might encounter the error that the message describes. It should contain a brief description of the problem in unambiguous terms.

The explanation is directed to the user who cannot resolve the error using only the information in the message. This user has sought additional information. The explanation should give a more complete description of the problem, suggest actions that will help resolve the problem, and direct the user to sources of information related to the problem and its resolution.

The following subsections give guidelines for writing messages and for writing message explanations.

> **Note:** These guidelines are intended for Cray Research, Inc. software developers who are writing messages for code included in Cray Research, Inc. software releases. Others may want to follow the guidelines to improve the general usability of their messages and explanations.

A.1 Guidelines for messages

A good message provides a user with specific information about the problem that the software has encountered. It conveys the context in which the problem occurred and, when possible, states the problem in a way that implicitly suggests a corrective action. A good message also is written with an awareness of the attitude that is expressed toward the user.

The challenge of writing good messages is conveying as much information as possible concisely. A good rule of thumb for messages is that users who are past the initial learning phase for a product should be able to recognize and correct the problem by using only the information in the message. The explanation exists for users who are new to the product.

Good messages demonstrate the following characteristics:

- Clearly stated

- Specific about the problem
- Respectful of users
- Grammatically correct

These characteristics are important for the usability of the messages in English, and they also improve translatability.

A.1.1 Clear messages

Clear messages state the problem as simply as possible without the use of specialized terminology. A clear message is unambiguous in its description of the problem.

Observe the following guidelines to make messages clear:

1. When describing a problem, use plain English instead of terms familiar only to a limited audience (for example, UNIX system terminology).

   For example, the following message is clear to a programmer familiar with the `stat`(2) system call, but it is not clear to most users:

   ```
   Cannot stat file
   ```

   The following message is preferred:

   ```
   Cannot get the status (with stat(2)) of
   the 'filename' file.
   ```

2. Choose message syntax carefully. Avoid long strings of modifiers.

   For example, the following message:

   ```
   bad swap superblock magic number
   ```

   could be worded more clearly as follows:

   ```
   The checkword number in the swap superblock
   is incorrect.
   ```

3. Use worded explanations rather than programming-language expressions.

   For example, the following message:

   ```
   end bp forw != NULL
   ```

   would be clearer if rewritten as follows:

   ```
   The I/O chain for the pty/tty device has
   failed an internal consistency check.
   ```

Follow these principles wherever possible. Remember that users do not know as much about the system, the program, or the origin of the error as you do.

A.1.2 Specific messages

Messages that are specific give users all of the information needed to correct the problem. Observe the following guidelines to make messages specific:

1. Identify the problem specifically, rather than in a general sense.

   For example, the following message is very vague:

   ```
   I/O error
   ```

   Instead, give information that is definite enough to point to a corrective action.

2. Explain the problem from the user's perspective rather than the system's perspective.

   For example, the following message:

   ```
   No device response
   ```

   would be better stated as follows:

   ```
   Cannot access the device you selected.
   ```

3. Include information specific to the situation.

   Instead of the following message:

   ```
   Terminating job
   ```

   it would be better to write:

   ```
   Terminating job 'job-identifier'.
   ```

4. Include information pertinent to the solution of the problem; do not force users to guess arbitrary limits.

   The following message:

   ```
   Identifier too long
   ```

   would be better stated as follows:

   ```
   The identifier must consist of 14 characters
   or less.
   ```

A.1.3 Respectful messages

The message should respect users and the situation. Respect is shown by adhering to guidelines, as follows.

1. State the problem neutrally or as a deficiency of the system rather than blaming the user for the problem.

   For example, the following message:

   ```
   Illegal expression 'exp' has been specified
   in the input file
   ```

   could be stated more neutrally as follows:

   ```
   Cannot accept the expression 'exp' in the
   input file.
   ```

2. Avoid unnecessarily hostile, violent, or threatening terminology. Terms such as *catastrophic*, *abort*, *illegal*, *kill*, *abandon*, and *disastrous* have a negative effect on users. Rephrase messages containing such words to be more neutral and less threatening.

   It can be difficult to follow this guideline in situations in which accepted UNIX terminology requires the use of a "hostile" word to provide an accurate technical description of the situation. For example, UNIX uses the term *kill* in a technical sense to describe forced job termination. In such a situation, it may not be possible to avoid the term. However, whenever it is within your control, avoid overly dramatic terminology.

3. Avoid introducing attempts at humor.

   Humor in messages has many dangers. Everyone's idea of what is funny differs, especially across cultures. Messages often occur repeatedly, and the humor wears off quickly. Therefore, the best policy is to avoid making messages humorous or cute.

A.1.4 Grammatical messages

A grammatical message is phrased as a complete sentence, is punctuated according to standard usage, contains no truncated words, uses conventional spelling, and contains articles, auxiliary verbs, and prepositions as dictated by standard English usage. Messages written in standard English are less likely to be misinterpreted and can be translated more accurately into foreign languages.

Observe the following guidelines to make messages grammatical:

1. Use capitalization in a standard way; start the message with a capital letter, and capitalize words and abbreviations as they would appear in narrative text.
2. Avoid beginning messages with a special character or a variable. Special characters and variables at the beginning of messages make them difficult to index.
3. Use punctuation in a standard way; include commas and semicolons when appropriate, and end the message with a period.
4. Observe Cray Research trademark and style conventions when using industry terms. Consult a writer if you have questions regarding Cray Research style conventions.
5. Spell out words completely.

   The space and time saved by writing `max` instead of `maximum` is not worth the lack of clarity it creates in the messages. Use only abbreviations that are very widely understood by users of Cray Research systems. For example, it is sensible to use IOS and OWS in messages instead of I/O subsystem and operator workstation. However, it is inappropriate to use MTU for maximum transmission unit in a message whose audience is the end user.
6. Write messages as complete sentences; include a verb and all the needed articles (a, an, the), prepositions, and auxiliary verbs.

The following example illustrates the intention of these guidelines.

The following message:

```
read error
```

would be more grammatical if phrased as follows:

```
An error occurred during an attempt to read
the 'filename' file.
```

The rewritten message is longer, but it is much less likely to be misunderstood or mistranslated. It is more specific, in addition to being more grammatically correct.

A.1.5 Severity levels in messages

Each message issued to users should have a severity level associated with it. The severity level should indicate to users how important the message is to the success of the job. To help users assess the impact an error has on a job, it is important that you use severity level designations in a consistent manner when you develop software.

The following guidelines apply to message severity levels:

- Indicate the message severity level in uppercase letters (for example, WARNING). Use of all uppercase letters calls attention to the severity level.
- Avoid the use of ERROR as a severity level. Messages that users receive are commonly called *error messages*. Therefore, to designate ERROR as a severity level is uninformative and creates an ambiguity in phrases such as *the error message manual*.
- If it does not conflict with third-party vendor constraints on your code, try to restrict your use of severity levels to the following set:

Level	Description
INFO	The system is communicating information to users, usually about the status of a job or process. An informational message requires no user action because a problem was not encountered.
EFFICIENCY	An inefficient use of the software or the hardware is suspected. Users should examine the code for a better way to perform the process.
CAUTION	A possible problem was detected. The output of the program is still usable, but the results may not be what users expect.
WARNING	A probable problem was encountered. The program continues to process from this point, but the output or the results are likely to be incorrect.
FATAL	A definite problem was encountered. The output produced from this point forward is unusable. Output may be suppressed after this point. Use this level of message when a fatal error is encountered in the input, rather than in the processing. If processing encounters an error that is terminal, issue an unrecoverable error. For example, when a compiler encounters an error in the program source it is compiling that renders further execution of the program useless, issue a fatal error message. But, if the compiler program itself encounters a situation in which it can no longer execute, perhaps because of hardware or system software problems, issue an unrecoverable error. Avoid the use of FATAL as a severity level when possible because, although it is standard in some contexts, *fatal* is an inappropriate word to use as a severity level because it is threatening and overused.

UNRECOVERABLE — An error has occurred that renders further processing impossible. The program terminates immediately.

You should issue this level of message only once and terminate processing immediately. See the description of FATAL for an example of the difference between fatal and unrecoverable errors.

These guidelines and suggested severity levels may not apply to all situations. The most important point to remember is that users rely on the message severity to indicate the nature of the problem. Be as consistent and as accurate as possible with this information.

A.1.6 Substitutable strings in messages

Care should be taken to limit the type of information substituted into messages. Only information that users supply should be substituted into error messages. Examples of user-supplied information include variable names, command-line options, and file names. Building messages from other types of substitutable strings can be a serious impediment to a correct translation.

Consider the following message:

Example 1:

```
Cannot find the file 'filename'.
```

You also may want to apply this message to a situation in which two file names are supplied by the user and neither is found. To cover this situation, you should create a second message that is issued when two files are involved in the error.

The second message might appear as follows:

Example 2:

```
Cannot find the files 'filename' and 'filename'.
```

An alternative in this situation would be to modify the first message to accommodate errors on both one or two files. However, doing so would create a potential for translation errors.

For example, consider the case where you change the message in example 1 to read as shown in example 3 in the message catalog:

Example 3:

```
Cannot find the file%s '%s' %s '%s'.
```

Then you replace the first string with `s` to make the word `file` plural, replace the second string with a file name, replace the third string with `and`, and replace the fourth string with another file name.

The result would be a message that would appear to users to be identical to example 2. However, the translator cannot determine which of the replaceable strings is really a user variable and which is part of the message text. Also, the pluralization of `file` by adding a trailing `s` is correct in English, but it would not be correct in most other languages. The insertion of the connective `and` between the file names might also be incorrect in the target language.

Because of the complexities involved in writing for translation, avoid writing messages to appear in multiple contexts. If you must issue a message that is a variation on the syntax of a similar message, write a new message to cover the variation, rather than try to adjust the first message to accommodate all cases.

A.2 Guidelines for explanations

Message explanations exist for the benefit of users who, upon receiving an error message, cannot resolve the problem without the following additional information:

- A more complete description of the problem
- A suggested course of action to solve the problem

A good explanation contains both types of information. The most natural way to format the information is to describe the problem in the first paragraph of the explanation and recommend solutions in the second paragraph. In some cases, more information will be given than comfortably fits in a paragraph. Add additional paragraph breaks as needed.

The following subsections discuss describing problems and solutions to users in message explanations.

A.2.1 Describing the problem

The message explanation provides a more complete description of the problem than the message itself does. Include the following information in the message description:

- Statement about the cause of the problem
- Context surrounding the cause of the problem
- System or job status affected by the problem

- References to documentation discussing related topics

The following message description states the cause of the problem clearly and outlines the status of the job:

```
The catalog specified for reset was not
defined with the RECOVERABLE attribute.
RESETCAT can reset only recoverable
catalogs. The command is terminated. The
catalog and CRA entries have not been
altered. The workfile has not been defined.
```

The description also could refer users to documentation on the RESETCAT command.

A.2.2 Describing the solution

The solution portion of the message explanation presents courses of action that users can pursue in solving the problem. Many messages result from complex or unknown causes. In these cases, users may have to test several conditions or try several solutions before arriving at one that applies to the problem. Most users realize that this is a fact of life. They do not expect a cure-all to be provided in the message explanation. Rather, they are looking for somewhere to start to solve the problem.

Include the following information in the message solution:

- Suggested problem remedies, listed in the order in which users should try them.
- Parameters, files, permissions, and other configuration information that might be related to the problem. Instruct users to check these items.
- Any steps needed to recover from the problem; for example, if the problem requires that a component be restarted after the problem is located, be sure to instruct the user to perform the restart.
- Possible or likely consequences of various courses of action, especially if those consequences are destructive. For example, if a suggested action might damage or destroy data, you must point that out to users. Do not assume that they know.
- References to documentation that discusses utilities, procedures, or configuration information needed to solve the problem.
- Recommendation to seek help if the problem is not a user-level error. In these cases, direct users to contact the system support staff and state the purpose of the contact. The following phrase can be used in this situation:

```
If none of the suggested actions resolve
the error condition, contact your system
```

```
support staff and request that
fill in an action the support staff should perform .
```

The following paragraph is an example of the solution portion of a message explanation:

```
To recover a nonrecoverable catalog and its
volumes, you must do a synchronized volume
restore of all volumes owned by the catalog.
If you have incorrectly specified the CATALOG
parameter, correct the parameter. If CATALOG
dname was specified, correct the associated
DLBL catalog name. Rerun the command.
```

As with the messages themselves, make the explanations as clear and specific as possible. Try to create a course of action for users that leads to the resolution of the problem.