Message Catalogs:Programmer's Guide

900 Series HP 3000 Computers

HEWLETT
PACKARD

HP Part No. 32650-90021
Printed in U.S.A. 19871101

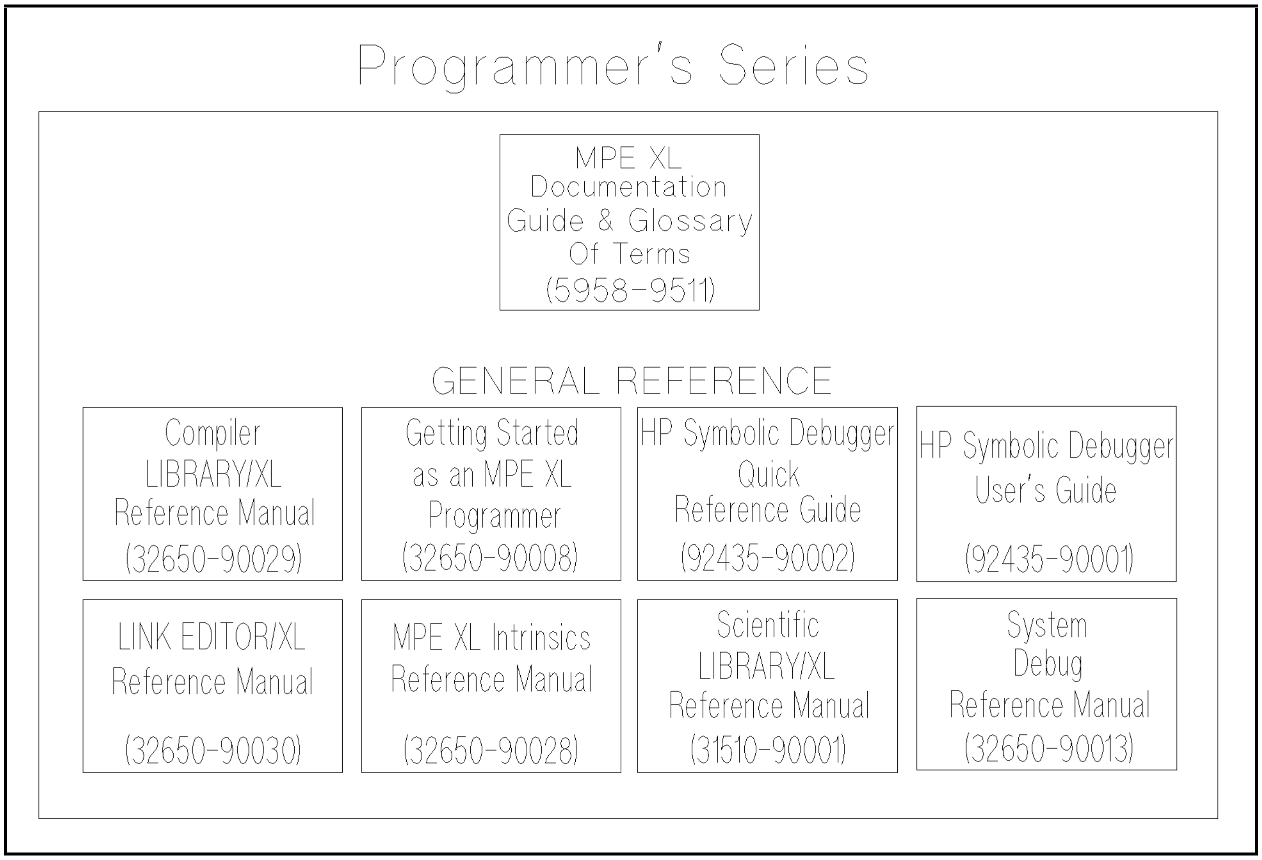
E1187

Printing History

The following table lists the printings of this document, together with the respective release dates for each edition. The software version indicates the version of the software product at the time this document was issued. Many product releases do not require changes to the document. Therefore, do not expect a one-to-one correspondence between product releases and document editions.

Edition	Date	Software Version
First Edition	November 1987	A.01.00

Documentation Map


Programmer's Series
MPE XL Documentation Guide & Glossary Of Terms (5958-9511)
GENERAL REFERENCE
Compiler LIBRARY/XL Reference Manual (32650-90029)
Getting Started as an MPE XL Programmer (32650-90008)
HP Symbolic Debugger Quick Reference Guide (92435-90002)
HP Symbolic Debugger User's Guide (92435-90001)
LINK EDITOR/XL Reference Manual (32650-90030)
MPE XL Intrinsics Reference Manual (32650-90028)
Scientific LIBRARY/XL Reference Manual (31510-90001)
System Debug Reference Manual (32650-90013)

Programmer's Series (con't)

PROGRAMMING TECHNIQUES

Accessing Files Programmer's Guide (32650-90017)	Command Interpreter Access & Variables Programmer's Guide (32650-90011)	Data Types Conversion Programmer's Guide (32650-90015)	Getting System Information Programmer's Guide (32650-90018)
Interprocess Communication Programmer's Guide (32650-90019)	Message Catalog Programmer's Guide (32650-90021)	Native Language Programmer's Guide (32650-90022)	Process Management Programmer's Guide (32650-90023)
Resource Management Programmer's Guide (32650-90024)	SORT-MERGE/XL Programmer's Guide (32650-90080)	Trap Handling Programmer's Guide (32650-90026)	User Logging Programmer's Guide (32650-90027)

Preface

MPE/iX, Multiprogramming Executive with Integrated POSIX, is the latest in a series of forward-compatible operating systems for the HP 3000 line of computers.

In HP documentation and in talking with HP 3000 users, you will encounter references to MPE XL, the direct predecessor of MPE/iX. MPE/iX is a superset of MPE XL. All programs written for MPE XL will run without change under MPE/iX You can continue to use MPE XL system documentation, although it may not refer to features added to the operating system to support POSIX (for example, hierarchical directories).

Finally, you may encounter references to MPE V, which is the operating system for HP 3000s, not based on PA-RISC architecture. MPE V software can be run on the PA-RISC (Series 900) HP 3000s in what is known as *compatibility mode*.

This manual is written for experienced programmers who are inexperienced in working with message catalogs. It is a programmer's guide that gives you step-by-step examples of creating, accessing, and modifying message catalogs. It also explains how to maintain message catalogs and create a HELP facility.

This manual contains only information on MPE/iX message catalogs. For information on MPE/iX intrinsics, refer to the *MPE/iX Intrinsics Reference Manual* (32650-90028). For information on MPE/iX native languages, refer to the *Native Language Programmer's Guide* (32650-90022).

This manual is intended for use by programmers that are developing or maintaining a message catalog. It assumes knowledge of general programming and MPE concepts.

Organization of This Manual

This manual is structured as follows:

Chapter 1	**Introduction** contains an introduction to message macilities on MPE/iX. It introduces the different message facilities.
Chapter 2	**Creating an Application Message Catalog** describes the facility for creating your own catalogs for your applications. It includes discussions on localized applications, `GENCAT.PUB.SYS`, application catalogs, source catalogs, catalog naming conventions, and parameter substitutions.
Chapter 3	**Accessing Application Messages** shows you how to use the catalog intrinsics (`CATOPEN, CATCLOSE,` and `CATREAD`) to obtain messages to use in your applications.
Chapter 4	**Modifying an Application Message Catalog** contains information about expanding formatted files, creating maintenance files, and using collision files.
Chapter 5	**Accessing System Error Messages** describes the System Message facility and how to use system error messages from the two system error message catalogs, `CATALOG.PUB.SYS` and `SYSCAT.PUB.SYS`, for your own use.
Chapter 6	**Creating Your Own HELP Facility** explains how to create a HELP file and format the file with `MAKECAT.PUB.SYS` to make a HELP facility.
Appendix A	**GENCAT Error Messages** lists information on errors associated with `GENCAT`. Included are error numbers, messages, meanings, and corrective actions.
Appendix B	**Maintenance Tasks for Catalogs Formatted With MAKECAT** explains how to use `MAKECAT`, convert programs and message catalogs from those that use `MAKECAT` formatted messages to those that use `GENCAT` formatted messages, and what the differences between `MAKECAT` and `GENCAT` are.
Appendix C	**COBOL Examples** presents the COBOL version of Pascal examples given in the body of the manual.
Appendix D	**FORTRAN Examples** presents the FORTRAN version of Pascal examples given in the body of the manual.

Conventions

`UPPERCASE`	In a syntax statement, commands and keywords are shown in uppercase characters. The characters must be entered in the order shown; however, you can enter the characters in either uppercase or lowercase. For example: `COMMAND` can be entered as any of the following: `command` `Command` `COMMAND` It cannot, however, be entered as: `comm` `com_mand` `comamnd`
italics	In a syntax statement or an example, a word in italics represents a parameter or argument that you must replace with the actual value. In the following example, you must replace *filename* with the name of the file: `COMMAND` *filename*
bold italics	In a syntax statement, a word in bold italics represents a parameter that you must replace with the actual value. In the following example, you must replace filename with the name of the file: `COMMAND(`*filename*`)`
punctuation	In a syntax statement, punctuation characters (other than brackets, braces, vertical bars, and ellipses) must be entered exactly as shown. In the following example, the parentheses and colon must be entered: `(`*filename*`):(`*filename*`)`
underlining	Within an example that contains interactive dialog, user input and user responses to prompts are indicated by underlining. In the following example, yes is the user's response to the prompt: `Do you want to continue? >>  yes`
`{ }`	In a syntax statement, braces enclose required elements. When several elements are stacked within braces, you must select one. In the following example, you must select either `ON` or `OFF`: $\texttt{COMMAND}\left\{\begin{array}{l}\texttt{ON}\\ \texttt{OFF}\end{array}\right\}$
`[ ]`	In a syntax statement, brackets enclose optional elements. In the following example, `OPTION` can be omitted: `COMMAND` *filename* `[OPTION]` When several elements are stacked within brackets, you can select one or none of the elements. In the following example, you can select `OPTION` or *parameter* or neither. The elements cannot be repeated. $\texttt{COMMAND}\ \textit{filename}\left[\begin{array}{l}\texttt{OPTION}\\ \textit{parameter}\end{array}\right]$

Conventions (continued)

`[ ... ]`	In a syntax statement, horizontal ellipses enclosed in brackets indicate that you can repeatedly select the element(s) that appear within the immediately preceding pair of brackets or braces. In the example below, you can select *parameter* zero or more times. Each instance of *parameter* must be preceded by a comma:
	
	`[,`*parameter*`][...]`
	
	In the example below, you only use the comma as a delimiter if *parameter* is repeated; no comma is used before the first occurrence of *parameter*:
	
	`[`*parameter*`][,...]`

`| ... |`	In a syntax statement, horizontal ellipses enclosed in vertical bars indicate that you can select more than one element within the immediately preceding pair of brackets or braces. However, each particular element can only be selected once. In the following example, you must select **A, AB, BA**, or **B**. The elements cannot be repeated. $\left\{\begin{matrix}\texttt{A}\\\texttt{B}\end{matrix}\right\}$ `| ... |`
...	In an example, horizontal or vertical ellipses indicate where portions of an example have been omitted.
Δ	In a syntax statement, the space symbol Δ shows a required blank. In the following example, *parameter* and *parameter* must be separated with a blank: `(`*parameter*`)Δ(`*parameter*`)`
( )	The symbol ( ) indicates a key on the keyboard. For example, (RETURN) represents the carriage return key or (Shift) represents the shift key.
(CTRL)*character*	(CTRL)*character* indicates a control character. For example, (CTRL)`Y` means that you press the control key and the Y key simultaneously.

Contents

Figures

Tables

1 Introduction

Message catalogs are files that contain informative user messages to be output from applications. These messages are numbered and grouped into numbered sets. The messages are accessed with their set and message numbers by your application program and output to users.

This manual tells you how to create, access, and modify your own message catalogs, access system message catalogs, and create your own HELP facility. It takes you step-by-step through the steps needed to work with message catalogs. This manual presents example catalogs and programs to help you create your own system messages and applications.

When to Use Message Catalogs

Using message catalogs to output messages is a convenient and efficient method to create a user interface. You may want to use message catalogs if you:

- Have prompts, error messages, informative messages, etc. that you output to users.
- Plan to localize the application.
- Do not want to recompile each time you alter your messages.
- Want easy access to the messages.

Message catalogs allow you to have a consistent and logical method of outputting messages to the user. The messages are separate from your code; therefore, localization (translation to other languages) is more efficient. Only the message catalog needs to be translated and all the messages are in a defined area. Because the messages are apart from the code, you won't need to retain the source code and recompile when you make changes to the catalog.

The use of message catalogs allows you to create application message catalogs with messages in a user's native language and access these messages programmatically. Messages such as prompts, commands, error messages, or conventions for date and time, can be stored in separate ASCII editor files. As a result, you can create and maintain files without changing the program itself.

Message catalogs are easily used. As shown in Figure 1-1, you open the catalog from your application, read and output messages, then close the catalog.

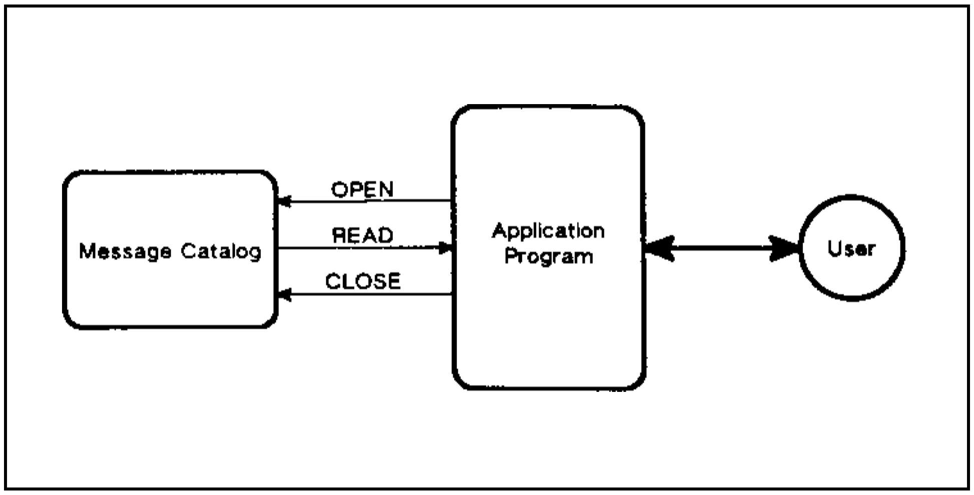


Figure 1-1. Message Catalog Access

MPE/iX supports three message facilities:

- Application Message Facility
- System Message Facility
- HELP Facility

Each is used for a different cataloging task.

Application Message Facility

The Application Message Facility contains:

- `GENCAT.PUB.SYS` utility program
- Catalog intrinsics
- User message catalogs

GENCAT Utility

GENCAT is a menu-driven program that performs various operations on message catalogs. It formats the message catalog for more efficient access, allows you to easily make changes to the catalog, and expands a formatted catalog to its original form. GENCAT can be run interactively or by a job.

Catalog Intrinsics

Three intrinsics are available for accessing application message catalogs: `CATOPEN`, `CATREAD`, and `CATCLOSE`. These intrinsics open the formatted catalog, read and output the messages, and close the catalog. Use of these intrinsics is discussed in Chapter 3.

User Message Catalogs

You create your own catalogs by organizing your numbered messages into numbered sets and then formatting them with GENCAT. Your messages may contain place holders so that at run time, parameters are substituted in the messages. Messages usually consist of error messages, prompts, and informational messages.

Creating, accessing, and modifying application message catalogs is discussed in Chapters 2-4.

System Message Facility

The MPE/iX System Message Facility consists of two message catalogs:

- `CATALOG.PUB.SYS`, which contains Compatibility Mode system error messages.
- `SYSCAT.PUB.SYS`, which contains Native Mode system error messages.

`CATALOG.PUB.SYS` is formatted with the MAKECAT utility. Its messages are accessed with the `GENMESSAGE` intrinsic. `CATALOG.PUB.SYS` contains messages from procedures executing in Compatibility Mode.

`SYSCAT.PUB.SYS` is formatted with the GENCAT utility. Its messages are accessed with the `CATREAD` intrinsic. `SYSCAT.PUB.SYS` contains messages from procedures executing in Native Mode.

You can only access messages in the System Message Facility; you can't modify the files. For example, you can output error messages from your program without creating your own error message catalog.

Accessing system error messages is discussed in Chapter 5.

HELP Facility

The HELP Facility is a special type of message facility. It contains `CICAT.PUB.SYS`, the default `HELP` subsystem catalog that provides users with online help for MPE/iX. `CICAT.PUB.SYS` can be equated to a your HELP file.

HELP catalogs are formatted with the MAKECAT utility and accessed directly by users.

The HELP Facility provides a user interface that allow messages to be accessed very differently from the user's standpoint. The HELP facility interface opens the catalog, reads and outputs messages, and closes the catalog without explicit user operations. The messages are identified without set and message numbers and may not be accessed by user's applications.

Creating your own HELP facility is discussed in Chapter 6.

Program Examples

This manual takes you step-by-step through the processes of: creating, accessing, and modifying application message catalogs (formatted with `GENCAT.PUB.SYS`), accessing system message catalogs, creating your own HELP facility, and maintaining application message catalogs that were formatted using `MAKECAT.PUB.SYS`.

This manual presents many examples; they show you each operation to perform and allow you to base your own actions on them. Complete program examples are shown, as well as program fragments, sample dialogs, and message catalog examples. Example programs are shown in the body of the manual in Pascal; examples in COBOL and FORTRAN are given in the appendices. Although error checking routines are not explained or presented in this manual, they are indicated on program examples. For information on intrinsic error checking, refer to the *MPE/iX Intrinsics Reference Manual* (32650-90028).

Summary

Message catalogs provide a consistent way to access messages such as error messages or user dialogs and are easily created, modified, and accessed. They are separate from the programs that call them, so they are easily modified. For example, translating applications into a native language only requires the message catalog to be translated.

You usually create your own message catalog to output messages from an application; you may use the system error messages if they are appropriate. A special type of message catalog is a HELP catalog. Messages are directly output to you through a predefined interface.

2

Creating an Application Message Catalog

Application messages catalogs are message catalogs that you build, access, and modify. These catalogs contain messages that you output to users from your applications. Application message catalogs are used to organize messages from an application program. These messages may be grouped by type of message (prompt, error message, etc.) and accessed from a message catalog to modularize a program; therefore, all messages output from an application can be placed in a single location separate from the code. The Application Message Facility contains the `GENCAT.PUB.SYS` utility program and the catalog intrinsics, `CATOPEN`, `CATREAD`, and `CATCLOSE`.

Overview

This chapter introduces you to the Application Message Facility and explains how to create a message catalog. Topics include:

- Using Application Message Catalogs
- Introducing the GENCAT Utility
- Creating a Source File
- Formatting a Message Catalog

Using Application Message Catalogs

You use application message catalogs whenever you want to keep application output messages separate from the program code. This separation is done so that messages can be altered without having to recompile the program source. Message catalogs are also used when you intend to create efficient, modularized programs. When the output messages are physically, as well as logically, grouped together they are easy to keep consistent and current.

Application message catalogs are often used for Native Language Support (NLS). When applications are used in many countries and translated into many languages, the messages that are output to the user are translated (localized) into the native language. Text, such as prompts, commands, and messages intended for the user's native language interaction with an application, is stored in message catalogs. Applications are localized without changing the program code. For information about message catalog translation, refer to the *Native Language Programmer's Guide* (32650-90022).

Introducing the GENCAT Utility

`GENCAT.PUB.SYS` is the program that formats, incorporates your modifications, and unformats (expands) the message catalog files. GENCAT is menu-driven with HELP available in each step.

GENCAT has an online HELP facility. To access HELP, enter the index number for HELP from the menu, or a "?" in response to any prompt that does not have a menu selection for HELP.

Explanations of GENCAT error messages are given in Appendix A.

Types of Files

GENCAT uses or creates four types of files:

- Source - You create source files; they contain the sets of messages for your application. Source files are EDIT/V-compatible, ASCII files. Creating a source file is described in this chapter.
- Formatted - These files are created by GENCAT from your source files. They contain a directory to the messages (as well as the messages) for quick access. The files are formatted in binary and are not EDIT/V-compatible. Formatted files are accessed by your application; the application then outputs the messages. Creating formatted files is described in this chapter.
- Maintenance - You create these ASCII files in an EDIT/V-compatible format. Maintenance files contain corrections to your source files, such as adding, deleting, or replacing messages and sets. Creating maintenance files is described in Chapter 4.
- Collision - When you use a maintenance file to modify your source file, GENCAT creates a collision file on your request. This ASCII, EDIT/V-compatible file contains all the changes made to the source file by a given maintenance file. If you use the collision file as a maintenance file on a previously modified source file, you will get the original source. Creating collision files is described in Chapter 4.

GENCAT dialog examples are given throughout this manual. The operations are performed on the example files: `SOURCE`, `FORMAT`, `MAINT`, and `COLISION`. They are source, formatted, maintenance, and collision files respectively. Operations on the example files show you how to use GENCAT for your own message catalogs.

Using GENCAT in Batch Mode

To run GENCAT from a batch job, enter your responses to GENCAT prompts in the `INFO =` string after specifying the execution of the GENCAT utility. Type the GENCAT menu numbers and your file names in the same order as you would interactively. Separate your responses by semicolons. Submit the job in the usual manner.

An example of formatting a source file, named `SOURCE`, to the formatted file, `FORMAT`, with GENCAT is:

```
!JOB Jobname,User/Userpass.Acct/Acctpass,Group/Groupass
!Run GENCAT.PUB.SYS;INFO = "3;SOURCE;FORMAT;0"
!EOJ
```

If GENCAT encounters an error while formatting, expanding, or modifying a catalog, it will abort the job.

For information on formatting with GENCAT, refer to the end of this chapter. For information about creating jobs, refer to the *MPE/iX Commands Reference Manual* (32650-90003).

GENCAT JCWs

GENCAT sets one of three Job Control Words (JCWs) at the conclusion of a formatting, maintenance, or expansion operation: `GCFORMAT`, `GCMAINT`, or `GCEXPAND`, respectively. If the operation completes successfully, the appropriate JCW is set to zero; if it fails, the JCW is set to the GENCAT error number. These errors are listed in Appendix A.

For information about JCWs, refer to *Command Interpreter Access & Variables Programmer's Guide* (32650-90011).

Creating a Source File

A source file is an MPE ASCII file in an EDIT/V-compatible format created in your favorite editor. The catalog contains messages to be output to users. The messages are divided into logical groupings of sets.

This section takes you through the tasks of:

- Specifying comments, sets, and messages with directives.
- Using special characters in your messages.
- Substituting parameters in your messages.

Directives

A source catalog contains directives that partition information in the message catalog and indicate the beginning of each record. Directives are not printed out when the catalog is accessed and always begin in column one. There are three types of directives in a source file:

- `$` to denote comment records.
- `$SET` or `$set` to mark the beginning of a `$SET` record.
- Message numbers to indicate message records.

Blank lines are not allowed in a message catalog.

Comment Records

Comments are used throughout the catalog to document sets and messages, and to make them easier to read. The format of a comment record, where *comment* is an optional string of characters is:

$*comment*

A space between $ and *comment* is optional.

$SET Records

A `$SET` record initiates a logical grouping of messages. Sets break the catalog into manageable segments containing logical groupings of messages (for example, one set of messages for prompts, one set for instructions, one set for error messages). Each set is identified by a set number. Set numbers:

- Identify each set.
- Must be in ascending sequence and unique within the catalog that contains them.
- Do not need to be consecutive.
- Must be greater than 0 and less than 256.
- May have leading zeros.

The format of a `$SET` record where `setnum`, an integer, is the required set number is:

```
$SET setnum comment
          - or -
$set setnum comment
```

The *comment* is an optional text string that must be separated from the set number by a blank. You may use `$SET` or `$set`, but not `$Set` or any variation of mixed cases. Examples of set directives are:

```
$SET 1 Prompts
$
$set 2 Error messages
```

Note the use of a blank comment record for clarity.

Message Records

Message records contain the messages that are output to the user. Message records consist of a message number followed by the message text. This text may be an error message, prompt, or any text which may change according to the language or the country where the program will be used. Message record numbers:

- Identify message locations within a set.
- Must be in ascending sequence and unique within the set that contains them.
- Do not need to be consecutive.
- Must be greater than 0 and less than 32,767.
- Must begin in column one.
- May have leading zeros.

For example, within a set, you can have messages 1-25, 101, 300-332, and 32,766. All of these message numbers can be used again in another set. The format for a message record where `msgnum`, an integer, is the required message number is:

```
msgnum message text
```

The *message text* is an optional character string which follows the message number. If the text is not preceded by a blank, GENCAT replaces the character immediately following the message number with a blank and informs you that a blank has replaced the character. An exception is made if one of two special characters, "%" or "&," follow the message number. These characters will not be replaced by a blank. Their meaning is explained in the following section.

An example of message records, set directives, and comments follows:

```
$SET 1 Prompts
1 ENTER FIRST NAME
2 ENTER LAST NAME
$
$
$set 2 Error Messages
1 NAME NOT ON DATA BASE
2 ILLEGAL INPUT
```

Special Characters

Special characters are used to control the format and content of messages when the messages are output.

- % - allows the message record to remain on several lines when printed out. It breaks up a line by posting a carriage return/line feed before writing the next record. For example:

  ```
  3 AN ERROR OCCURRED DURING THE LOADING %
  OF THE DATA BASE.
  ```

  When this message is accessed, it results in a display of:

  ```
  AN ERROR OCCURRED DURING THE LOADING
  OF THE DATA BASE.
  ```

- & - indicates that the message record is continued on the next line in the source file but is printed out as one line. For example:

  ```
  98 THE NUMBER OF FILES &
  DOES NOT MATCH THE &
  SYSTEM'S CALCULATIONS.
  ```

  When this message is accessed, it is displayed as:

  ```
  THE NUMBER OF FILES DOES NOT MATCH THE SYSTEM'S CALCULATIONS.
  ```

  Note the use of blanks as separators preceding the ampersand.

- ~ - indicates a literal character The special character that follows it is treated as a part of the message and is printed out when accessed. If you want a tilde in your message, you must put two tildes in a row.
- ! - indicates the position for a parameter to be substituted in at run time. This is explained in detail below.

Parameter Substitution

Parameter substitutions are used to insert values that will be known at run time into your messages. An exclamation mark (`!`) is used within a message to indicate where the parameter is to be inserted. `!` is treated as a special character; if you want to use an exclamation point in your message, use the tilde before it. You are allowed up to five parameter substitutions in each message. The parameters that are substituted are strings with an ASCII null as the last character. You may choose positional or numerical parameter substitution. Mixing these two types within a message is not allowed; if you do so, GENCAT will not create a formatted file.

The `CATREAD` intrinsic, shown in the following examples, retrieves a message from the catalog and substitutes the necessary prarmeters. Using `CATREAD` is discussed in Chapter 3.

Positional Parameter Substitution

Positional parameter substitution allows the parameters to be substituted from left to right. The example listed below will demonstrate the use of positional parameter substitution.

Using the following values for parameter substitution:

```
var
  Parm_1 : packed array of CHAR [0:5];
  Parm_2 : packed array of CHAR [0:2];

  Parm_1 := 'MARY';
  Parm_2 := '3';
```

Message 400 in set 13 is:

```
400 INPUT FROM ! ON TERMINAL NUMBER !
```

The execution of the `CATREAD` intrinsic call `... CATREAD ( ... , parm_1, parm_2, ... );` results in the following message:

```
INPUT FROM MARY ON TERMINAL 3
```

Numerical Parameter Substitution

Numerical parameters allow the user to decide where the parameters are to be placed within the message. The exclamation mark (`!`) is immediately followed by a number in the range 1 through 5. Numerical parameter substitution is not generally used. It is useful when you rewrite your messages, or rearrange the parameters in the message, without changing the `CATREAD` call in the application. The example listed below demonstrates the use of numerical parameter substitution and uses the parameter definitions given above.

Message 401 in set 13 is:

```
401 INPUT FROM TERMINAL NUMBER !2 BY !1
```

The execution of the same `CATREAD` intrinsic call as above (`... CATREAD ( ... , parm_1, parm_2, ... );`) results in the following message:

```
INPUT FROM TERMINAL NUMBER 3 BY MARY
```

Example Source Catalog

Notice the `$SET` numbers, message numbers, message comments, and the use of blanks in the example source catalog listed below:

```
$SET 1 Prompts
1 ENTER FIRST NAME
2 ENTER LAST NAME
$
$
$set 2 Error Messages
1 NAME NOT ON DATA BASE
2 ILLEGAL INPUT
3 AN ERROR OCCURRED DURING THE LOADING %
OF THE DATA BASE.
98 THE NUMBER OF FILES &
DOES NOT MATCH THE &
SYSTEM'S CALCULATIONS.
$
$set 13 Run-Time Messages
400 INPUT FROM ! ON TERMINAL NUMBER !
401 INPUT FROM TERMINAL Number !2 BY !1
```

The source catalog given above is used in further examples in this manual and denoted by the file name, `SOURCE`.

Naming Convention

Before a source file is formatted by the GENCAT utility, you must decide how you are going to name your formatted catalog. Catalogs that are not going to be localized may use any valid MPE/iX file name. Catalogs that are going to be localized should take some guidelines into consideration.

An application that has been localized into more than one language will typically have a separate message catalog for each language. A naming convention facilitates using different localized versions of files required by an application program.

Each native language supported by Native Language Support (NLS) has a three-digit language ID number. This ID number can be used as the last three characters of each catalog file name to identify each localized catalog. This convention is used when the `NLAPPEND` intrinsic selects the message catalog for the local language at run-time.

The `NLAPPEND` intrinsic is called to concatenate the language ID number and a generic five-digit catalog file name to form the name of the catalog that is opened.

For example, an original unlocalized message catalog is APCAT000. The message catalog in German would be APCAT008 because the language ID for German is 8. `NLAPPEND` concatenates 008 to APCAT to create the name of the catalog that is opened with `CATOPEN`.

Refer to the *Native Language Programmer's Guide* (32650-90022) for a complete list of native languages and their corresponding language ID numbers. Run the NLUTIL utility to see the languages that are available on your system.

Formatting a Source File

You must format the source catalogs so the catalog intrinsics can access them. GENCAT formatted message files are binary and cannot be edited. Formatting compacts the files and creates a directory, which saves disc space and reduces access time. As shown in Figure 2-1, GENCAT creates a formatted catalog from a source file.

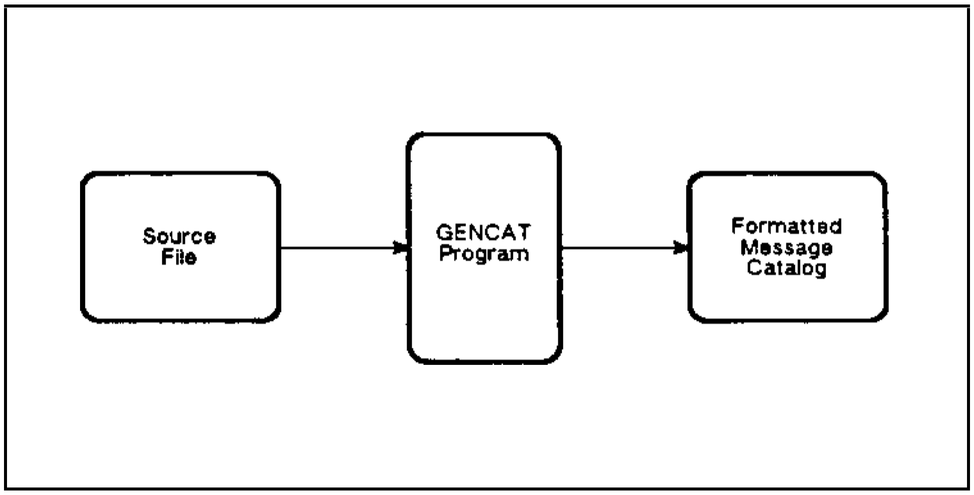


Figure 2-1. Formatting a Source File With GENCAT

During the formatting process, GENCAT verifies that:

- All directives are legal and used correctly.
- Set numbers are in ascending order.
- Set numbers are greater than 0 and less than 256.
- Message numbers are in ascending order within each set.
- Message numbers are greater than 0 and less than 32,767.
- Continuation and concatenation characters are correct.
- Parameter substitution characters are used correctly.

The dialog listed below is an example of formatting a source catalog:

```
:RUN GENCAT.PUB.SYS

HP32414A.02.01 GENCAT/3000 (C) HEWLETT-PACKARD., 1983

ENTER INDEX OF DESIRED FUNCTION

    0.  EXIT.
    1.  HELP.
    2.  MODIFY SOURCE CATALOG.
    3.  FORMAT SOURCE INTO FORMATTED CATALOG.
    4.  EXPAND FORMATTED CATALOG INTO SOURCE.
```

```
        >>3

        ENTER NAME OF SOURCE FILE TO BE FORMATTED

        >>SOURCE

        FORMATTING...

        ENTER NAME FOR NEW FORMATTED FILE

        >>FORMAT

        TOTAL NUMBER OF SET FORMATTED = 3
        TOTAL NUMBER OF MESSAGES FORMATTED = 8

        FORMATTING SUCCESSFUL
```

The formatted catalog given above is used in further examples in this manual and denoted by the file name, `FORMAT`.

Summary

To create an application message catalog, you first produce a source file that contains directives (comment records, set records, and message numbers), messages, and special characters (`&`, `%`, `~`, and `!`).

Messages are identified by message and set number and may have run-time parameters inserted when accessed.

After deciding on a naming convention, if you have Native Language Support (NLS) considerations, the source message catalog is then formatted by the GENCAT utility. The GENCAT program can be run by a job and sets JCWs to indicate success or failure of operations.

The formatted message is accessed by the application program which then inserts run-time parameters and outputs messages to the user. This access is described in Chapter 3.

3

Accessing Application Message Catalogs

After you create a formatted message catalog with the GENCAT utility, you access the formatted catalog from your application with system intrinsics, as shown in Figure 3-1. To read and output the messages, you must first open the catalog. Messages may be output to a file or a buffer. If run-time parameters are specified, they are substituted when messages are output. Before your program finishes, you must close the catalog.

Overview

This chapter explains how to access your application message catalog and presents an example of accessing the example catalog, `FORMAT`, that was created in the last chapter. Topics include:

- Opening a Catalog - using the `CATOPEN` intrinsic
- Reading Messages - using the `CATREAD` intrinsic, specifying parameters, and outputting messages
- Closing a Catalog - using the `CATCLOSE` intrinsic
- Example of Accessing an Application Message Catalog

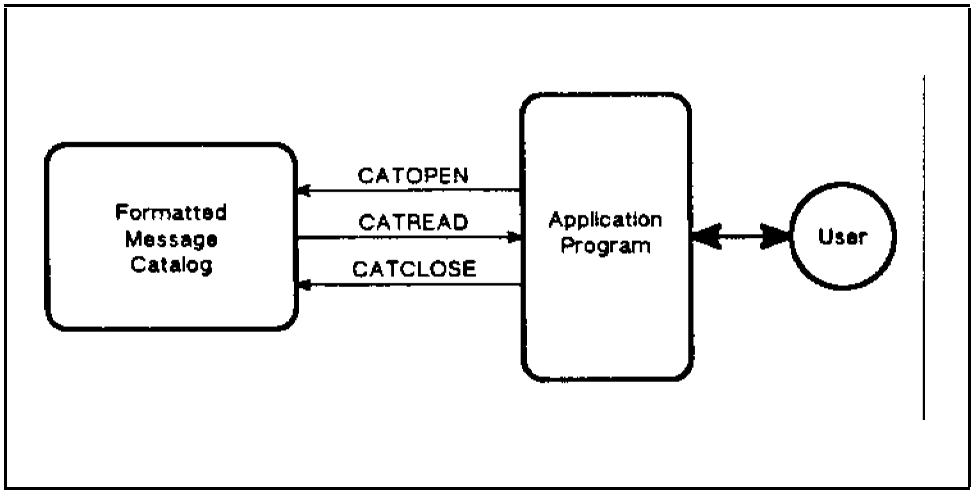


Figure 3-1. Accessing an Application Message Catalog

Opening a Catalog

The `CATOPEN` intrinsic opens an application message catalog. The syntax for the `CATOPEN` intrinsic is:

catindex := CATOPEN (**formaldesignator, catstatus**);

The *catindex* is a value returned by the `CATREAD` intrinsic used to identify the message catalog being accessed. This number is not the same as the file number. The *formaldesignator* parameter contains the string that identifies the catalog file to open; *catstatus* returns the error number. It tells if the call is successful. An example of opening the example catalog, `FORMAT`, would be done as follows:

```
var
  Designator : packed array [1..7] of CHAR;
  Catstatus  : packed array [1..2] of SHORTINT;
  Catindex   : INTEGER;

  Designator := 'FORMAT ';
  Catindex := CATOPEN (Designator, Catstatus);
```

For detailed information about the `CATOPEN` intrinsic, refer to the *MPE/iX Intrinsics Reference Manual* (32650-90028).

Reading Messages

The `CATREAD` intrinsic reads the message specified by the set and message number. When you use `CATREAD` to read messages, the message facility fetches the message from a message catalog, inserts parameters (if specified), and then routes the message to a file or returns the message in a buffer to the calling program. The syntax for the `CATREAD` intrinsic is:

msglength := CATREAD (**catindex, setnum, msgnum, catstatus,**
buffer, buffersize, parm1, parm2, parm3, parm4,
parm5, msgdest);

The functional return, *msglength*, receives the length of the message in bytes. The *catindex* parameter refers to the catalog identifier you received from `CATOPEN`. The parameters, *setnum* and *msgnum* specify the set and message number of the message to be output. The *catstatus* parameter tells you if the `CATOPEN` call resulted in error, and, if so, what the error is. The optional parameters, *buffer* and *buffersize*, give the buffer to put the message in and the size of the buffer, respectively. The substitution parameters, *parm1* through *parm5*, contain character strings to be inserted into the message at run time. The file number to which the message may be sent is given in *msgdest*.

Parameter Substitution

Parameters may be inserted into the message read from the catalog. Parameter substitution is used when a message output contains information only known at run time. The parameters are passed to the message with the *param1*, *param2, param3, param4,* and *param5* parameters in the `CATREAD` intrinsic and are inserted in the message wherever an "`!`" is found. Parameters are inserted from left to right in positional parameter substation and in the numerical position indicated in numerical parameter substitution. In either case, if *param(n)* is included in the `CATREAD` call, *param(n-1)* must be present (that is, you cannot specify *param3* unless *param1* and *parm2* are specified. Refer to "Parameter Substitutions" in Chapter 2, for more information about parameter substitution. All substitutional parameters are passed as strings that must terminate with an ASCII null character.

Message Output

Messages may be output to a buffer or a file. If you output to a buffer, you specify the buffer and buffer size with the *buffer* and the *buffersize* parameters. To output to a file, you specify the file number (returned from `HPFOPEN`) and message length with the *msgdest* and the *buffersize* parameters. To output to `$STDLIST`, use a file number of 0 (zero).

To output message #400 from set #13 to `$STDLIST`, a call to the `CATREAD` intrinsic is done as follows:

```
var
  Msglength: SHORTINT;
  Catindex : INTEGER;  {Returned by CATOPEN}
  Setnum   : SHORTINT;
  Msgnum   : SHORTINT;
  Catstatus: Packed array [1..2] of SHORTINT;
  Parm_1,
  Parm_2   : STRING [ 5 ];
  Msgdest  : SHORTINT;
  Dumy     : INTEGER;

Setnum := 13;
Msgnum := 400;
Msgdest := 0; {Output to $STDLIST}
Parm_1 := 'MARY';
Parm_2 :='3';

{Append ASCII null}
STRWRITE (Parm_1, (STRLEN(Parm_1)+1), Dumy, CHR(0));
STRWRITE (Parm_2, (STRLEN(Parm_2)+1), Dumy, CHR(0));
Msglength := CATREAD (Catindex, Setnum, Msgnum,
   Catstatus,,, Parm_1, Parm_2,,,, Msgdest);
```

For detailed information about the `CATREAD` intrinsic, refer to the *MPE/iX Intrinsics Reference Manual* (32650-90028).

Closing a Catalog

`CATCLOSE` closes the message catalog specified by the *catindex* parameter. The syntax for the `CATCLOSE` intrinsic is:

CATCLOSE(**catindex, catstatus**)

The *catindex* parameter contains the value returned by the `CATOPEN` intrinsic that identifies the message catalog. The first element of *catstatus* returns the error number that tells if the call was successful.

Closing the example catalog using the `CATCLOSE` intrinsic is done as follows:

```
var
  Catstatus  : packed array [1..2] of SHORTINT;
  Catindex   : INTEGER:  {Returned by CATOPEN}

  CATCLOSE (Catindex, Catstatus);
```

For detailed information about the `CATCLOSE` intrinsic, refer to the *MPE/iX Intrinsics Reference Manual* (32650-90028).

Example of Accessing an Application Message Catalog

This example shows the access of the sample catalog called `FORMAT` (created in the last chapter); the source of this sample catalog (`SOURCE`) is listed below.

```
$SET 1 Prompts
1 ENTER FIRST NAME
2 ENTER LAST NAME
$
$
$set 2 Error Messages
1 NAME NOT ON DATA BASE
2 ILLEGAL INPUT
3 AN ERROR OCCURRED DURING THE LOADING %
OF THE DATA BASE.
98 THE NUMBER OF FILES &
DOES NOT MATCH THE &
SYSTEM'S CALCULATIONS.
$
$set 13 Run-Time Messages
400 INPUT FROM ! ON TERMINAL NUMBER !
401 INPUT FROM TERMINAL NUMBER !2 BY !1
```

The program uses message 1 in set 1 to prompt for a first name, substitutes the name in message 400 of set 13, and outputs the message. All output is written to the terminal.

```
Program MSGCAT (input,output);

var
  Catindex   : INTEGER;
  Catstatus  : packed array [1..2] of CHAR;

Function CATOPEN: INTEGER;  intrinsic;
Function CATREAD: SHORTINT; intrinsic;
Procedure CATCLOSE;         intrinsic;

Procedure OPEN_A_CATALOG;
{This procedure opens FORMAT}

var
  Designator : packed array [1..7] of SHORTINT;

begin
  Designator := 'FORMAT ';  {specify file name}
  Catindex := CATOPEN (Designator, Catstatus);
  {Call procedure to check Catstatus for errors}
end;

Procedure READ_A_CATALOG;
{This procedure reads a message}
{from FORMAT and prints it to  }
{$STDLIST                      }

var
  Setnum   : SHORTINT;
  Msgnum   : SHORTINT;
  Parm_1,
  Parm_2   : STRING [ 5 ];
  Dumy     : INTEGER;
  Msgdest  : SHORTINT;
  Msglength: SHORTINT;

begin
  Msgdest := 0; {Output to $STDLIST}
  Setnum := 1;
  Msgnum := 1;
  Msglength := CATREAD (Catindex, Setnum, Msgnum,
     Catstatus,,,,,,,, Msgdest);
  {Call procedure to check Catstatus}
  Readln (Parm_1);
  Parm_2 :='3';

  {Append ASCII null}
  STRWRITE (Parm_1, (STRLEN(Parm_1)+1), Dumy, CHR(0));
  STRWRITE (Parm_2, (STRLEN(Parm_2)+1), Dumy, CHR(0));
  Setnum := 13;
  Msgnum := 400;   {Output operation message}
  Msglength := CATREAD (Catindex, Setnum, Msgnum,
     Catstatus,,, Parm_1, Parm_2,,,, Msgdest);
  {Call procedure to check Catstatus}
end;
```

```
        Procedure CLOSE_A_CATALOG;
        {This procedure closes FORMAT}

        begin
          CATCLOSE (Catindex, Catstatus);
          {Call procedure to check Catstatus}
        end;

        begin  {main}
          OPEN_A_CATALOG;
          READ_A_CATALOG;
          CLOSE_A_CATALOG;
        end.
```

When this example is run, the output is as follows:

```
        ENTER FIRST NAME

        MARY

        INPUT FROM MARY ON TERMINAL NUMBER 3
```

Summary

After you have formatted your catalog, you access it with your program. This access is done using the "CATOPEN, CATREAD, and CATCLOSE intrinsics. The CATOPEN intrinsic is used to open the formatted catalog. CATREAD reads messages from the catalog and substitutes run-time parameters; they may be in positional or numerical form. The substitutional parameters must be strings with an ASCII null appended. CATCLOSE closes the catalog.

4

Modifying the Application Message Catalog

You modify a message catalog by adding, replacing, and deleting set, message, and comment records in the source file. This modification is done by merging two files, an old source file, and the maintenance file to create a new source file, as shown in Figure 4-1. The old source file is the catalog you originally created and input to the GENCAT utility to be formatted. The maintenance file contains the changes that will be made to the old source file.

You may use one of two methods to merge these files, either by line numbers or by set and message numbers, to create a new source. Unless consecutive blocks of a source file require modification, the set and message number merge method is the easiest.

You may keep changes made to a source during a maintenance merge in a collision file. Collision files give you the capability to rollback to the old version of the source file.

After you create a modified source file, you format it with GENCAT. The formatted catalog can then be accessed by your application program. For information about formatting the source file, refer to "Formatting a Source Catalog" in Chapter 2. For information about accessing your formatted catalog, refer to Chapter 3.

Overview

This chapter explains how to modify your application message catalog using maintenance and collision files. Topics include:

- Creating an Expanded Source File
- Creating a Maintenance File
- Creating a New Source File
- Rolling Back to an Old Source File

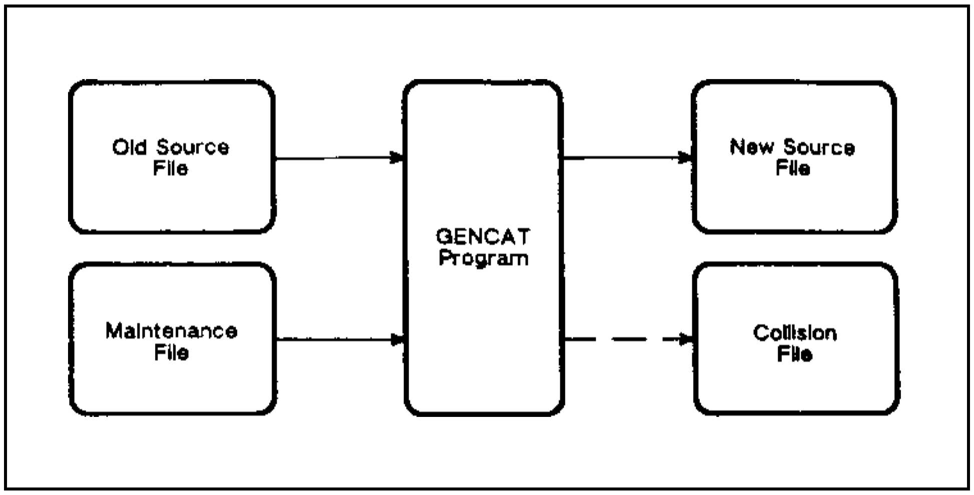


Figure 4-1. Merging Files to Create a New Source (Creating a Collision File is Optional)

Creating an Expanded Source File

To modify your application message catalog, you must have a source file. If you do not have access to your original source file, re-create a source file by expanding the formatted catalog with the GENCAT utility, as shown in Figure 4-2.

The expanded source file doesn't have the comments that were included in the original, but the rest of the content in the catalog is the same. The expanded file is created as an unnumbered file, even if the original source was numbered. An expanded source file and an original source file are used in the same way, they are both source files.

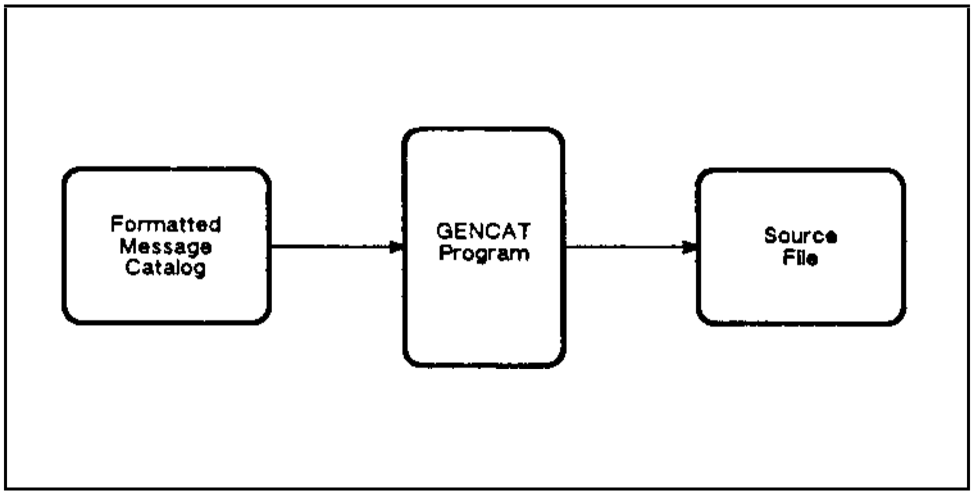


Figure 4-2. Expanding a GENCAT Formatted Catalog

Listed below is an example of the user dialog for expanding a formatted message catalog:

```
:RUN GENCAT.PUB.SYS
```

```
HP32414A.02.01 GENCAT/3000 (C) HEWLETT-PACKARD., 1983

ENTER INDEX OF DESIRED FUNCTION
    0.  EXIT.
    1.  HELP.
    2.  MODIFY SOURCE CATALOG.
    3.  FORMAT SOURCE INTO FORMATTED CATALOG.
    4.  EXPAND FORMATTED CATALOG INTO SOURCE.

>>4

ENTER NAME OF FORMATTED CATALOG TO EXPAND

>>FORMAT

ENTER NAME OF NEW SOURCE FILE


>>SOURCE

EXPANDING...

TOTAL NUMBER OF SETS EXPANDED = 3
TOTAL NUMBER OF MESSAGES EXPANDED = 8

EXPANSION SUCCESSFULLY COMPLETED
```

Modifying a Source File

There are two ways to modify a source catalog:

- Edit the source file
- Merge the source file with a maintenance file

Editing the Source File

The simplest way to modify the catalog is to text the file into EDIT/V and make changes directly. Although this is simple, it has many drawbacks:

- You have no record of the changes you've made; therefore, if the file doesn't work, you don't know what has been changed.
- Because there is no record of the changes, you have no rollback capability; you can't go back to the old version of the source.
- Most message files are quite large, so it is slow and inefficient to edit them.

For these reasons, editing the source file directly is not recommended for most catalogs and will not be discussed further.

Merging Source and Maintenance Files

You use GENCAT to merge source and maintenance files. You can use either of two different merging methods:

- Merging by Line Number
- Merging by Set and Message Numbers

Each method requires a different type of maintenance file. Creating each type of maintenance file is described below, and examples are given.

For examples of merging files, the example source catalog created earlier in this manual is used. The example catalog is listed below. The numbers at the left are the line numbers in the source file; to merge by line number, the source and maintenance files must be numbered. When you merge by set and message number, the files may be numbered or unnumbered.

```
 1    $SET 1 Prompts
 2    1 ENTER FIRST NAME
 3    2 ENTER LAST NAME
 4    $
 5    $
 6    $set 2 Error Messages
 7    1 NAME NOT ON DATA BASE
 8    2 ILLEGAL INPUT
 9    3 ERROR OCCURRED DURING THE LOADING %
10    OF THE DATA BASE.
11    98 THE NUMBER OF FILES &
12    DOES NOT MATCH THE &
13    SYSTEM'S CALCULATIONS.
14    $
15    $set 13 Run-Time Messages
16    400 INPUT FROM ! ON TERMINAL NUMBER !
17    401 INPUT FROM TERMINAL NUMBER !2 BY !1
```

Merging by Line Numbers

Merging a maintenance file against a source catalog file by line numbers allows adding, replacing, or deleting records. These records may be `$SET`, message, or comment records. This method is recommended if your changes are in blocks, you don't have a lot of modifications, or you want to add, delete, or modify comment lines; otherwise, you should use the merge by set and message number method described in the next section.

To merge files by line numbers, both source and maintenance file must be numbered. If either of them are unnumbered, GENCAT will fail.

Caution Merging by line numbers allows you to delete a set directive without deleting the messages within it or to add messages and set numbers in non-ascending order. These are errors that GENCAT will not allow. Be aware that you are adding and deleting information by record number.

Modifying a Record

If the maintenance file's line number is common to the source file's, the source's record is overwritten by the maintenance record.

Adding a Record

If the line number in the maintenance file does not exist in the source, the record represented by that line number from the maintenance file is added to the source at that line number. Make sure you don't have any blank lines in your maintenance file; GENCAT will merge them into your source file and when you try to format it, GENCAT will abort.

Deleting a Record

The directives `$EDIT` and `$EDIT VOID=XXXXXXXX` are used to delete records from the source file. `$EDIT` deletes the line in the source file that the `$EDIT` directive is on in the maintenance file. If `$EDIT VOID` is used, the records beginning with and including the record number of the `$EDIT VOID` record to record `XXXXXXXX` are deleted. The line number `XXXXXXXX` represents the line number `XXXXX.XXX` of the source file.

Example

Below is an example of merging by line numbers. The maintenance file that follows, is merged with the example source catalog created in Chapter 2 and repeated at the beginning of this chapter. The numbers on the left are the line numbers in the file.

```
 2    1 PLEASE ENTER YOUR FIRST NAME
 3    $EDIT
 5    $This is where $Set 2 used to be
 6    $EDIT VOID=13000
14.1  $SET 3 Misc Messages
14.2  4 Welcome to the System
14.3  6 Please wait ...
14.4  $
```

The new source file created by merging by line number is listed below:

```
 1    $SET 1 Prompts
 2    1 PLEASE ENTER YOUR FIRST NAME
 4    $
 5    $This is where $Set 2 used to be
14    $
14.1  $SET 3 Misc Messages
14.2  4 Welcome to the System
14.3  6 Please wait ...
14.4  $
15    $set 13 Run-Time Messages
16    400 INPUT FROM ! ON TERMINAL NUMBER !
17    401 INPUT FROM TERMINAL NUMBER !2 BY !1
```

Merging by Set and Message Number

This method is usually the easiest method of modifying your source. When the source is merged with a maintenance file by set and message numbers, you have rollback capability and an easy way to build a maintenance file.

When GENCAT reads a `$SET` record from the maintenance file, all records following the `$SET` record are considered to be message records or comment records within that set until GENCAT reads another `$SET` record or exhausts the maintenance file. Set numbers must be in ascending order, and all message numbers must be in ascending order within each set. GENCAT ensures that the new source file is created with the correct structure of set and message numbers.

The first record GENCAT expects to read from the maintenance file is a `$SET`, `$DELSET`, or a comment record. GENCAT will continue to read and evaluate the maintenance file records until there is an error or the maintenance file is exhausted. After GENCAT reads a maintenance file record, it is evaluated according to a set of rules, and a copy of the source is modified as necessary; a new source is then created. The following rules for evaluation apply to set numbers and message numbers.

Adding a Set

Any new set numbers are added to the source catalog file. All message numbers and messages following this set record are assumed to be new, and will be added to the source file.

Deleting a Set

To delete a set and the messages it contains, use the directive `$DELSET` *setnum*. The `$DELSET` directive is allowed only in a maintenance file. It instructs GENCAT to delete the entire set of messages denoted by *setnum*. A comment may follow *setnum*, providing it is preceded by at least one blank. The `$DELSET` directive is not written to the new file.

The directive may be either in upper case or lower case (`$DELSET` or `$delset`). Mixed cases are not allowed (for example, `$DELSet` or `$deLseT`).

Modifying a Set or Message

Set or message numbers, if already present, signify changes to the sets and messages currently in the source catalog.

Adding a Message

New message numbers within a set are added to the new source. When any new message numbers are added, you must specify the set to which they belong.

All message numbers are evaluated according to the rules for message numbers (ascending order, 1 through 32,766).

Deleting a Message

Message numbers that are already present are deleted if no text follows the message number. The line of text is replaced by a comment line.

Comment Records

Comment records cannot be added, deleted, or modified when you merge by set and message number. You must merge by line number to perform operations on comment records. All comment lines included in a maintenance file for message and set number merging are ignored by GENCAT.

Example

Below is an example of merging by set and message numbers. The maintenance file that follows, is merged with the example source catalog created in Chapter 2 and repeated at the beginning of this chapter. No line numbers are shown because these files can be numbered or unnumbered.

```
$set 1
1 PLEASE ENTER YOUR FIRST NAME
2
$DELSET 2
$SET 3 Misc Messages
4 Welcome to the System
6 Please wait ...
```

The new source file created by merging by line number is listed below:

```
$set 1
1 PLEASE ENTER YOUR FIRST NAME
$
$
$
$SET 3 Misc Messages
4 Welcome to the System
6 Please wait ...
$set 13
400 INPUT FROM ! ON TERMINAL NUMBER !
401 INPUT FROM TERMINAL NUMBER !2 BY !1
```

Creating a New Source File

Figure 4-3 shows how the new source file is created by merging the old source with the maintenance file. A collision file may be created at this time. Collision files are described in the next section.

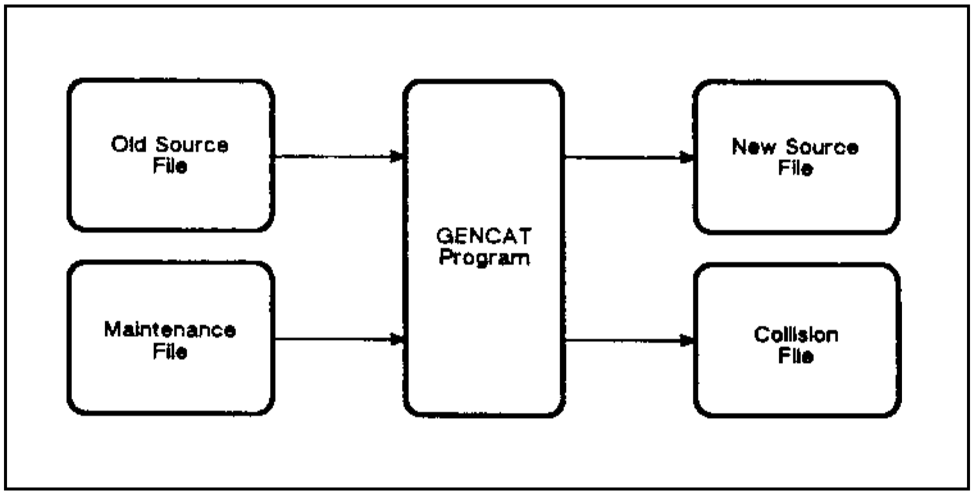


Figure 4-3. Creating a New Source

The dialog listed below is an example of merging source and maintenance files. Note that the method of merging you specify depends on the type of maintenance file you created. To use GENCAT to modify your source catalog, enter:

```
:RUN GENCAT.PUB.SYS

HP32414A.02.01 GENCAT/3000 (C) HEWLETT-PACKARD., 1983

 ENTER INDEX OF DESIRED FUNCTION

 0.  EXT.
 1.  HELP.
 2.  MODIFY SOURCE CATALOG.
 3.  FORMAT SOURCE INTO FORMATTED CATALOG.
 4.  EXPAND FORMATTED CATALOG INTO SOURCE.

>>2

ENTER NAME OF CATALOG SOURCE FILE TO BE MODIFIED

>>SOURCE

ENTER NAME OF MAINTENANCE FILE

>>MAINT

ENTER INDEX OF MERGE TYPE

0.  DO NOT MERGE.
1.  HELP.
2.  BY LINE NUMBER.
3.  BY SET/MESSAGE NUMBER.
```

```
>>2     or >>3
```

Entering an "0" or Return aborts the maintenance function and returns to the main menu.

You have the option of saving all the modifications resulting from the merge in a collision file.

```
SAVE COLLISIONS?  ENTER "YES" OR "NO"

>>YES

ENTER NAME OF COLLISION FILE

>>COLISION
```

If the name of an existing file is entered, the prompt is repeated. A Return continues the merging without saving the collisions.

GENCAT merges the source and maintenance files into a temporary file, and will prompt for the name of a permanent file:

```
ENTER NAME OF NEW SOURCE CATALOG FILE

>>NEWSOURC
```

This prompt is repeated until a unique file name or a Return is entered. The temporary file is copied to the new permanent file. If a Return is entered, the merging is aborted.

After a new source is created, it must be formatted before it can be accessed. Formatting a source catalog is discussed in Chapter 3.

Rolling Back to the Old Source File

After creating a new source file by merging an old source file with a maintenance file, you may want to return to using the old source after it is no longer available. To rollback to the old source, use a collision file, as shown in Figure 4-4.

Collision files are created at your option when GENCAT merges source and maintenance files. A collision file contains the changes that were made to an old source. A collision file's contents depends on the source and maintenance files and the method used of merging. Unlike a maintenance file which contains information to change the old source into the new source, a collision file contains information to re-create the old source from the new source. To rollback to the previous version of the source, the collision file is merged against the new source with the same merging method that was used to create it.

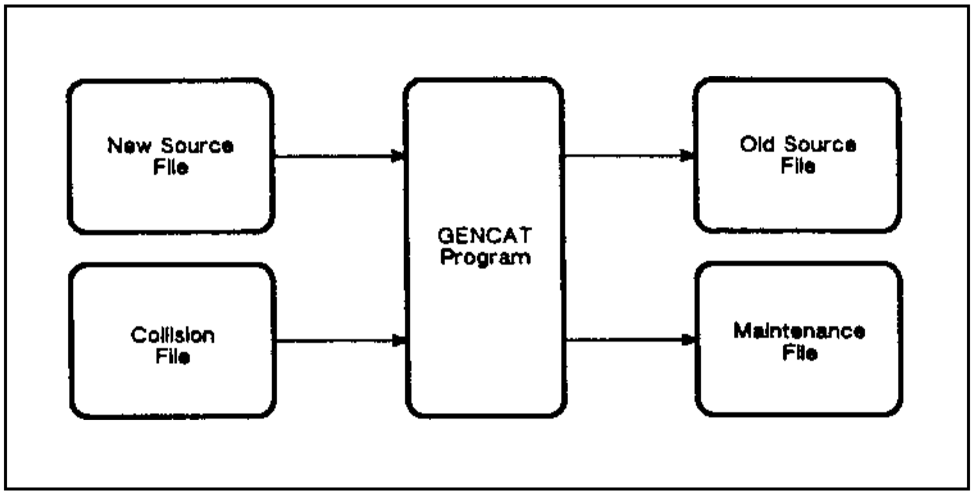


Figure 4-4. Rolling Back to the Old Source Catalog

Summary

To modify a message catalog, you modify your source file then reformat the new source before accessing the message catalog.

To create a new source, merge the maintenance files against the old source by line number, or set and message number. A collision file may be created during the merging operation. When the collision file is used as a maintenance file on the new source, the old source is re-created.

5

Accessing System Error Messages

The System Message Facility consists of two error message catalogs. The message catalogs, `CATALOG.PUB.SYS` and `SYSCAT.PUB.SYS`, contain Compatibility Mode (CM) and Native Mode (NM) error messages. You can access these messages, but cannot modify them. To output error messages without creating your own message catalog, use system error messages.

When to Use System Error Messages

Only if you prefer to use system error messages should you consider accessing them. Unless you have specific reasons for using system error messages, such as creating a transparent interface, creating your own message catalogs and accessing them is the easiest way to use message catalogs. To learn about creating your own message catalogs, refer to Chapter 2.

Overview

This chapter tells you how to use system error messages as output from your applications. It explains:

- Message Catalog Format
- Accessing `CATALOG.PUB.SYS`, the CM error message catalog, using:
  - `HPFOPEN` to open the catalog
  - `GENMESSAGE` to read and output message
  - `FCLOSE` to close the catalog
- Accessing `SYSCAT.PUB.SYS`, the NM error message catalog using:
  - `CATOPEN` to open the catalog
  - `CATREAD` to read and output messages
  - `CATCLOSE` to close the catalog

`CATALOG.PUB.SYS` contains the CM error messages. It was formatted with the `MAKECAT.PUB.SYS` program. The catalog is opened and closed with the `HPFOPEN` and `FCLOSE` intrinsics; messages are read with the `GENMESSAGE` intrinsic. `GENMESSAGE` allows you to access the message catalog and substitute parameters in the message.

`SYSCAT.PUB.SYS` contains the NM error messages. This message catalog was formatted with the `GENCAT.PUB.SYS` program. You use the catalog intrinsics to open, read from, and close this catalog. They are `CATOPEN, CATREAD,` and `CATCLOSE`. `CATREAD` allows parameter substitution where applicable.

Both CATALOG.PUB.SYS and SYSCAT.PUB.SYS may be accessed from Native Mode. Figure 5-1 shows the catalogs and the intrinsics that access them.

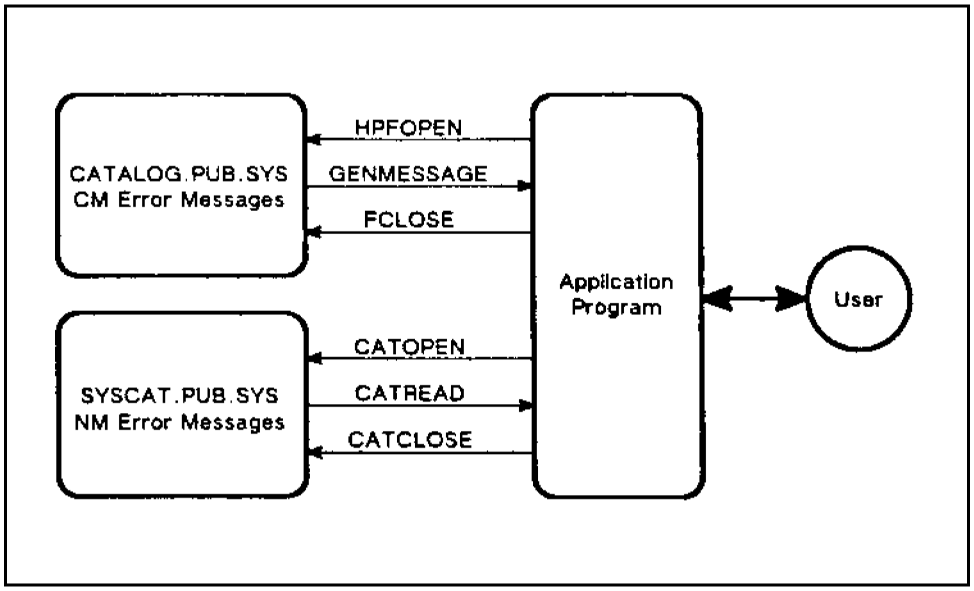


Figure 5-1. Accessing System Error Messages

Format of a Message Catalog

Although the two system error message catalogs contain different sets of messages, their format is the same. The messages are numbered and grouped into numbered sets. The sets are logical divisions of messages. Before you can access the messages, you must know the messages that are available and their identifying message and set message numbers.

To examine the CM error messages you can use, go into your editor and text in CATALOG.PUB.SYS; to examine the NM error messages, expand the formatted file SYSCAT.PUB.SYS, as shown in Chapter 4, and text the expanded catalog into your editor. Both of these files are very large, but you can view them from within your editor. As you look at the message catalog, you will see messages, message numbers, set directives, set numbers, comments, and special characters, such as %, &, ! and ~.

Message File Format Example

The messages you see will look similar to the following example from CATALOG.PUB.SYS:

```
$SET 1     SYSTEM MESSAGES
1 LDEV#!IN USE BY FILE SYSTEM
2 LDEV#!IN USE BY DIAGNOSTICS
3 LDEV#!IN USE, DOWN PENDING
5 IS "!" ON LDEV#! (Y/N)?
    .
    .
    .
$MESSAGE 35 IS TWO LINES LONG, A PARAMETER STARTS THE
```

```
$FIRST LINE, AND THE SECOND LINE IS "HP32002"
35!%
HP32002B.00.!
    .
    .
    .
276 LDEV # FOR "!" ON ! (NUM)!
$
$SET 2 CIERROR MESSAGES
82 STREAM FACILITY NOT ENABLED: SEE OPERATOR.(CIERR 82)
200 MORE THAN 30 PARAMETERS TO BUILD COMMAND.(CIERR 200)
    .
    .
    .
```

Directives and Special Characters

Directives and special characters are not output to users, but they are important factors in the unformatted message catalog.

A directive begins in column one and denotes the beginning of a set, a comment, or a message. They are:

- `$SET` - indicates the beginning of a logical set of messages
- `$` - proceeds a comment
- *message number* - indicates the beginning of a message

Special characters are used to control the format and content of messages when the messages are output.

- `%` - lets a message continue on the next line in the catalog and when printed out. When a message is output to a buffer, a space is inserted where the `%` was.
- `&` - lets a message continue on the next line in the catalog, but, when accessed, the message is printed on one line.
- `!` - allows a parameter to be inserted at run time. Up to five parameter substitutions are allowed in a message, and substitution is done from left to right.
- `~` - when preceding a special character, allows the special character to be printed.

Accessing the CM Error Message Catalog

The CM error message catalog, CATALOG.PUB.SYS, is read with the GENMESSAGE intrinsic; the catalog file is opened with HPFOPEN and closed with FCLOSE. Accessing any catalog that was formatted with MAKECAT.PUB.SYS is similar to accessing CATALOG.PUB.SYS. Figure 5-2 shows how intrinsics are used to access the CM error message catalog.

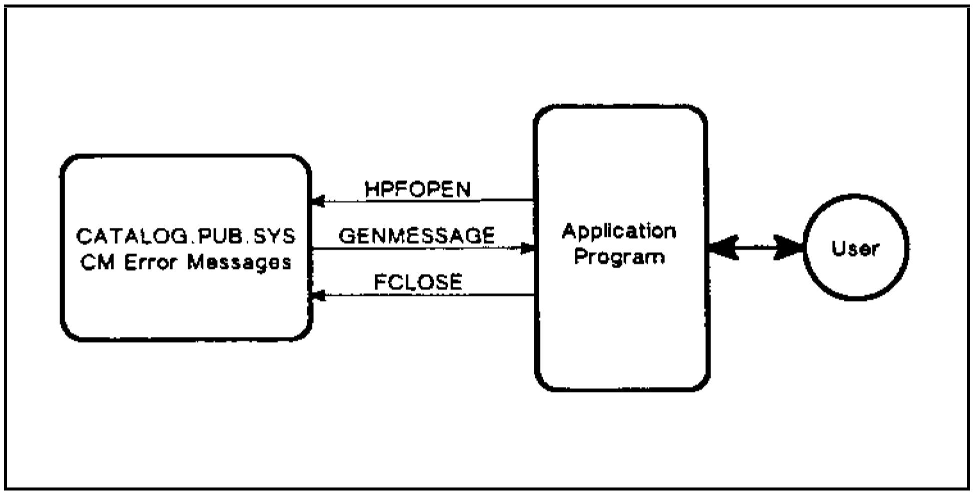


Figure 5-2. Accessing CATALOG.PUB.SYS

Opening the CM Error Message Catalog

CATALOG.PUB.SYS must be opened as a permanent ASCII file with buffering inhibited and multirecord mode (PERMANENT (OLD), ASCII, NOBUF, MULTI). You use HPFOPEN to open the catalog. HPFOPEN returns the file number for CATALOG.PUB.SYS. This file number is a required parameter for the GENMESSAGE intrinsic.

The HPFOPEN intrinsic is the most efficient way to open a file for access. When using the HPFOPEN intrinsic, the parameters are:

domain option (item#3) = 1 (PERMANENT)

multirecord option (item#15) = 1 (MULTI)

inhibit buffering option (item#46) = 1 (NOBUF)

ASCII/*Binary option* (item#53) = 1 (ASCII)

To open CATALOG with HPFOPEN:

```
const   {These are the item numbers for HPFOPEN}
  Designator   = 2;
  Domain       = 3:
  MultiRec     = 15;
  Buffering    = 46;
  ASCII/Binary = 53;

var  {These are the parameters and options}
     { for HPFOPEN                         }
  Filenum      : INTEGER;   {Returned by HPFOPEN}
  Status       : INTEGER;   {Returned by HPFOPEN}
```

```
FileName     : packed array [1..20] of CHAR;
Perm         : INTEGER;
On           : INTEGER;
Inhibited    : INTEGER;
ASCII        : INTEGER;

FileName := '%CATALOG.PUB.SYS%';
Perm := 1;
On := 1;
Inhibited := 1;
ASCII := 1;
HPFOPEN (Filenum, Status, Designator, FileName,
 Domain, Perm, MultiRec, On, Buffering, Inhibited,
 ASCII/Binary, ASCII);
```

`Filenum` returns the file number for `CATALOG.PUB.SYS`; `Status` returns a value that indicates if the intrinsic call was successful. If it was not successful, `Status` gives you information about the error.

For detailed information about the `HPFOPEN` intrinsic, refer to the *MPE XL Intrinsics Reference Manual* (32650-90028).

Reading Messages With GENMESSAGE

Use the `GENMESSAGE` intrinsic to read messages from `CATALOG.PUB.SYS`. Call `GENMESSAGE` with the file number for `CATALOG.PUB.SYS`, a set number, a message number, and any values to be substituted in the message.

When you use `GENMESSAGE` to read messages, the message facility fetches the message from a message catalog, inserts parameters (if specified), and then routes the message to a file or returns the message in a buffer to the calling program. The syntax for the `GENMESSAGE` intrinsic is:

msglength := `GENMESSAGE` (**filenum, setnum, msgnum,** *buffer, buffersize, parmask, param1, param2, param3, param4, param5, msgdestination, errornum*);

The parameters *param1* ... *param5* are used to substitute values in the message at run time.

Parameter Substitution

Parameters may be inserted into the message read from the catalog. Parameter substitution is used when a message output contains information only known at run time, such as a ldev number or a session name. Parameters are passed to the message with the *param1*, *param2, param3, param4,* and *param5* parameters in the `GENMESSAGE` intrinsic and are inserted in the message wherever a "`!`" is found. Parameters are inserted in the following order: *param1* substitutes for the leftmost "`!`" in the message, *param2* for the next `!` to the right, and so forth. If *param(n)* is present, *param(n-1)* must be present (that is, you cannot specify *param3* unless *param1* and *param2* are specified).

To specify the format of each of your parameters, use the *parmask* parameter of the `GENMESSAGE` intrinsic.

The *parmask* parameter indicates the format of each of the five substitution parameters. Three bits describe the data type for each of these parameters. With bit zero being the leftmost bit, the value of *parmask* is represented as follows:

Bits (0:3) *param1*
Bits (3:3) *param2*
Bits (6:3) *param3*
Bits (9:3) *param4*
Bits (12:3) *param5*

These bit values are as follows:

000	Parameter is a string that is terminated by an ASCII null (0).
001	Parameter is a 16-bit signed integer.
010	Parameter is a 32-bit signed integer by reference.
011	Parameter is ignored.

The positions of the bit values given above, indicate which substitution parameter's data type is being specified. For example, *parmask* = `OCTAL ('13333')` denotes that the first parameter is passed as a 16-bit signed integer, and all other parameters are ignored.

Note If a substitutional parameter (*param1* through *param5*) is not specified, the value in the *parmask* for its format is ignored.

Message Output

Messages may be output to a buffer or a file. If you output to a buffer, you specify the buffer and a buffer size with the *buffer* and the *buffersize* parameters. To output to a file, you specify the file number and message the *buffersize* parameters. To output to `$STDLIST`, use a file number of 0 (zero).

To output message #210 from set #1 to `$STDLIST`, use `GENMESSAGE` to access `CATALOG.PUB.SYS` in the following manner:

```
var
   Msglength  : SHORTINT; {Returns length of message}
   Filenum    : SHORTINT; {Value assigned by HPFOPEN}
   Setnum     : SHORTINT;
   Msgnum     : SHORTINT;
   Parmask    : SHORTINT;
   Param1     : INTEGER;
   Msgdestination  : SHORTINT;
   Errornum   : SHORTINT;

   Setnum := 1;
   Msgnum := 201;
   Parmask := OCTAL ('13333');
   Param1 := 95;
   Msglength := GENMESSAGE (Filenum, Setnum, Msgnum,,,
      Parmask, Param1,,,,, Msgdestingaion, Errornum);
```

For detailed information about the GENMESSAGE intrinsic, refer to the *MPE/iX Intrinsics Reference Manual* (32650-90028).

Closing the CM Error Message Catalog

To close the CM error message catalog, use the FCLOSE intrinsic. Close the catalog with the same domain as when opened (PERM) with unrestricted access.

```
        var
           Filenum      : INTEGER;  {Returned by HPFOPEN}
           Disposition  : SHORTINT;
           Securitycode : SHORTINT;

           Disposition := 0;   {No changes in domain }
                               {or disc space}
           Securitycode := 0;  {Unrestricted access}
           FCLOSE (Filenum, Disposition, Securitycode)
```

For detailed information about the FCLOSE intrinsic, refer to the *MPE/iX Intrinsics Reference Manual* (32650-90028).

Example of Accessing the CM Error Message Catalog

The following listing is a Pascal program that inserts the value 95 into message number 201 in message set 1 in the message catalog CATALOG.PUB.SYS. The message is output to $STDLIST. The accessed portion of the message catalog is:

```
        $SET 1
           .
           .
           .
        201 SYSTEM LOG FILE NUMBER ! IS ON
```

```
   Program CM_MSGCAT (input,output);
   {This program reads a message from CATALOG.PUB.SYS   }
   {and writes it to $STDLIST                           }

   var
     Filenum      : INTEGER;

   Procedure HPFOPEN       ; intrinsic;
   Function GENMESSAGE: SHORTINT; intrinsic;
   Procedure FCLOSE             ; intrinsic;

   Procedure OPEN_CATALOG;
   {This procedure opens CATALOG.PUB.SYS }

   const   {These are the item numbers for HPFOPEN}
     Designator   = 2;
     Domain       = 3;
     MultiRec     = 15;
     Buffering    = 46;
     ASCII_Binary = 53;
```

```
var  {These are the parameters for HPFOPEN}
  Status       : INTEGER;   {Returned by HPFOPEN}
  FileName     : packed array [1..20] of char;
  Perm         : INTEGER;
  On           : INTEGER;
  Inhibited    : INTEGER;
  ASCII_file   : INTEGER;

begin
  FileName := '%CATALOG.PUB.SYS%';
  Perm := 1;
  On := 1;
  Inhibited := 1;
  ASCII_file := 1;
  HPFOPEN (Filenum, Status, Designator, FileName,
     Domain, Perm, MultiRec, On, Buffering, Inhibited,
     ASCII_Binary, ASCII_file);

  {Call procedure to check Status for errors}
end;

Procedure READ_CATALOG;
{This procedure reads and outputs a message from }
{CATALOG.PUB.SYS and outputs it to $STDLIST     }

var
   msglength  : SHORTINT;  {Returns length of message}
   Setnum     : SHORTINT;
   Msgnum     : SHORTINT;
   Parmask    : SHORTINT;
   Param1     : integer;
   Msgdestination  : SHORTINT;
   Errornum   : SHORTINT;

begin
   Setnum  := 1;
   Msgnum  := 201;
   Msgdestination := 0;
   Parmask := Octal('13333');
   Param1  := 95;
   msglength := GENMESSAGE (Filenum, Setnum, Msgnum,,,
      Parmask, Param1,,,,, Msgdestination, Errornum);

   {Call procedure to check Errornum for errors}
end;

Procedure CLOSE_CATALOG;
{This procedure closes CATALOG.PUB.SYS}

var
   Disposition  : SHORTINT;
```

```
      Securitycode : SHORTINT;

   begin
      Disposition  := 0; {No changes in domain }
                         {or disc space}
      Securitycode := 0; {Unrestricted access}
      FCLOSE (Filenum, Disposition, Securitycode);

      {Call procedure to check Condition Code for errors}
   end;

   begin {main}
      OPEN_CATALOG;
      READ_CATALOG;
      CLOSE_CATALOG;
   end.
```

When this program is executed, the output is:

```
SYSTEM LOG FILE NUMBER 95 IS ON
```

Accessing the NM Error Message Catalog

To open, read, and close the NM error message catalog, SYSCAT.PUB.SYS, use the catalog intrinsics: CATOPEN, CATREAD, and CATCLOSE. Accessing any catalog that was formatted with GENCAT.PUB.SYS is similar to accessing SYSCAT.PUB.SYS. Figure 5-3 shows the intrinsics used to access the NM system error catalog.

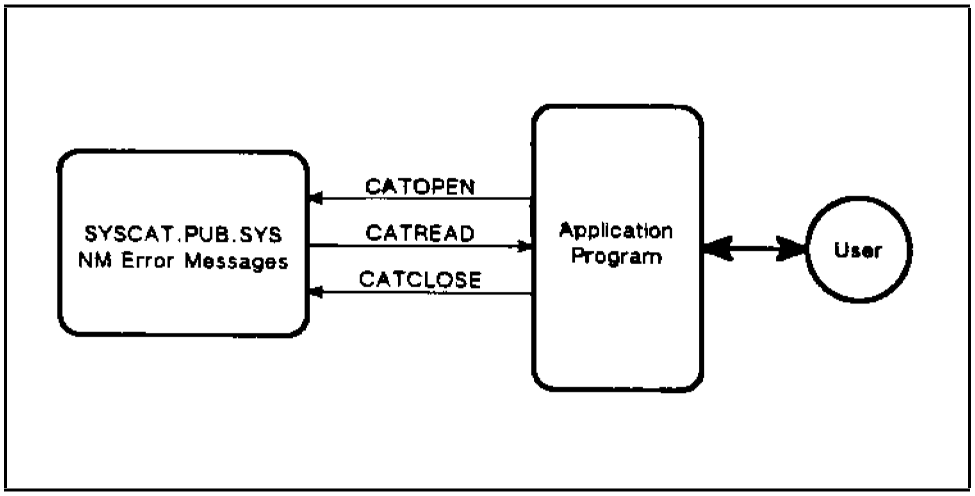


Figure 5-3. Accessing SYSCAT.PUB.SYS

Opening the NM Error Message Catalog

The `CATOPEN` intrinsic opens the NM error message catalog. The syntax for `CATOPEN` is:

catindex := CATOPEN (**formaldesignator, catstatus**);

The *catindex* is an index used to identify the message catalog being accessed. This number is not the same as a file number. The *formaldesignator* parameter contains a string that identifies the catalog file to open; *catstatus* returns the error number. If the first element of *catstatus* is zero, the intrinsic call was successful, if this element is not zero, its value indicates the error that occurred. Opening `SYSCAT.PUB.SYS` is done as follows:

```
var
  Designator : packed array [1..20] of CHAR;
  Catstatus  : packed array [1..2] of SHORTINT;
  Catindex   : INTEGER;

  Designator := 'SYSCAT.PUB.SYS ';
  Catindex := CATOPEN (Designator, Catstatus);
```

For detailed information about the `CATOPEN` intrinsic, refer to the *MPE/iX Intrinsics Reference Manual* (32650-90028).

Reading Messages With CATREAD

The `CATREAD` intrinsic reads the message specified by set and message numbers from the catalog specified by *catindex*. When you use `CATREAD` to read messages, the message facility fetches the message from a message catalog, inserts parameters (if specified), and then routes the message to a file or returns the message in a buffer to the calling program. The syntax for `CATREAD` is:

msglength :=CATREAD (**catindex, setnum, msgnum, catstatus,**
buffer, buffersize, parm1, parm2, parm3, parm4,
parm5, msgdest);

The parameters *parm1 ... parm5* are used to substitute values in the message at run time.

Parameter Substitution

Parameters may be inserted into the message read from the catalog. Parameter substitution is used when a message output contains information only known at run time, such as an ldev number or a session name. Parameters are passed to the message with the *param1*, *param2*, *param3*, *param4*, and *param5* parameters in the `CATREAD` intrinsic and are inserted in the message wherever an "`!`" is found. Parameters are inserted in the following order: *param1* substitutes for the leftmost "`!`" in the message, *param2* for the next `!` to the right, and so forth. If *param(n)* is present, *param(n-1)* must be present (that is, you cannot specify *param3* unless *param1* and *param2* are specified). Parameters are passed as strings that must have the ASCII null character appended.

Message Output

Messages may be output to a buffer or a file. If you output to a buffer, you specify the buffer and a buffer size with the *buffer* and the *buffersize* parameters. To output to a file, you specify the file number (returned from `HPFOPEN`) and message length with the *msgdest* and the *buffersize* parameters. To output to `$STDLIST`, use a file number of 0 (zero).

To output message #8 from set #221 to $STDLIST, a call to the CATREAD intrinsic is done as follows:

```
var
  Catindex  : INTEGER;
  Catstatus : packed array [1..2] of SHORTINT;
  Setnum    : SHORTINT;
  Msgnum    : SHORTINT;
  Parm1     : STRING [ 3 ];
  Dumy      : INTEGER:
  Msgdest   : SHORTINT;
  Msglength : SHORTINT;

  Setnum := 221;
  Msgnum := 8;
  Parm1  := '42';
     {Append ASCII null}
  STRWRITE (Parm1, STRLEN(Parm1)+1, Dumy, CHR(0);
  Msgdest  := 0;  {Output to $STDLIST}
  Msglength := CATREAD (Catindex, Setnum, Msgnum,
     Catstatus,,, Parm1,,,,, Msgdest);
```

For detailed information about the CATREAD intrinsic, refer to the *MPE/iX Intrinsics Reference Manual* (32650-90028).

Closing the NM Error Message Catalog

CATCLOSE closes the message catalog specified by the *catindex* parameter. The syntax for CATCLOSE is:

CATCLOSE (**catindex, catstatus**)

The *catindex* parameter contains the value returned by the CATOPEN intrinsic that identifies the message catalog. The first element of *catstatus* returns the error number that tells if the call was successful.

An example of the CATCLOSE intrinsic follows:

```
var
  Catstatus  : packed array [1..2] of SHORTINT;
  Catindex   : INTEGER;  {Returned by CATOPEN}

  CATCLOSE (Catindex, Catstatus);
```

For detailed information about the CATCLOSE intrinsic, refer to the *MPE/iX Intrinsics Reference Manual* (32650-90028).

Example of Accessing the NM Error Message Catalog

The following listing is a Pascal program that inserts the value 42 into message #8 in set #201 in the message catalog `SYSCAT.PUB.SYS`. The message is output to `$STDLIST`. The accessed portion of the message catalog is:

```
       $SET 201
          .
          .
          .
       008 The value passed for parameter #! is invalid.
```

```
Program NM_MSGCAT (input, output);

var
  Catindex   : INTEGER;
  Catstatus  : packed array [1..2] of SHORTINT;

Function CATOPEN:  INTEGER; intrinsic;
Function CATREAD: SHORTINT; intrinsic;
Procedure CATCLOSE        ; intrinsic;

Procedure OPEN_SYSCAT;
{This procedure opens SYSCAT.PUB.SYS}

var
  Designator : packed array [1..20] of CHAR;

begin
  Designator := 'SYSCAT.PUB.SYS ';
  Catindex := CATOPEN (Designator, Catstatus);

  {Call procedure to check Catstatus for errors}
end;

Procedure READ_SYSCAT;
{This procedure reads a message from SYSCAT.PUB.SYS}
{and prints it to $STDLIST                         }

var
  Setnum    : SHORTINT;
  Msgnum    : SHORTINT;
  Parm1     : STRING [ 3 ];
  Dumy      : INTEGER;
  Msgdest   : SHORTINT;
  Msglength : SHORTINT;

begin
  Setnum := 221;
  Msgnum := 8;
  Parm1 :='42';
  STRWRITE (Parm1, STRLEN(Parm1)+1, Dumy, CHR(0));
```

```
    {Append ASCII null}
    Msgdest  := 0;  {Output to $STDLIST}
    Msglength := CATREAD (Catindex, Setnum, Msgnum,
       Catstatus,,, Parm1,,,,, Msgdest);

    {Call procedure to check Catstatus for errors}
  end;

  Procedure CLOSE_SYSCAT;
  {This procedure closes SYSCAT.PUB.SYS}

  begin
    CATCLOSE (Catindex, Catstatus);
    {Call procedure to check Catstatus for errors}
  end;

  begin {main}
     OPEN_SYSCAT;
     READ_SYSCAT;
     CLOSE_SYSCAT;

  end.
```

When this program is executed, the output is:

```
  The value passed for parameter #42 is invalid.
```

Summary

You are allowed to output the messages from the MPE operating system to use in your own manner, from your own applications. You should only use the System Error Messages if you have a particular reason for doing so. Otherwise, you should create your own message catalog. Using the `HPFOPEN`, `GENMESSAGE`, and `FCLOSE` intrinsics, you access `CATALOG.PUB.SYS`, the Compatibility Mode error messages. The catalog intrinsics, `CATOPEN`, `CATREAD`, and `CATCLOSE`, access `SYSCAT.PUB.SYS`, the Native Mode error message catalog.

Messages are identified by message and set numbers. Up to five paramters can be substituted in messages at run time. Messages may be output to a file or a buffer.

6

Creating Your Own HELP Facility

A HELP facility is a help catalog and the user interface to that catalog. You create your own HELP facility by developing a help catalog and allowing users to access it via the MPE/iX HELP user interface. You are probably already familiar with the MPE/iX HELP user interface; you use it when you access the MPE/iX HELP facility.

HELP facilities provide extra information to the user by outputting explanatory text in response to keyword input. You create this information easily and access it in a simple and consistent manner.

To create your own HELP facility, you first create a source file that contains keywords to identify sections of the catalog. These keywords allow the user access to the HELP facility. The source file is then formatted and the user interface is redirected to the formatted file. Finally, users access the HELP facility.

Overview

This chapter tells you how to create your own HELP facility. Topics include:

- Creating the Source File
- Formatting the Source File
- Accessing the HELP Facility

Creating the Source File

The source file is in EDIT/V format and must have a maximum record length of 72 characters. It contains special catalog commands that begin with a backslash (\) in column 1. There are two different kinds of commands used in a source file:

- Keyword Commands
- File Commands

Keyword Commands

Keywords identify the set of text that is output when a keyword is input to the HELP facility. For example, if the keyword `HELLO` is input to the MPE/iX HELP Facility, a description of the `:HELLO` command, and an explanation of its syntax, operation, and parameters is output.

There are three keyword commands:

- `\ENTRY=`*entryname comment*, where *entryname* is a major divisional entry point in your HELP facility and *comment* is optional text about the entry. The *entryname* must be in upper case.
- `\ITEM=`*itemname*, where *itemname* is an entry point subordinate to *entryname*.
- `\SUBITEM=`*subitemname*, where *subitemname* is a subordinate entry point to *itemname*.

Keywords are *entrynames, itemnames,* and *subitemnames*. Keywords must not contain spaces or nonalphabetic characters. In your source file, the text on the lines between the keyword commands is output to the user and is accessed in the hierarchical method shown in Figure 6-1.

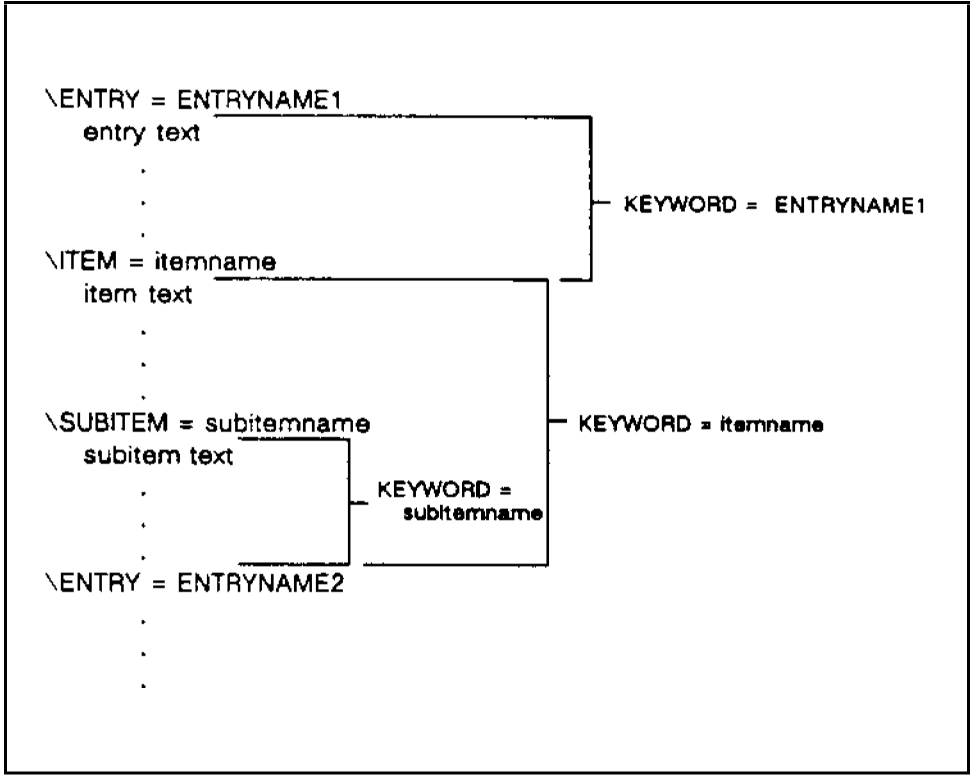


Figure 6-1. Keyword Subdivisions

File Commands

The other commands in the source file specify information that is not output to users. The file commands are:

- `\ALL` indicates the end of the source file. It is required.
- `\STOPHELP` denotes the start of a section in the file that will not be shown when the facility is accessed.
- `\STARTHELP` denotes the point at which the HELP facility resumes displaying help information.
- `\SUBSET`, put at the beginning of the file, allows all text between `\STOPHELP` and `\STARTHELP` to be excluded from the formatted catalog.

Example Source File

The following is an example of a HELP source catalog:

```
\entry = ENTRYKEY
This is a entry point
\item = itemkey
This is the listing for an item
\subitem = subitemkey
This is a subitem
\subitem = more
This is another subitem.
\stophelp
This text will not be displayed in the help catalog.
\starthelp
\entry = ENTRY2
This is the second entry point
\all
```

Formatting the Source File

To create a HELP catalog that is accessible through the MPE/iX help user interface, format the source file with the MAKECAT utility.

The `MAKECAT` utility takes its input from the file `INPUT` and outputs to the file `HELPCAT`. You can redirect these files. When you run the `MAKECAT` utility, you use the `HELP` entry point.

For example, if your source file is called `MYSOURCE` and you want your HELP catalog to be called `MYHELP`, you use MAKECAT to build a HELP catalog by entering:

```
:FILE INPUT=MYSOURCE
:RUN MAKECAT.PUB.SYS, HELP
END OF PROGRAM
:RENAME HELPCAT,MYHELP
```

If a file named `HELPCAT` exists prior to running MAKECAT, MAKECAT will abort with the messages:

```
        ** FILE ERROR ON CATALOG (100)

        PROGRAM TERMINATED IN ERROR STATE.  (CIERR 976)
```

Accessing the HELP Facility

To use your HELP facility you must disable the system HELP facility and assign your HELP file to `CICAT.PUB.SYS`. Create a UDC or User Command to perform these functions easily. To access your HELP facility with the command **`HELPME`**, use the following UDC:

```
        HELPME PARM1=" " PARM2= " "
        FILE CICAT.PUB.SYS = MYHELP
        HELP !PARM1 !PARM2
        RESET CICAT.PUB.SYS
```

You can access the HELP in one of two modes: subsystem or immediate. Immediate mode gives you the information on the one or two keywords about which you inquire (for example, `HELPME` **`keyword1 keyword2`**). Keywords are entry, item, or subitem names. You enter the subsystem mode when you call the facility without keywords (for example, `HELPME`). When in subsystem mode, you enter `ALL, EXIT`, or a keyword at the greater than (`>`) prompt.

Summary

The HELP Facility is a different kind of message facility; messages are not identified by set and message numbers. To create your own HELP facility, you create your source file, format the file with MAKECAT and then redirect the MPE/iX user interface to your HELP catalog. This interface defaults to the system HELP catalog, `CICAT.PUB.SYS`.

A

GENCAT Error Messages

The following messages are returned from GENCAT:

1	**MESSAGE**	`FREAD ERROR ON SOURCE FILE.`
	CAUSE	A failure by `FREAD` when reading a source message catalog.
	ACTION	Re-create the source message catalog.
2	**MESSAGE**	`INPUT FILE MUST HAVE AT LEAST ONE RECORD.`
	CAUSE	The file has an EOF of zero (0).
	ACTION	Place at least one record in the file.
3	**MESSAGE**	`INPUT FILE MUST CONTAIN FIXED LENGTH RECORDS ONLY.`
	CAUSE	File does not have a fixed record length.
	ACTION	Create the file with a fixed record length.
4	**MESSAGE**	`INPUT FILE MUST BE ASCII FILE ONLY.`
	CAUSE	Source and maintenance files must have records that are in ASCII format.
	ACTION	Create the source and maintenance files with ASCII format.
5	**MESSAGE**	`INPUT FILE RECORD SIZE MUST BE BETWEEN 40 AND 256 BYTES.`
	CAUSE	The record size of a source or maintenance file is greater than 256 bytes (128 words) or less than 40 bytes (20 words).
	ACTION	Create a source and maintenance file with a record size greater or equal to 40 bytes or less than or equal to 256 bytes. (Note that this record length includes any line numbers in the file.)

6	**MESSAGE**	`SET NUMBERS MUST BE BETWEEN 1 AND 255.`
	CAUSE	A set number in a maintenance or source file is not greater than or equal to 1, or not less than or equal to 255. The set number may be negative or it may not be numeric.
	ACTION	Change set number to a value between 1 and 255 inclusive.
8	**MESSAGE**	`SET NUMBERS MUST BE IN ASCENDING SEQUENCE.`
	CAUSE	A set number is less than or equal to the previous set number in the source file. Error can be detected at format time or during a maintenance function.
	ACTION	Change numbers to strict ascending sequence.
9	**MESSAGE**	`MESSAGE NUMBERS MUST BE BETWEEN 1 AND 32766.`
	CAUSE	A message number value is not between 1 and 32,766 inclusive.
	ACTION	Change message number value to a value that is between 1 and 32,766 inclusive.
10	**MESSAGE**	`MESSAGES MUST EITHER CONTAIN ALL NUMBERED OR ALL POSITIONAL PARAMETER SUBSTITUTION CHARACTERS. MIXES NOT ALLOWED.`
	CAUSE	During the scan of the message, GENCAT detected a mix of parameter substitution characters. For example, a message contained numeric substitution characters as well as positional substitution characters.
	ACTION	Change the parameter substitution characters either to all numeric substitution or all positional substitution characters. (Note that this is for each message only.)
11	**MESSAGE**	`MESSAGE NUMBERS MUST BE IN ASCENDING SEQUENCE.`
	CAUSE	A message number was processed that is less than or equal to the previous message number. The message numbers within a set are not in ascending sequence.
	ACTION	Arrange the messages within the set so that their numbers are in strict ascending order.

12	**MESSAGE**	`MESSAGE CONTAINS NON-BLANK CHARACTER IMMEDIATELY FOLLOWING MESSAGE NUMBER.  NON-BLANK CHARACTER ASSUMED TO BE A BLANK.`
	CAUSE	GENCAT detected a non-blank character immediately following the message number in a message. GENCAT replaces this character with a blank.
	ACTION	Insert a blank between the message number and the message text.
13	**MESSAGE**	`EXPECTED ONE OF THE FOLLOWING INPUTS: 0, 1, 2, 3, 4, OR A RETURN.`
	CAUSE	GENCAT detected an incorrect input in response to the first menu (which prompts for a function).
	ACTION	Respond only with 0, 1, 2, 3, 4, or a Return.
14	**MESSAGE**	`EXPECTED ONE OF THE FOLLOWING INPUTS: 0, 1, 2, 3, OR A RETURN.`
	CAUSE	GENCAT detected an incorrect input in response to the menu that prompts for the type of merging it is to perform.
	ACTION	Respond only with 0, 1, 2, 3, or a Return.
15	**MESSAGE**	`EXPECTED AN EXISTENT FILE AS INPUT.`
	CAUSE	The file does not exist on the system.
	ACTION	Either create the file or input the name of a file that does exist on the system.
16	**MESSAGE**	`EXPECTED A UNIQUE, NON-EXISTENT FILE NAME AS INPUT.`
	CAUSE	The file already exists on the system. The name of the file should be one that does not exist on the system.
	ACTION	Purge the file or input the name of a file that does not exist on the system.

17	**MESSAGE**	`EXPECTED A RESPONSE OF "YES" OR "NO" AS INPUT.`
	CAUSE	GENCAT requires a response of either `YES`, `yes`, `NO`, or `no` to the prompt of `SAVE COLLISIONS?`.
	ACTION	Respond with `YES`, `yes`, `NO`, or `no`.
18	**MESSAGE**	`INPUT FILES MUST HAVE EQUAL RECORD SIZES FOR THIS FUNCTION.`
	CAUSE	Source and maintenance files must have equal record sizes if the maintenance file is to modify the source file.
	ACTION	Create a maintenance file that has a record size equal to the record size of the source file.
20	**MESSAGE**	`THE CONSTRUCT OF $DELSET IS NOT ALLOWED IN THE SOURCE.`
	CAUSE	The construct `$DELSET`, which may be used in a maintenance file, was detected in a source file during a maintenance function.
	ACTION	Remove `$DELSET` construct from the source file.
21	**MESSAGE**	`ONLY FIVE (5) POSITIONAL PARAMETER SUBSTITUTIONS ALLOWED PER MESSAGE.`
	CAUSE	GENCAT detected more than five (5) parameter substitution characters in one message. Up to five parameter substitution characters are allowed per message.
	ACTION	Only five (5) or fewer parameter substitution characters per message.
22	**MESSAGE**	`MAINTENANCE FILE MUST BE NUMBERED FOR LINE-NUMBER MERGES.`
	CAUSE	The maintenance file is an unnumbered file. The maintenance file must be a numbered file if it is to be used in a line-number merge.
	ACTION	Number the maintenance file if the file is to be used in a line-number merge.

23	**MESSAGE**	`SOURCE FILE MUST BE NUMBERED FOR LINE-NUMBER MERGES.`
	CAUSE	The source file is an unnumbered file. The source file must be a numbered file if it is to be used in a line-number merge.
	ACTION	Number the source file if the file is to be used in a line-number merge.
24	**MESSAGE**	`SOURCE FILE CANNOT CONTAIN FORMS OF $EDIT.`
	CAUSE	During a line-number merge, GENCAT examines the source file for `$EDIT` and `$EDIT VOID=` constructs. These are not allowed since if collision files are to be used, an ambiguity would exist if the `$EDIT` and `$EDIT VOID=` were left in the source file.
	ACTION	Remove all occurrences of `$EDIT` and `$EDIT VOID=` from the source file.
25	**MESSAGE**	`SEQUENCE NUMBER IN $EDIT VOID RECORD CONTAINS TOO MANY DIGITS.  EIGHT IS THE MAXIMUM.`
	CAUSE	The value following the `$EDIT VOID=` may have a maximum of eight place holders.
	ACTION	Re-evaluate this value and correct it, as it represents a line number.
26	**MESSAGE**	`FILE IS NOT A FORMATTED FILE.`
	CAUSE	GENCAT can only expand formatted catalogs (for example, files formatted by GENCAT).
	ACTION	Format the file using GENCAT.
27	**MESSAGE**	`SET RECORD IS REQUIRED BEFORE A MESSAGE RECORD IS FORMATTED.`
	CAUSE	A message was found before the set number was defined.
	ACTION	Place the message in a set or place a set number before the message.

28	**MESSAGE**	`VALUE IN RIGHT BYTE OF KANJI CHARACTER IS INVALID.`
	CAUSE	Your message contains special escape sequences provided by HP that are used for research and development activities. These special escape sequences are not supported by Hewlett-Packard and Hewlett-Packard assumes no responsibility for their use.
	ACTION	For messages 28 through 32, consult your Hewlett-Packard representative, or remove all occurrences of the form `esc$<terminator>` or `ESC(<terminator>` from your message catalog. Where `ESC` is the escape character, `<terminator>` is "@" or "A" through "Z".
29	**MESSAGE**	`SCAN COMPLETED WITH NO CLOSING KANJI ESCAPE SEQUENCE. EXPECTS A CLOSING KANJI ESCAPE SEQUENCE TO TERMINATE KANJI CHARACTER SEQUENCE.`
	CAUSE	See Message Number 28.
	ACTION	See Message Number 28.
30	**MESSAGE**	`INCOMPLETE KANJI CLOSING ESCAPE SEQUENCE DETECTED.`
	CAUSE	See Message Number 28.
	ACTION	See Message Number 28.
31	**MESSAGE**	`VALUE IN LEFT-BYTE OF KANJI CHARACTER IS INVALID.`
	CAUSE	See Message Number 28.
	ACTION	See Message Number 28.
32	**MESSAGE**	`VALUE IN PARAMETER SECTION OF KANJI ESCAPE SEQUENCE IS INVALID.  EXPECTED A STRING OF DIGITS.`
	CAUSE	See Message Number 28.
	ACTION	See Message Number 28.

B

Catalogs Formatted With MAKECAT

Some older catalogs have been built using the `MAKECAT.PUB.SYS` program. Although that method is not the most efficient or convenient, the catalogs may still be used under MPE XL. The following information is included for your convenience. If you are going to create a catalog, refer to Chapters 2-4 for information about application message catalogs.

Creating a Formatted Catalog

To create a catalog, you must first create a source file and then format it with the MAKECAT utility.

Source files for MAKECAT are `EDIT/V` files, with set numbers 1-255, and message numbers 1-32,766. The MAKECAT utility is used to build message catalogs (and HELP catalogs). The program's input file has the formal designator `INPUT`. MAKECAT reads from the input file and builds a temporary file with the formal designator `CATALOG` that exists in your group and account. MAKECAT also renames any old temporary `CATALOG` to `CAT`*nn*, using an incremental numbering scheme (for example, `CAT1, CAT2`).

To use MAKECAT to build your own message catalog from a source named `SOURCE`, enter:

```
:FILE INPUT=SOURCE
:RUN MAKECAT.PUB.SYS
VALID MESSAGE CATALOG
:SAVE CATALOG
```

Accessing the Message Catalog

After creating a formatted catalog, you access it from your program using system intrinsics. You first open the catalog with `HPFOPEN`, then you read and output the messages using `GENMESSAGE`. After you have finished outputting messages, close the catalog with `FCLOSE`. For detailed information on these intrinsic, refer to the *MPE XL Intrinsics Reference Manual* (32650-90028).

Modifying the Message Catalog

To modify your message catalog:

1. Text your source file into the editor
2. Make the desired changes.
3. Keep the file under a new name and exit the Editor
4. If `SOURCE1` is the name of your new source file, enter:

```
:FILE INPUT=SOURCE1
:RUN MAKECAT.PUB.SYS
VALID MESSAGE CATALOG
:SAVE CATALOG
```

Converting MAKECAT Formatted Catalogs

To increase the efficiency of message catalog accessing, you may want to convert your MAKECAT formatted message catalog to a GENCAT formatted catalog. GENCAT is the newer facility for creating formatted message catalogs. It is faster, allows more set numbers, allows you to modify your catalogs easily, and permits catalog rollback. If you decide to perform the conversion you must perform two tasks:

- Alter the Catalog
- Change the Intrinsic Access

Altering the Catalog

Changing a MAKECAT formatted file to a GENCAT formatted file is easy; just run the source through the GENCAT program to format it. For details about GENCAT and how it formats source files, refer to Chapter 3 of this manual.

Changing the Intrinsic Access

To access a GENCAT formatted catalog, change all the `HPFOPEN` calls in the program to the catalog to `CATOPEN` calls with the correct parameters. `GENMESSAGE` intrinsic calls must be changed to `CATREAD`; change `FCLOSE` to `CATCLOSE` when closing the catalog. For explanations of the intrinsics, refer to the *MPE XL Intrinsics Reference Manual* (32650-90028). Also refer to Chapter 3, "Accessing Application Message Catalogs".

GENCAT MAKECAT Comparison

To allow you to compare the GENCAT and the MAKECAT utilities, the differences between GENCAT and MAKECAT are listed in tabular form below.

Table B-1. MAKECAT/GENCAT Comparison

FEATURES	MAKECAT	GENCAT
Access Methods	The `FOPEN`, `GENMESSAGE`, and `FCLOSE` intrinsics open, access, and close formatted MAKECAT catalogs.	`CATOPEN`, `CATREAD`, and `CATCLOSE` intrinsics open, access and close formatted GENCAT catalogs.
Formatting	Places an internal directory in the file's user labels. The file is formatted in place without creating a new file.	A source message file is formatted into another file, leaving the original source intact. The application uses the formatted file. The original source file can be purged. The formatted file can be expanded to restore the original source file.
Function	Converts or formats HELP and message files into catalogs. Installs system message catalog, using the `BUILD` entry point.	Formats application message catalogs. Provides maintenance facility to modify existing source catalogs. Provides capability of expanding a formatted file back into the original source file.
Input	The name of a file must be entered in a file equation. `:FILE INPUT=<`*your file*`>`.	GENCAT prompts the user for the name of a file.
Literal Character	Not supported.	The tilde "~" serves as a literal character, causing the character which immediately follows it to be treated as text.
Messages	The message number range per set is 1-32,766.	The message number range per set is 1-32,766.
Numerical Parameters	Not supported.	Up to 5 numerical parameters can be contained in a message.

Table B-1. MAKECAT/GENCAT Comparison (Cont.)

FEATURES	MAKECAT	GENCAT
Output	Saves the formatted file as a temporary file with the name `CATALOG`.	GENCAT prompts the user for the name of the formatted file. The file is saved as a permanent file.
Processing	Formats more quickly than GENCAT.	GENCAT verifies each message for correct parameter substitution characters. Manipulates two temporary files while formatting the source file.
Record Format	Accepts source files of any size, but the file it saves has a record size of 80 bytes. The system message catalog is fixed binary. An application catalog is fixed ASCII.	Accepts source catalog files with record sizes from 40 to 256 bytes. The formatted file has a record size of 128 words, and is fixed binary. When a formatted catalog is expanded into a source catalog, the new source catalog is fixed ASCII with a record size identical to the original source catalog.
		When maintenance is being performed, both the source file and the maintenance file must be of equal lengths in fixed ASCII. The resulting source file, and collision file, if specified will be fixed ASCII, and their record sizes will equal the record size of the original source file.
Sets	The set directive is `$SET`. The set number range for a catalog is 1-63.	The set directive can be `$SET` or `$set`. The set number range for a source catalog is 1-255.
User Interface	The user must know which entry points to use and when to use them. Files are input via file equations. Error messages require user interpretation.	GENCAT is menu-driven. The menus originate from a catalog. Each prompt has HELP text associated with it. Error messages are self-explanatory.

C

COBOL Progam Examples

Example of Accessing an Application Message Catalog

This example shows the access of the sample catalog called FORMAT (created in the Chapter 2); the source of this sample catalog (SOURCE) is listed below.

```
$SET 1 Prompts
1 ENTER FIRST NAME
2 ENTER LAST NAME
$
$
$set 2 Error Messages
1 NAME NOT ON DATA BASE
2 ILLEGAL INPUT
3 AN ERROR OCCURRED DURING THE LOADING %
OF THE DATA BASE.
98 THE NUMBER OF FILES &
DOES NOT MATCH THE &
SYSTEM'S CALCULATIONS.
$
$set 13 Run-Time Messages
400 INPUT FROM ! ON TERMINAL NUMBER !
401 INPUT FROM TERMINAL NUMBER !2 BY !1
```

The program uses message 1 in set 1 to prompt for a first name, substitutes the name in message 400 of set 13, and outputs the message. All output is written to the terminal.

```
001000 IDENTIFICATION DIVISION.
001001 PROGRAM-ID. CATMSSG.
001003*
001004 ENVIRONMENT DIVISION.
001005 DATA DIVISION.
001006 WORKING-STORAGE SECTION.
001007 77 CAT-INDEX                     PIC S9(9) COMP.
001008 01 CAT-STATUS.
001009    03 CAT-STATUS-1               PIC S9(4) COMP.
001010    03 CAT-STATUS-2               PIC S9(4) COMP.
001011 77 CAT-FILE                      PIC X(20).
001012 77 MSGLENGTH                     PIC S9(4) COMP.
001013 77 SETNUM                        PIC S9(4) COMP.
001014 77 MSGNUM                        PIC S9(4) COMP.
001015 77 PARM-1                        PIC X(5).
001016 77 ASCII-NULL                    PIC X VALUE %0.
001017 77 PARM-2                        PIC X(1).
001018 77 ASCII-NULL-1                  PIC X VALUE %0.
```

```
001019 77 MSGDESTINATION                   PIC S9(4) COMP.
001020*
001021 PROCEDURE DIVISION.
001022*
001023 START-OF-PROGRAM.
001024     PERFORM OPEN-A-CATATLOG.
001025     PERFORM READ-A-CATATLOG.
001026     PERFORM CLOSE-A-CATATLOG.
001027     STOP RUN.
001028*
001029 OPEN-A-CATATLOG.
001030     MOVE "FORMAT%" TO CAT-FILE.
001031     CALL INTRINSIC "CATOPEN" USING  CAT-FILE,
001032                                     CAT-STATUS
001033                              GIVING CAT-INDEX.
001034*
001035* CHECK CAT-STATUS FOR SUCCESS
001036*
001037 READ-A-CATATLOG.
001038     MOVE 1 TO SETNUM.
001039     MOVE 1 TO MSGNUM.
001040     MOVE 0 TO MSGDESTINATION.
001041     CALL INTRINSIC "CATREAD" USING CAT-INDEX,
001042                                    SETNUM,
001043                                    MSGNUM,
001044                                    CAT-STATUS,
001045                                    \\, \\,
001046                                    \\,
001047                                    \\, \\, \\, \\,
001048                                    MSGDESTINATION.
001051     ACCEPT PARM-1.
001052     MOVE '3' TO PARM-2.
001053     MOVE 13 TO SETNUM.
001054     MOVE 400 TO MSGNUM.
001055     CALL INTRINSIC "CATREAD" USING  CAT-INDEX,
001056                                     SETNUM,
001057                                     MSGNUM,
001058                                     CAT-STATUS,
001059                                     \\, \\,
001060                                     PARM-1,
001061                                     PARM-2,
001062                                     \\, \\, \\,
001063                                     MSGDESTINATION.
001064*
001065* CHECK ERRORNUM FOR SUCCESS
001066*
001067 CLOSE-A-CATATLOG.
001068     CALL INTRINSIC "CATCLOSE" USING CAT-INDEX,
001069                                     CAT-STATUS.
001070*
```

```
001071* CHECK CONDITION CODE FOR SUCCESS
001072*
```

When this program is executed, the output is:

```
ENTER FIRST NAME

MARY

INPUT FROM MARY ON TERMINAL NUMBER 3
```

Example of Accessing the CM Error Message Catalog

The following listing is a COBOL program that inserts the value 95 into message number 201 in message set 1 in the message catalog `CATALOG.PUB.SYS`. The message is output to $STDLIST. The accessed portion of the message catalog is:

```
     $SET 1
        .
        .
        .
     210 SYSTEM LOG FILE NUMBER ! is ON
```

```
001000 IDENTIFICATION DIVISION.
001100 PROGRAM-ID. CM-MSGCAT.
001300*
001400 ENVIRONMENT DIVISION.
001500 DATA DIVISION.
001600 WORKING-STORAGE SECTION.
001700 77 FILE-NUM                PIC S9(9) COMP.
001800 77 DESIGNATOR              PIC S9(9) COMP VALUE 2.
001900 77 DOMAIN                  PIC S9(9) COMP VALUE 3.
002000 77 MULTIREC                PIC S9(9) COMP VALUE 15.
002100 77 BUFFERED                PIC S9(9) COMP VALUE 46.
002200 77 ASCII-BINARY            PIC S9(9) COMP VALUE 53.
002300 77 HPFOPEN-STATUS          PIC S9(9) COMP.
002400 77 HPFOPEN-FILE            PIC X(20).
002500 77 PERM-DOMAIN             PIC S9(9) COMP.
002600 77 MULTIREC-ON             PIC S9(9) COMP.
002700 77 INHIBITED               PIC S9(9) COMP.
002800 77 ASCII-FILE              PIC S9(9) COMP.
002900 77 MSGLENGTH               PIC S9(4) COMP.
003000 77 SETNUM                  PIC S9(4) COMP.
003100 77 MSGNUM                  PIC S9(4) COMP.
003200 77 PARM-MASK               PIC S9(4) COMP.
003300 77 PARM-1                  PIC S9(9) COMP.
003400 77 MSGDESTINATION          PIC S9(4) COMP.
003500 77 ERRORNUM                PIC S9(4) COMP.
003600 77 DISPOSITION             PIC S9(4) COMP.
003700 77 CLOSE-SECURITY          PIC S9(4) COMP.
```

```
003800*
003900 PROCEDURE DIVISION.
004000*
004100 START-OF-PROGRAM.
004200     PERFORM OPEN-CATALOG.
004300     PERFORM READ-CATALOG.
004400     PERFORM CLOSE-CATALOG.
004500     STOP RUN.
004600*

004700 OPEN-CATALOG.
004900     MOVE "%CATALOG.PUB.SYS%" TO HPFOPEN-FILE.
005000     MOVE 1 TO PERM-DOMAIN.
005100     MOVE 1 TO MULTIREC-ON.
005200     MOVE 1 TO INHIBITED.
005300     MOVE 1 TO ASCII-FILE.
005400     CALL INTRINSIC "HPFOPEN" USING  FILE-NUM,
005500                                     HPFOPEN-STATUS,
005600                                     DESIGNATOR,
005700                                     HPFOPEN-FILE,
005800                                     DOMAIN,
005900                                     PERM-DOMAIN,
006000                                     MULTIREC,
006100                                     MULTIREC-ON,
006200                                     BUFFERED,
006300                                     INHIBITED,
006400                                     ASCII-BINARY,
006500                                     ASCII-FILE.
006600*
006610* CHECK HPFOPEN-STATUS FOR SUCCESS
006620*
006700 READ-CATALOG.
006900     MOVE 1 TO SETNUM.
007000     MOVE 201 TO MSGNUM.
007100     MOVE 0 TO MSGDESTINATION.
007200     MOVE 95 TO PARM-1.
007300     MOVE %13333 TO PARM-MASK.
007400     CALL INTRINSIC "GENMESSAGE" USING \FILE-NUM\,
007500                                       \SETNUM\,
007600                                       \MSGNUM\,
007700                                       \\, \\,
007800                                       \PARM-MASK\,
007900                                       \PARM-1\,
008000                                       \\, \\, \\, \\,
008100                                       \MSGDESTINATION\,
008200                                       ERRORNUM.
008400*
008410* CHECK ERRORNUM FOR SUCCESS
008420*
008500 CLOSE-CATALOG.
008700     MOVE 0 TO DISPOSITION.
```

```
008800     MOVE 0 TO CLOSE-SECURITY.
008900     CALL INTRINSIC "FCLOSE" USING FILE-NUM,
009000                                   DISPOSITION,
009100                                   CLOSE-SECURITY.
009200*
009300* CHECK CONDITION CODE FOR SUCCESS
009400*
```

When this program is executed, the output is:

```
SYSTEM LOG FILE NUMBER 92 IS ON
```

Example of Accessing the NM Error Message Catalog

The following listing is a COBOL program that inserts the value 42 into message number 8 in message set 201 in the message catalog `SYSCAT.PUB.SYS`. The message is output to $STDLIST. The accessed portion of the message catalog is:

```
$SET 201
   .
   .
   .
008 The value passed for parameter #! is invalid.
```

The value passed for parameter #42 is invalid.

```
001000 IDENTIFICATION DIVISION.
001010 PROGRAM-ID. NM-MSGCAT.
001030*
001040 ENVIRONMENT DIVISION.
001050 DATA DIVISION.
001060 WORKING-STORAGE SECTION.
001070 77 CAT-INDEX                   PIC S9(9) COMP.
001080 01 CAT-STATUS.
001090    03 CAT-STATUS-1             PIC S9(4) COMP.
001100    03 CAT-STATUS-2             PIC S9(4) COMP.
001110 77 CAT-FILE                    PIC X(20).
001120 77 MSGLENGTH                   PIC S9(4) COMP.
001130 77 SETNUM                      PIC S9(4) COMP.
001140 77 MSGNUM                      PIC S9(4) COMP.
001150 77 PARM-1                      PIC X(3).
001160 77 ASCII-NULL                  PIC X VALUE %0.
001170 77 MSGDESTINATION              PIC S9(4) COMP.
001180*
001190 PROCEDURE DIVISION.
001200*
001210 START-OF-PROGRAM.
001220     PERFORM OPEN-SYSCAT.
001230     PERFORM READ-SYSCAT.
001240     PERFORM CLOSE-SYSCAT.
001250     STOP RUN.
```

```
001260*
001270 OPEN-SYSCAT.
001280     MOVE "SYSCAT.PUB.SYS%" TO CAT-FILE.
001290     CALL INTRINSIC "CATOPEN" USING  CAT-FILE,
001300                                     CAT-STATUS
001310                              GIVING CAT-INDEX.
001320*
001330* CHECK CAT-STATUS FOR SUCCESS
001340*
001350 READ-SYSCAT.
001360     MOVE 221 TO SETNUM.
001370     MOVE 8 TO MSGNUM.
001380     MOVE 0 TO MSGDESTINATION.
001390     MOVE "42" TO PARM-1.
001400     CALL INTRINSIC "CATREAD" USING CAT-INDEX,
001410                                    SETNUM,
001420                                    MSGNUM,
001430                                    CAT-STATUS,
001440                                    \\, \\,
001450                                    PARM-1,
001460                                    \\, \\, \\, \\,
001470                                    MSGDESTINATION.
001480*
001490* CHECK ERRORNUM FOR SUCCESS
001500*
001510 CLOSE-SYSCAT.
001520     CALL INTRINSIC "CATCLOSE" USING CAT-INDEX,
001530                                     CAT-STATUS.
001540*
001550* CHECK CONDITION CODE FOR SUCCESS
```

When this program is executed, the output is:

```
The value passed for parameter #42 is invalid.
```

D

FORTRAN Progam Examples

Example of Accessing an Application Message Catalog

This example shows the access of the sample catalog called FORMAT (created in the Chapter 2); the source of this sample catalog (SOURCE) is listed below.

```
$SET 1 Prompts
1 ENTER FIRST NAME
2 ENTER LAST NAME
$
$
$set 2 Error Messages
1 NAME NOT ON DATA BASE
2 ILLEGAL INPUT
3 AN ERROR OCCURRED DURING THE LOADING %
OF THE DATA BASE.
98 THE NUMBER OF FILES &
DOES NOT MATCH THE &
SYSTEM'S CALCULATIONS.
$
$set 13 Run-Time Messages
400 INPUT FROM ! ON TERMINAL NUMBER ! RESULTS IN SUCCESSFUL &
OPERATION
401 INPUT FROM TERMINAL NUMBER !2 BY !1 RESULTS IN SUCCESSFUL &
OPERATION
```

The program uses message 1 in set 1 to prompt for a first name, substitutes the name in message 400 of set 13, and outputs the message. All output is written to the terminal.

```
$control standard_level system
*
      program MSGCAT
*
      system intrinsic QUIT
*
*
      logical STATUS_RETURN
*
      character DESIGNATOR*20
*
      call OPEN_A_CATALOG (STATUS_RETURN)
      if (STATUS_RETURN) go to 10
      print *,'OPEN_A_CATALOG Failed.  Terminating.'
      call QUIT (1)
*
```

```
   10 call READ_A_CATALOG (STATUS_RETURN)
*
      if (STATUS_RETURN) go to 20
      print *,'READ_A_CATALOG Failed.  Terminating.'
      call QUIT (2)
*
   20 call CLOSE_A_CATALOG (STATUS_RETURN)
      if (STATUS_RETURN) go to 30
      print *,'CLOSE_A_CATALOG Failed.  Terminating'
      call QUIT (3)
*
   30 stop
      end

*
*******************************************************
*
      subroutine OPEN_A_CATALOG (STATUS_RETURN)
*
      system intrinsic CATOPEN
*
      integer CATINDEX
     2       ,CATSTATUS(2)
*
      logical STATUS_RETURN
*
      character DESIGNATOR*20
*
      STATUS_RETURN = .true.
      DESIGNATOR = 'FORMAT'
      CATINDEX = CATOPEN (DESIGNATOR, CATSTATUS)
      if (CATSTATUS(1) .ne. 0) STATUS_RETURN = .false.
*
      return
      end
*
*******************************************************
*
      subroutine READ_A_CATALOG (STATUS_RETURN)
*
      system intrinsic CATREAD
*
      integer*2 SETNUM
     2         ,MSGNUM
     3         ,MSGDEST
     4         ,MSGLENGTH
     5         ,CATSTATUS(2)
*
      integer*4 CATINDEX
     2         ,DUMY
*
```

```
      logical STATUS_RETURN
*
      character PARM1*6
     2         ,PARM2*2
*
      STATUS_RETURN = .true.
      SETNUM     = 1
      MSGNUM     = 1
      MSGDEST = 0
      MSGLENGTH = CATREAD (CATINDEX
     2                    ,SETNUM
     3                    ,MSGNUM
     4                    ,CATSTATUS,,,
     5                ,,,,,MSGDEST)
      if (CATSTATUS(1) .ne. 0) STATUS_RETURN = .false.
*
      SETNUM = 13
      MSGNUM = 400
      read *,PARM1(1:5)
      write (PARM1(6:6),15) 0b
      PARM2(1:1) = '3'
      write (PARM2(2:2),15) 0b
   15 format (r1)
*
      MSGLENGTH = CATREAD (CATINDEX
     2                    ,SETNUM
     3                    ,MSGNUM
     4                    ,CATSTATUS
     5                  ,,,PARM1
     6                    ,PARM2
     7                 ,,,,MSGDEST)
      if (CATSTATUS(1) .ne. 0) STATUS_RETURN = .false.
      return
      end
*
*******************************************************
*
      subroutine CLOSE_A_CATALOG (STATUS_RETURN)
*
      system intrinsic CATCLOSE
*
      integer*2 CATSTATUS(2)
*
      integer*4 CATINDEX
*
      logical STATUS_RETURN
*
      STATUS_RETURN = .true.
      call CATCLOSE (CATINDEX
     2              ,CATSTATUS)
```

```
        if (CATSTATUS(1) .ne. 0) STATUS_RETURN = .false.
        return
        end
```

When this program is executed, the output is:

```
ENTER FIRST NAME

MARY

INPUT FROM MARY ON TERMINAL NUMBER 3
```

Example of Accessing the CM Error Message Catalog

The following listing is a FORTRAN program that inserts the value 95 into message number 201 in message set 1 in the message catalog `CATALOG.PUB.SYS`. The message is output to $STDLIST. The accessed portion of the message catalog is:

```
     $SET 1
        .
        .
        .
     210 SYSTEM LOG FILE NUMBER ! IS ON
```

```
$control standard_level system
      program CM_MSGCAT
*
      system intrinsic QUIT
*
      integer*2    MSGLEN
     2             ,SETNUM
     3             ,MSGNUM
     4             ,PARMASK
     5             ,DESTINATION
*
      integer*4    PARAM1
     2             ,FILENUM
*
      logical      STATUS_RETURN
*
      character*20 FILENAME
*
*     Initialize variables to open file.
*
      FILENAME      = '%CATALOG.PUB.SYS%'
      STATUS_RETURN = .true.
*
*     Open the file.
*
      call OPEN_CATALOG (FILENAME
     2                 ,FILENUM
     3                 ,STATUS_RETURN)
*
      if (STATUS_RETURN) go to 10
      print *,'Open failed.  Terminating.'
      call QUIT (1)
*
*     Initialize variables to read message.
*
   10 SETNUM        = 1
      MSGNUM        = 201
      DESTINATION   = 0
      PARMASK       = 13333B
      PARAM1        = 95
      STATUS_RETURN = .true.
*
*     Generate the message, which will be displayed on $STDLIST
*
      call READ_CATALOG (FILENUM
     2                 ,SETNUM
     3                 ,MSGNUM
     4                 ,DESTINATION
     5                 ,PARMASK
     6                 ,PARAM1
     7                 ,STATUS_RETURN)
      if (STATUS_RETURN) go to 20
      print *,'Read failed.  Terminating.'
      call QUIT (2)
*
*     Close the catalog with default disposition and unrestricted access
*
   20 call CLOSE_CATALOG (FILENUM
     2                  ,STATUS_RETURN)
      if (STATUS_RETURN) go to 30
      print *,'Close failed.  Terminating.'
      call QUIT (3)
*
*     Normal end of program
*
   30 stop
      end
*
************************************************************
*
      subroutine OPEN_CATALOG (FILENAME
     2                       ,FILENUM
     3                       ,STATUS_RETURN)
*
      system intrinsic HPFOPEN
*
```

```
      integer STATUS
     2       ,PERM
     3       ,ON
     4       ,INHIBITED
     5       ,ASCII_FILE
     6       ,DESIGNATOR
     7       ,DOMAIN
     8       ,MULTIREC
     9       ,BUFFERING
     A       ,ASCII_BINARY
     B       ,FILENUM
*
      logical STATUS_RETURN
*
      character*20 FILENAME
*
*     Initialize variables
*
      DESIGNATOR    = 2
      DOMAIN        = 3
      MULTIREC      = 15
      BUFFERING     = 46
      ASCII_BINARY  = 53
      PERM          = 1
      ON            = 1
      INHIBITED     = 1
      ASCII_ENTITY = 1
      STATUS_RETURN = .true.
*
*     Open the file
*
      call HPFOPEN (FILENUM,      STATUS
     2             ,DESIGNATOR,   FILENAME
     3             ,DOMAIN,       PERM
     4             ,MULTIREC,     ON
     5             ,BUFFERING,    INHIBITED
     6             ,ASCII_BINARY, ASCII_FILE)
*
      if (STATUS .ne. 0) STATUS_RETURN = .false.
*
      return
      end
*
************************************************************
*
      subroutine READ_CATALOG (FILENUM
     2                        ,SETNUM
     3                        ,MSGNUM
     4                        ,DESTINATION
     5                        ,PARMASK
     6                        ,PARAM1
     7                        ,STATUS_RETURN)
*
      system intrinsic GENMESSAGE
*
      integer*2 MSGLENGTH
     2         ,SETNUM
     3         ,MSGNUM
     4         ,PARMASK
     5         ,DESTINATION
     6         ,ERRORNUM
*
      integer*4 PARAM1
     2         ,FILENUM
*
      logical   STATUS_RETURN
*
*     Initialize variables
*
      STATUS_RETURN = .true.
*
*     Generate the message
*
      MSGLENGTH = GENMESSAGE (FILENUM
     2                       ,SETNUM
     3                       ,MSGNUM
     4                     ,,,PARMASK
     5                       ,PARAM1
     6                   ,,,,,DESTINATION
     7                       ,ERRORNUM)
*
      if (ERRORNUM .ne. 0) STATUS_RETURN = .false.
*
      return
      end
*
************************************************************
*
      subroutine CLOSE_CATALOG (FILENUM
     2                         ,STATUS_RETURN)
*
      system intrinsic FCLOSE
*
      integer*2 DISPOSITION
     2         ,SECURITYCODE
*
      integer*4 FILENUM
*
      logical   STATUS_RETURN
*
*     Initialize variables
```

```
*
      DISPOSITION   = 0
      SECURITYCODE  = 0
      STATUS_RETURN = .true.
*
*     Close file
*
      call FCLOSE (FILENUM
     2            ,DISPOSITION
     3            ,SECURITYCODE)
*
      if (ccode()) 10,20,10
   10 STATUS_RETURN = .false.
*
   20 return
      end
```

When this program is executed, the output is:

```
SYSTEM LOG FILE NUMBER 92 IS ON
```

Example of Accessing the NM Error Message Catalog

The following listing is a FORTRAN program that inserts the value 42 into message number 8 in message set 201 in the message catalog SYSCAT.PUB.SYS. The message is output to $STDLIST. The accessed portion of the message catalog is:

```
     $SET 201
        .
        .
        .
     008 The value passed for parameter #! is invalid.
```

```
$control standard_level system
*
      program NM_MSGCAT
*
      system intrinsic QUIT
*
      integer*2    SETNUM
     2            ,MSGNUM
     3            ,MSGDEST
     4            ,MSGLENGTH
*
      integer CATINDEX
     2       ,CATSTATUS(2)
*
      logical STATUS_RETURN
*
      character DESIGNATOR*20
     2        ,PARM1*3
*
      DESIGNATOR = 'SYSCAT.PUB.SYS'
      call OPEN_SYSCAT (DESIGNATOR
     2                 ,CATINDEX
     3                 ,STATUS_RETURN)
      if (STATUS_RETURN) go to 10
      print *,'OPEN_SYSCAT Failed.  Terminating.'
      call QUIT (1)
*
   10 SETNUM     = 221
      MSGNUM     = 8
      PARM1(1:2) = '42'
      write (PARM1(3:3),15) 0b
   15 format (r1)
      MSGDEST = 0
*
      call READ_SYSCAT (CATINDEX
     2                 ,SETNUM
     3                 ,MSGNUM
     4                 ,PARM1
     5                 ,MSGDEST
     6                 ,STATUS_RETURN)
*
      if (STATUS_RETURN) go to 20
      print *,'READ_SYSCAT Failed.  Terminating.'
      call QUIT (2)
*
   20 call CLOSE_SYSCAT (CATINDEX
     2                  ,STATUS_RETURN)
      if (STATUS_RETURN) go to 30
      print *,'CLOSE_SYSCAT Failed.  Terminating'
      call QUIT (3)
*
   30 stop
      end
*
*******************************************************
*
      subroutine OPEN_SYSCAT (DESIGNATOR
     2                       ,CATINDEX
     3                       ,STATUS_RETURN)
*
      system intrinsic CATOPEN
*
      integer CATINDEX
     2       ,CATSTATUS(2)
*
      logical STATUS_RETURN
*
```

```
      character DESIGNATOR*20
*
      STATUS_RETURN = .true.
      CATINDEX = CATOPEN (DESIGNATOR, CATSTATUS)
      if (CATSTATUS(1) .ne. 0) STATUS_RETURN = .false.
*
      return
      end
*
*******************************************************
*
      subroutine READ_SYSCAT (CATINDEX
     2                       ,SETNUM
     3                       ,MSGNUM
     4                       ,PARM1
     5                       ,MSGDEST
     6                       ,STATUS_RETURN)
*
      system intrinsic CATREAD
*
      integer*2 SETNUM
     2         ,MSGNUM
     3         ,MSGDEST
     4         ,MSGLENGTH
     5         ,CATSTATUS(2)
*
      integer*4 CATINDEX
     2         ,DUMY
*
      logical STATUS_RETURN
*
      character PARM1*3
*
      STATUS_RETURN = .true.
      MSGLENGTH = CATREAD (CATINDEX
     2                    ,SETNUM
     3                    ,MSGNUM
     4                    ,CATSTATUS
     5                  ,,,PARM1
     6                ,,,,,MSGDEST)
      if (CATSTATUS(1) .ne. 0) STATUS_RETURN = .false.
      return
      end
*
*******************************************************
*
      subroutine CLOSE_SYSCAT (CATINDEX
     2                        ,STATUS_RETURN)
*
      system intrinsic CATCLOSE
```

```
*
      integer*2 CATSTATUS(2)
*
      integer*4 CATINDEX
*
      logical STATUS_RETURN
*
      STATUS_RETURN = .true.
      call CATCLOSE (CATINDEX
     2              ,CATSTATUS)
      if (CATSTATUS(1) .ne. 0) STATUS_RETURN = .false.
      return
      end
```

When this program is executed, the output is:

```
The value passed for parameter #42 is invalid
```

Index