User Logging Programmer's Guide

900 Series HP 3000 Computer Systems

HP Part No. 32650-90027
Printed in U.S.A. 19871101

U0788

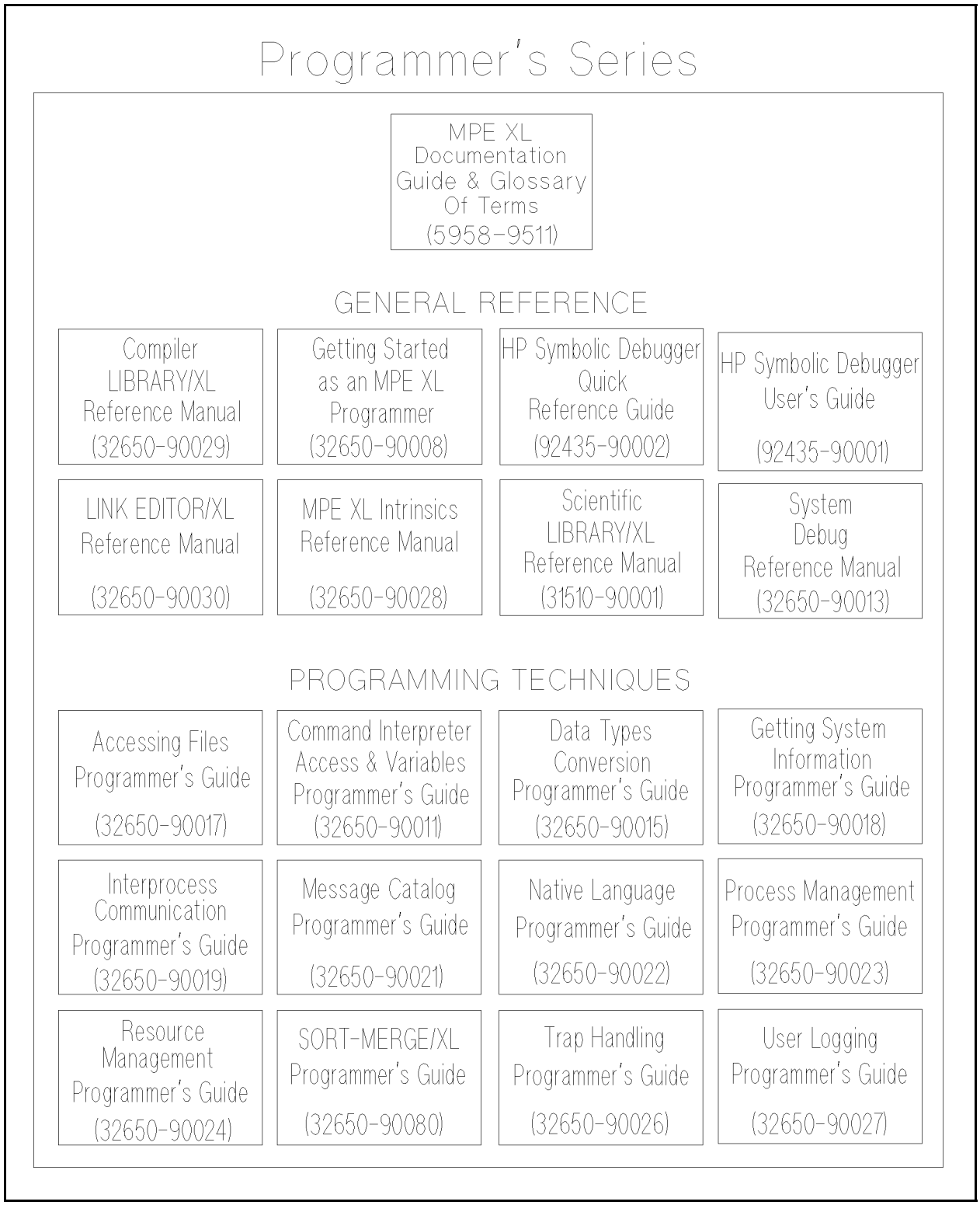
Hewlett-Packard Company
3000 Hanover Street
Palo Alto, CA 94304 U.S.A.

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
Update #1	July 1988	A.10.00

Documentation Map



Preface

The User Logging Programmer's Guide describes the MPE/iX User Logging Facility. It explains the differences between user logging in MPE/iX and MPE VE Operating Systems. Serial Logging, disc logging, and application design are discussed.

The manuals listed below have been referenced in this manual and provide additional information:

MPE/iX Commands Reference Manual (32650-90003)

MPE/iX Intrinsics Reference Manual (32650-90028)

MPE VE System Operations and Resource Management Reference Manual (32033-90005)

This manual contains the following information:

Chapter 1 **Introduction** introduces the subject matter contained in this manual, the types of applications you would want to use User Logging for, and the differences between MPE V/E and MPE/iX.

Chapter 2 **User Logging Intrinsics** introduces the programmatic use of its special intrinsics.

Chapter 3 **User Logging Commands** introduces the commands that may be used at a system level to perform User Logging functions.

Chapter 4 **The User Logging Process** explains the media to be used, how to create necessary files, and how to control the logging process.

Chapter 5 **User Logging in an Application** explains, in detail, the use of the User Logging intrinsics.

Chapter 6 **Recovery** explains the recovery process and how to use the different logging records.

Appendix A **Suggested User Logging Procedure** gives a basic outline of what information is required to implement a User Logging procedure.

Appendix B **Record Formats** explains the contents of the logging records.

Appendix C **User Logging Error Codes** lists all error codes returned by User Logging intrinsics.

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`:

`COMMAND` { `ON` / `OFF` }

`[ ]` 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.

`COMMAND` *filename* [`OPTION` / *parameter*]

[...]

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:

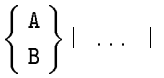
[,*parameter*][...]

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.



...

In an example, horizontal or vertical ellipses indicate where portions of an example have been omitted.

In a syntax statement, the space symbol ␣ shows a required blank. In the following example, *parameter* and *parameter* must be separated with a blank:

(*parameter*)␣(*parameter*)

KEY

The symbol KEY 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, CTRLY means that you press the control key and the Y key simultaneously.

Contents

Tables

1

Introduction

The MPE/iX User Logging facility provides a method for recording events (for example, to record additions and modifications to user databases and files). MPE/iX commands and intrinsics are used to setup logfiles and record to them. Power failure recovery, buffering, and logfile overflow handling can be done for the user automatically. Multiple user processes may share the same logfile and share the system process that manages it. This allows the User Logging overhead to be shared between multiple user processes.

User Logging can be used to keep a record of changes made to data for historical purposes or to aid in the recovery process of lost data due to application and system failures.

User Logging Applications

An application, which uses multiple files, may require multiple actions in order to record information for a single event. For example, an order entry system, where an order is placed for an item must be reflected in files containing the following types of data:

- Shipping data
- The inventory
- The cost of goods sold
- The accounts receivable

If this data is in separate files, each file must be updated with a separate call to `FWRITE`. If the shipping data was updated, but a program abort or system failure occurred before the accounts receivable was updated, then the order might be shipped but never billed. Depending upon the point at which the failure occurred, other files may have inconsistencies in their data such as:

- The item may be shipped, but still be included in inventory.
- The inventory may be reduced, but cost of goods sold was not increased.

The collection of all actions (such as, calls to `FWRITE`) which must be complete, in order for all data involved to be consistent, will be referred to as a single transaction. A transaction is complete only after all of these actions are completed.

This type of application may utilize User Logging to record information about each action which makes up the transaction. The log record could contain information about the transaction and the user could then write a recovery program to reapply to the backup copies of the data structures, the changes for those transactions which completed. This recovery program would not reapply changes which were part of transactions, that did not fully complete. This ensures the preservation of data integrity through a program abort or system failure.

2

User Logging Intrinsics

User Logging is performed programmatically by using the intrinsics listed below. The user that starts an application which calls the User Logging intrinsics, must have User Logging (LG) or System Supervisor (OP) capabilities. Refer to the *Intrinsics Reference Manual* (32650-90028) for the syntax of the following intrinsics:

Note There are no major differences between Native Mode and Compatibility Mode for User Logging.

`BEGINLOG` `BEGINLOG` writes a special record to the user logfile to mark the beginning of a logical transaction:

- The logging memory buffer is flushed to ensure that the record gets to the logfile.
- The *data* parameter can be used to write data to the logfile in the `BEGINLOG` record.
- The *mode* parameter is used to specify WAIT or NOWAIT I/O for the intrinsic.

`CLOSELOG` `CLOSELOG` removes the link to the logging facility:

- A record is written to the logfile to record the close for the user.
- The *mode* parameter is used to specify WAIT or NOWAIT I/O for the intrinsic.

`ENDLOG` `ENDLOG` writes a special record to the user logfile to mark the end of a logical transaction:

- The logging memory buffer is flushed to ensure that the record gets to the logfile.
- The *data* parameter can be used to write data to the logfile in the `ENDLOG` record.
- The *mode* parameter is used to specify WAIT or NOWAIT I/O for the intrinsic.

`FLUSHLOG` `FLUSHLOG` flushes the records in the User Logging memory buffer to the logfile. This ensures that those records will be recoverable in the event of a system failure.

LOGINFO	LOGINFO provides information about an open logfile and the previous logfile (if one exists) in the logfile set, or it may be used to obtain information about the whole logfile set (for example, the total number of records written).
LOGSTATUS	LOGSTATUS provides information about a current, open logfile (for example, to determine the amount of space available in a DISC logfile).
OPENLOG	OPENLOG links the calling application to a User Logging process: ■ The logging identifier specified in the *logid* parameter, must be a valid logging identifier for an active logging process. ■ An index is returned which is used in subsequent intrinsic calls to reference the OPENLOG link to the User Logging process. ■ A record is written to the logfile to identify the user opening the logfile. ■ The *mode* parameter is used to specify WAIT or NOWAIT I/O for the intrinsic.
WRITELOG	WRITELOG writes a record, of the user's data, to a logfile. ■ The *mode* parameter is used to specify WAIT, NOWAIT I/O, or WRITE-and-FLUSH for the intrinsic.

3

User Logging Commands

User Logging can be performed at the system level by using the commands listed below. Refer to the *MPE/iX Commands Reference Manual* (32650-90003) for the syntax and a more detailed description of the following commands:

:ALTLOG	:ALTLOG alters the attributes of an existing User Logging identifier.
:CHANGELOG	:CHANGELOG changes the logfile without stopping or interrupting the logging process. By specifying a device, the user can switch the logging process from logging to a logfile on one device to a logfile on another device.
:GETLOG	:GETLOG establishes a logging identifier and its attributes on the system. The user must supply the name of the logfile and the device class where the logfile resides.
:LISTLOG	:LISTLOG lists currently active logging identifiers on the system, the logfile name and creator, and whether :CHANGELOG is allowed and/or AUTO enabled.
:LOG	:LOG starts, restarts, or stops a User Logging process.
:RELLOG	:RELLOG removes a logging identifier from the system.
:SHOWLOGSTATUS	:SHOWLOGSTATUS provides status information for currently open logfiles and the amount of space available on a DISC logfile.

Note

The :ALTLOG, :GETLOG, CHANGELOG, and :RELLOG commands require the user to have User Logging (LG) or System Operator (OP) capability.

The :LOG command requires the user to have System Operator (OP) capability.

The :LISTLOG command requires the user to have User Logging (LG) capability.

4

The User Logging Process

There are two levels of User Logging; the user's level and the system level. At the user's level there are application and recovery programs that access the logfile. At the system level, there must be a User Logging process running.

The User Logging system process allows up to 256 users to access a single logfile at the same time (configurable at system startup); it handles buffering and power failure recovery, and provides data about the status of the logfile. Before you can write to a logfile, a User Logging process must be running for that logfile. As a system process, the user logging process does not belong to a particular user. Any problems encountered by the User Logging process will be reported to the System Console.

Note Since problems are reported to the System Console and the process must be restarted each time the system is started, responsibility for keeping the process running is usually given to the System Operator.

Which Media to Use

When deciding which media (`DISC` or `TAPE`) to use, consider the availability of disc space versus the availability of a tape drive on the system. Speed is not an issue when logging to `TAPE`, User Logging has an internal buffer (4K words) which brings the logging speed up to that of logging to `DISC`.

Creating the LOGID

The User Logging system process is identifiable to the system by a *logid*. A maximum of 64 logging processes may be defined on the system at one time (configurable at system startup). The *logid* is an 8-character identifier created with the `:GETLOG` command. To create a new *logid*, the logfile name to be used by the process and the media (`DISC` or `TAPE`) it is to reside on, must be specified. Other variables that may be specified include: the group and account of a `DISC` logfile, a password, and what to do when the logfile is full.

Identifying LOGIDs on the System

To find the *logid*s currently known to the system and the attributes of each, use the :LISTLOG command:

```
:LISTLOG

LOGID          CREATOR     CHANGE  AUTO  CURRENT LOG FILE

BON            MARK.MPEM    YES     NO   BONOO3.DOVI.MPEM
KATHY          KATHY.MPEM   YES     YES  KATOO1.SNIDER.MPEM
TEST1          KATHY.MPEM   NO      NO   LOGF.TEST.MPEM
```

A password will be displayed to the *logid*'s creator only if the creator has specified the ;PASS option:

```
:LISTLOG logid;PASS

LOGID          CREATOR     CHANGE  AUTO  CURRENT LOG FILE

TEST1/SECRET   KATHY.MPEM   NO      NO   LOGF.TEST.MPEM
```

Changing Attributes Associated with LOGIDs

All attributes associated with the *logid* can be changed by the creator of the *logid* using the :ALTLOG command when the logging process is not running. An application accessing the logging process supplies the *logid* and password; this allows the other attributes to change without changing an application. A *logid* can be removed from the system by the creator only, using the :RELLOG command.

If a password is specified with :ALTLOG or :GETLOG it will be required each time the User Logging process is accessed. For example, when using the OPENLOG intrinsic. Any user with LG or OP capability which supplies the *logid* and password will be allowed access to the logging process.

Building a DISC Logfile

If the logfile to be used will reside on disc, use the :BUILD command to create the logfile and specify ;CODE=LOG. Ensure the file is large enough to contain one day's data. All other defaults in the :BUILD command may be used for file attributes. Refer to the :BUILD command in the *MPE/iX Commands Reference Manual* (32650-90003).

Controlling the Logging Process

When *logid* has been created and the logfile built, the Operator will control the logging process using the :LOG command. The :LOG command allows the operator to start a logging process to a new logfile, stop a logging process (for example, to change attributes using :ALTLOG, or to shut the system down), or restart a logging process to a logfile which has already been accessed. Problems which may occur when using the :LOG command include:

- The user of the :LOG command does not have OP capabilities or the operator has not done an :ALLOW of the :LOG command for the user.
- START was specified instead of RESTART for a file which already has records in it.
- STOP was requested while user processes are accessing the logfile.

Logging to DISC

When logging to DISC, for User Logging to continue when a full logfile is encountered, the :CHANGELOG command may be used. The :CHANGELOG command will open a new User Logging file with the same attributes as the previous file. To use the :CHANGELOG command, the User Logging *filename* specified in the :GETLOG command must end with the three digits "001". Each time a new logfile is constructed, by the CHANGELOG command, the value represented by the last three digits is incremented by 1 (for example, *filename*001—-> *filename*002). If the logfile name specified with the :GETLOG command does not end with the three digits "001", a warning will be given (to the user issuing the command) that the :CHANGELOG command will not be allowed for this file.

If the ;AUTO option is specified with the :GETLOG or :ALTLOG command for a disc file, where :CHANGELOG is allowed, the User Logging process will automatically perform a :CHANGELOG whenever the disc file becomes full. If the ;AUTO option is not specified, when the disc logfile becomes full the User Logging process will close the logfile and terminate, preventing any applications which are linked to it from writing any additional log records.

Logging to TAPE

When logging to tape, if an end-of-tape is encountered, the Operator will be requested to mount the next tape in the volume set and the logging will continue. The user may issue a :CHANGELOG command for logging to be switched to a new logfile (DISC or TAPE) before end-of-tape is reached.

Note The ;AUTO option is ignored if the logfile media is TAPE.

Linking Logfiles

When a :CHANGELOG command is issued, information about the new logfile and is written to the old logfile, and information about the old logfile is written as the first record in the new logfile. This allows the recovery programs to find the next logfile in the set.

Note Using :CHANGELOG is recommended over stopping the process, switching logfiles (by renaming files or using :ALTLOG to point to a new logfile), and starting the process. Using :CHANGELOG is easier and it creates these links for use in recovery.

It is not required that all logfiles in a set be on the same media. The first file could be on disc, the second on tape, the third on disc, etc. This can be controlled with the ;DEV= option of the :CHANGELOG command.

After a System Backup

Usually, when it is time to backup a database, the operator will stop the logging processes; the database and all associated logfiles will be stored off, and the old logfile(s) will be purged from disc and rebuilt. If :CHANGELOG was allowed, it is important to use the :ALTLOG command to reset the last three digits of the logfile name back to "001". Failure to do this will cause problems when recovering data.

Obtaining Status of Open User Logging Files

To display the status information of currently open User Logging files, use the :SHOWLOGSTATUS command:

```
:SHOWLOGSTATUS KATHY

LOGID  CHANGE AUTO USERS STATE    CUR REC  MAX REC  % USED

KATHY  YES    YES  1     INACTIVE  65       1023     6 %
```

Note INACTIVE is displayed when a process is waiting for information from the user processes.

If there are processes with the logfile open, and a :LOG command with the STOP option is issued, the operator will receive a warning at the System Console, and the logging process will continue.

5

User Logging in an Application

When the logging process is running, an application can begin writing log records, to the logfile, through the use of intrinsics.

Using the OPENLOG Intrinsic

The `OPENLOG` intrinsic provides access to the User Logging facility. When an application calls `OPENLOG` for a currently running logging process, the logging process builds the data structures which are necessary to allow the application to record data in the logfile, and a record is written to the logfile identifying the application. If the *logid* is not valid or the specified User Logging process is not running, an error will be returned.

The identification information in the `OPENLOG` record can be used, by the recovery program, to identify the records in the logfile, which belong to the process it is trying to recover. In order for the identification information to be useful for recovery, the application must retain some information of its own, particularly if the logfile will be accessed by more than one user process.

It is recommended that the application have a separate file to record such information as: user, group, and account which is accessing the logfile (this can be obtained with the `WHO` intrinsic); the *pin#* of the process accessing the logfile (this can be obtained with the `GETPROCID` intrinsic); the fully qualified file name of the current logfile and the ASCII code of the *logid* (these can be obtained with the `LOGINFO` intrinsic); the date and time that the logfile was opened by the application (call the `CALENDAR` and `CLOCK` intrinsics immediately following a successful call to `OPENLOG`). How this information will be used will be explained in Chapter 6, "Recovery".

Using the WRITELOG Intrinsic

After the logfile is opened, the data can be written to the logfile with the WRITELOG intrinsic. Data will be passed in the WRITELOG intrinsic call and User Logging will add 9 words of information to the beginning of each record. The 9 words of information describe which process recorded the data, when it was recorded, which User Logging intrinsic was used to record it, and a checksum for validity checking. The data supplied by the user should include information about the action being recorded, such as:

- What was the purpose of the transaction
- What operation was done
- Which file was modified
- Which record in the file was modified
- What did the data in the record look like after the action

Enough information must be included to allow the user to recreate the action from the log record if recovery is necessary.

Using the BEGINLOG and ENDLOG Intrinsics

To take full advantage of the data protection of User Logging, a logical transaction should be defined for the application. All actions that must be performed together for data to be in a consistent state should be included. In case the transaction is interrupted, the beginning and end of each transaction should be marked allowing fully and partially completed transactions to be identified.

The BEGINLOG intrinsic writes a special record to the logfile, marking the beginning of a logical transaction. When BEGINLOG is used, the User Logging memory buffer is flushed to ensure the record gets to the logfile. A BEGINLOG record with no data can be written to mark the beginning of a transaction, or data may be included (refer to the WRITELOG intrinsic). Including data allows the user to include information about the entire logical transaction, or to mark the beginning of the logical transaction and write the first log record with a single intrinsic call.

Note The WRITELOG intrinsic may be used to write logging records for each action within the transaction.

ENDLOG contains the same information as BEGINLOG except it marks the end of a transaction. ENDLOG may be called with or without data, and it flushes the User Logging memory buffer to ensure the record gets to the logfile.

BEGINLOG and ENDLOG pairs may be nested within other pairs and may be called any number of times. User Logging does not keep track of matching BEGINLOGs and ENDLOGs; it is the user's responsibility to ensure they are properly matched. If they are not matched, they will not be useful in recovery.

Using the CLOSELOG Intrinsic

When the application has completed accessing the logfile, the CLOSELOG intrinsic is called. The CLOSELOG intrinsic will post a record to indicate all transactions for the process have been completed.

OPENLOG and CLOSELOG may be called multiple times by the same user.

Using the MODE Parameter

Each User Logging intrinsic includes a *mode* parameter used to indicate whether the I/O is to be performed with WAIT or NOWAIT. If NOWAIT is specified and the User Logging process is busy with a long operation (for example, allocation of disc space or writing a block to tape, etc.) and cannot accept data, an error will be returned and the operation should be attempted again.

If WAIT is specified and the User Logging process is busy, the intrinsic will return to its caller only after the User Logging process becomes available and the I/O completes.

WRITELOG allows a third value for the *mode* parameter, WRITE-and-FLUSH, which causes the User Logging buffer to be flushed at the first opportunity.

Using the FLUSHLOG Intrinsic

To flush the buffer without writing information, use the FLUSHLOG intrinsic. Flushing the buffer adds additional overhead to the process, but ensures that the write to disc is complete before control is returned to the next executable instruction. This may be desirable for critical data or nonreproducible data.

Using the LOGINFO and LOGSTATUS Intrinsics

The LOGINFO and LOGSTATUS intrinsics may be used programmatically to obtain information about an open User Logging file. The information obtained is similar to that available through the SHOWLOGSTATUS command. These intrinsics are primarily used to obtain information about how full the logfile is, especially if AUTO is not in effect for the logfile.

6

Recovery

Each application which accesses User Logging should have a recovery program written by the user. The recovery program should read the formatted data in the logfile and apply it to a backup copy of the data structures belonging to the application. The recovery program can then be run in case of a system failure or program abort to recover the transactions that had fully completed before the problem occurred.

Recovery After System Failure

After a system failure, the operator should perform a system startup with system RECOVERY. If successful, User Logging has its own Warmstart recovery process which opens each logfile that was active at the time of the failure. It will write a crash record to the end of the data in the logfile, move the end-of-file pointer to the end of the last good block, and close the logfile. This will place the file in a "good" state so it can be used by the recovery program or by User Logging when the logging process is restarted. If system RECOVERY is not done, the logfile can still be read because, when User Logging opened the logfile for the process, it moved the end-of-file pointer to the file limit. All the records in the file can be read, even though those at the end of the file are not valid data.

To handle this case, the program should check the CKSUM word of each record. The CKSUM is calculated by XORing all of the 16-bit words in the record using a base of -1. When the recovery program encounters a record whose CKSUM is not valid, it will assume that it has come to the end of valid data in the file.

The logfile being read should be accessed through the MPE file system with `FOPEN, FREAD`, etc. There are no User Logging intrinsics which allow users to read logfiles.

If more than one process accessed the logfile, each record must be checked and a determination made to see if the record belongs to the process being recovered. To do this, check the file belonging to the process that contains the user, group, and account accessing the logfile, the *pin#* of the process, the fully qualified file name of the logfile, the *logid*, and the date and time the logfile was opened by the application.

Open the logfile, search the logfile for the *logid*, user, group, and account which match in the `OPENLOG` record (refer to Appendix B, "Record Formats"). Verify the match by using the *pin#*, and the date and time of the open. When the correct `OPENLOG` record has been found, obtain the *log#*. All records with that *log#* belong to the process.

Backup copies of the application data structures will be needed for the recovery process. The recovery program should look for BEGINLOG and ENDLOG records. When an ENDLOG record is found, the transaction has been completed, and all actions represented by the WRITELOG records within that transaction should be applied to the backup data structures. If a BEGINLOG is found without a matching ENDLOG, the transaction was not complete, its actions should not be reapplied.

CONTINUATION Records

Each BEGINLOG, WRITELOG, and ENDLOG record contains up to 119 words of user data. If more than 119 words were passed to the intrinsic, one or more CONTINUATION records will contain the remainder of the data. The LEN field will reflect the total number of words that were passed. For example, if a length of 140 words was passed to the WRITELOG intrinsic, the LEN field of the WRITELOG record will contain 140 words. There will be 119 words in the user data area of the WRITELOG record and 21 words in the CONTINUATION record. The LEN field of the CONTINUATION record will also reflect the total number of words (140). A positive LEN indicates 16-bit words, a negative LEN indicates 8-bit bytes.

CHANGELOG Records

If the recovery program encounters a CHANGELOG record with a code of 12, the record will contain the name and type of the next logfile in the set; continue recovery with the next logfile. The first record of the new logfile should be a CHANGELOG record with a code of 11. It will contain the name and type of the previous logfile, and may be used as a check when the new logfile is opened.

When Recovery is Complete

When the recovery program has completed, there are two options:

- The database and logfiles can be backed up, the logfile rebuilt, the name of the logfile changed to end with 001 (if CHANGELOG is allowed), and a :LOG command with the :START option issued (:LOG *logid* ,START) to start a new set of logfiles.
- If system recovery was done to the logfile, the :LOG *logid*, RESTART command may be issued to restart the User Logging process. This will continue logging to the same logfile(s).

The second option will not work if system startup with RECOVERY was not performed; if so, use the first option. The application may be restarted after the point of the last complete transaction.

Power Failure

User logging automatically recovers from power failures occurring on the I/O device where the logfile resides. If User Logging detects that a power failure has occurred, it will automatically read the logfile until the last good record is read. It will then rewrite the data, from its buffer, starting at that block.

The Operator may be prompted to reset the tape drive and place it back online. If the power failure occurs at the beginning of the current tape, the Operator will be requested to mount the previous tape and then remount the current tape. The system console will display messages that the User Logging Power Failure Recovery has completed.

A

Suggested User Logging Procedure

The suggested User Logging procedure includes the following:

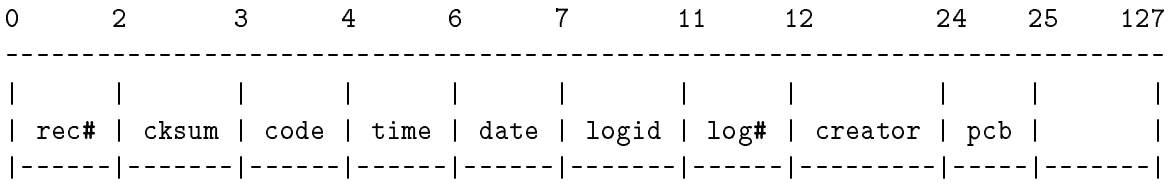
1. Use the `:GETLOG` command to create a *logid* for the application. If data security is required, use the `;PASS` option of the `:GETLOG` command. Specify the configured device class name (`DISC` or `TAPE`). If the *logid* specified in the `:GETLOG` command is associated with a disc file, build the file with the `;CODE=LOG` option of the `:BUILD` command and enough disc space to contain one day's output.
2. Design the application and data structures.
3. Determine what information will be required for recovery of the application's data structures.
4. Write the application and include a separate file which contains the identification information, and include the appropriate calls to User Logging intrinsics to record the data necessary for recovery.
5. Design and write the recovery program. The recovery program must recognize the User Logging file record formats and the application's data structures.
6. Have the logging process for your *logid* started (refer to the `:LOG` command in the *MPE/iX Commands Reference Manual* (32650-90003)).
7. Store a copy of the application's data structure to a backup medium in case recovery procedures are required.
8. Run the application.
9. If it is necessary to recover the data structures, `:RESTORE` the backup copies and run the recovery program. This recovers any changes that were made to the data structures since backup was done.

B

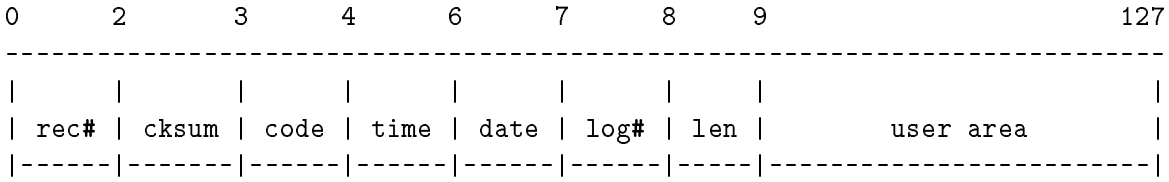
Record Formats

Record formats are required for direct access to the logging files by the user. The following logging record formats indicate where information resides during the logging process.

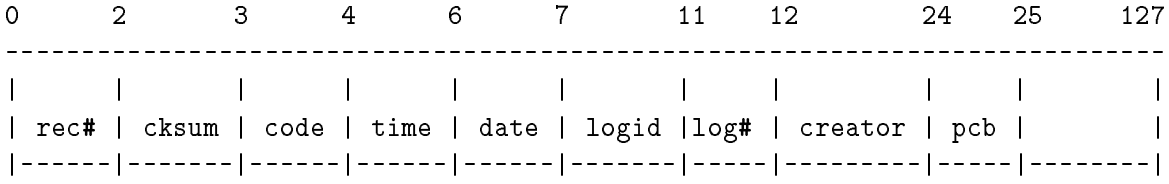
Logging Record Format:
record size = 128 words
user area = 119 words



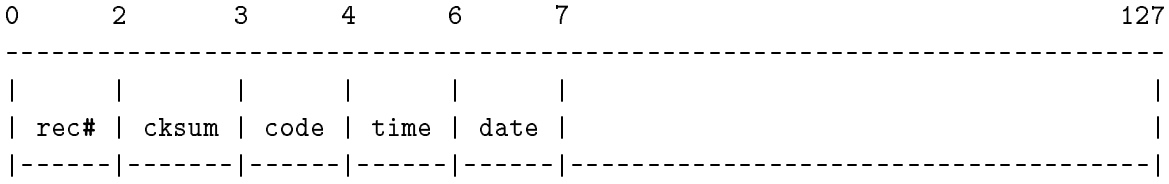
Log Record at OPENLOG



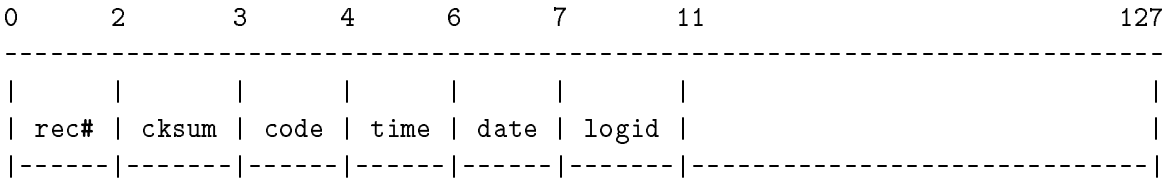
Log Record at WRITELOG



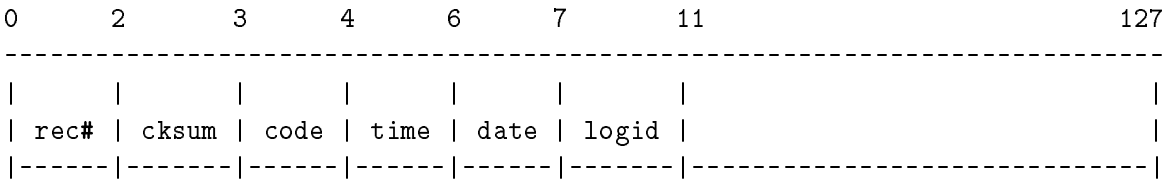
Log Record at CLOSELOG



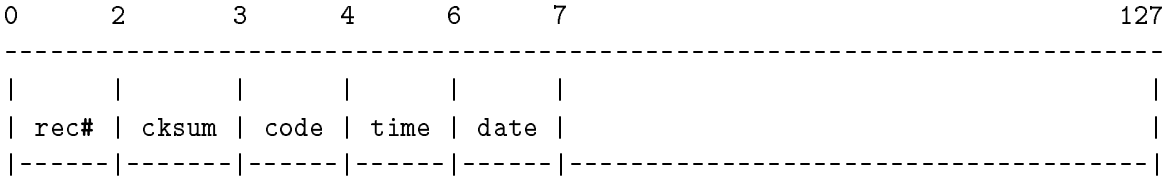
Crash Mark



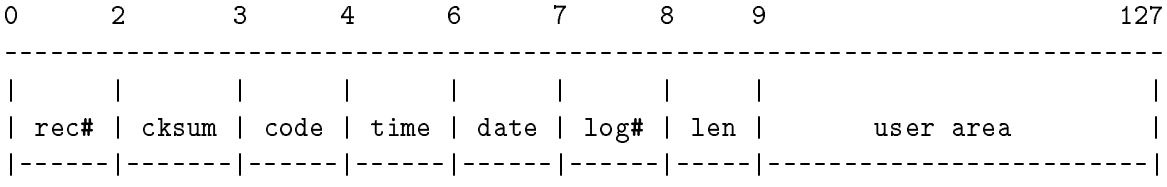
Header Record (Start/Restart)



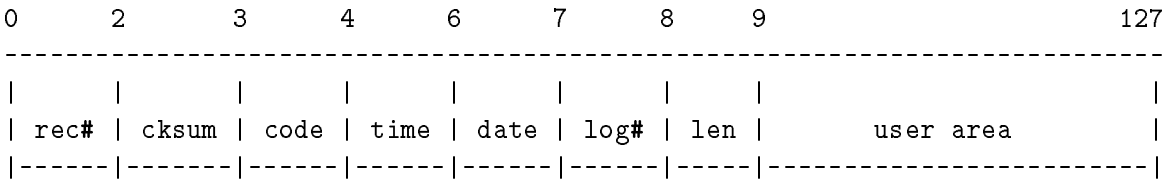
Trailer Record (Stop)



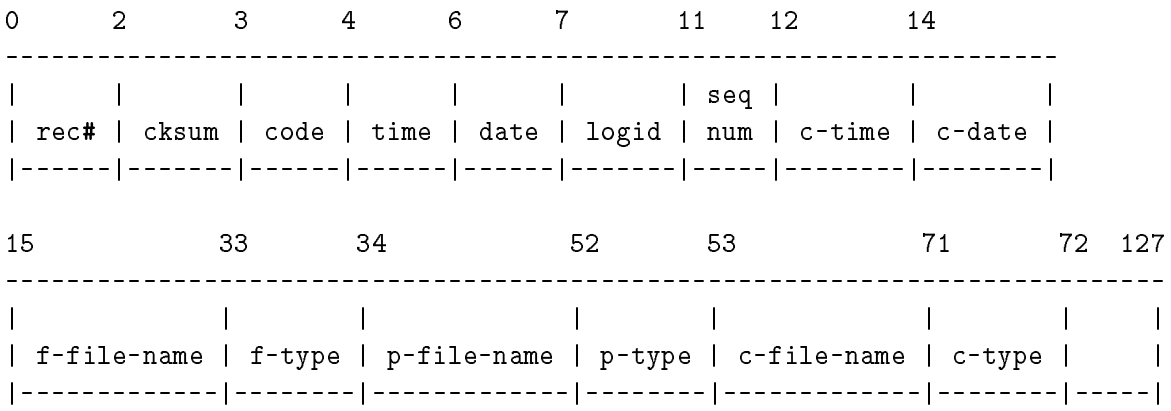
Null Record



Begin Transaction Marker



End Transaction Marker



Log Record at CHANGELOG

Table B-1 lists the code definitions of record formats, and Table B-2 lists the data fields of log records. (Notes for each follow the tables.)

Table B-1. Code Definition

Code=	#	Definition
(8:8)	1	Open log record
	2	User/subsystem record
	3	Close log record
	4	Header record
	5	Trailer record
	6	Restart record
	7	Continuation of user subsystem record
	9	Crash marker
	10	End transaction record
	11	Begin transaction record
	12	Change log record (resides in new file; points to old file)
	13	Change log record (resides in old file; points to new file)
	32	Null record
(0:8)		Subsystem code (can be specified with Privmode only)

Code Definition Notes:

The code in the second byte of Word 3 of each logging record identifies the type of record. For example, a "1" in the second half of the third word indicates an `OPENLOG` record.

Privileged users can define a subsystem code in the first half of the logging record code word bits (0:8). This code is passed in the *index* parameter of the `OPENLOG` intrinsic.

The null record is used as a filler.

If Code = 12, then *p-file-name* = previous file in the set and *p-file-name* = previous file type in the set.

If Code = 13, then *p-file-name* = next file in the set and *p-file-name* = next file type in the set.

Table B-2. Data Fields of Log Records

Field	Description
REC#	Double Integer
CKSUM	Integer
CODE	Integer
TIME	Double Integer (from `CLOCK` intrinsic)
DATE	Integer (from `CALENDAR` intrinsic)
LOGID	ASCII
LOG#	Integer
LEN	Integer
USERAREA	ASCII
CREATOR	ASCII (Name of user which opened the file)
PCB	Integer
SEQ NUM	Integer
C-DATE	Double Integer
C-TIME	Double
F-FILE-NAME	ASCII
P-FILE-NAME	ASCII
C-FILE-NAME	ASCII
F-TYPE	Integer
P-TYPE	Integer
C-TYPE	Integer

Data Fields of Log Records Notes:

The checksum (CKSUM) algorithm uses the Exclusive-Or (XOR) function against a base of negative one.

The length field (LEN) contains the length passed to `WRITELOG`, `BEGINLOG`, or `ENDLOG`. If a `CONTINUATION` record is part of the transaction, that record will also contain the data length. For example, if a length of 140 (words) is passed to the intrinsic, the LEN field will contain 140. Since the user area will only accommodate 119 words the remaining 21 words will be stored in the `CONTINUATION` record. The LEN field of the `CONTINUATION` record will indicate the total number of words in the transaction (140 in this example). A positive number indicates 16-bit words; a negative number refers to 8-bit bytes.

C

User Logging Error Codes

Table C-1 lists the User Logging error codes and their meanings. These error codes are returned in the *logstatus* parameter of each of the User Logging intrinsics.

Table C-1. User Logging Error Codes

Code	Meaning	Corrective Action
0	No error for this intrinsic call.	None.
1	User requested `NOWAIT` mode and the logging process is busy.	Retry the intrinsic call.
2	Parameter out of bounds in logging intrinsic. For example, the address of *data*, for `WRITELOG`, and the *length* given will go beyond the top of the stack.	Check all parameters to be sure they are correct and addressed properly.
3	A request to open or write to a logging process that is not running.	Use the `:SHOWLOGSTATUS` command to determine which User Logging processes are running. To start or restart a User Logging process, the system supervisor can use the `:LOG` command.
4	Incorrect *index* parameter passed to a User Logging intrinsic.	The *index* parameter must be a number that was returned by a successful call to the `OPENLOG` intrinsic. Do not change it. It will not be valid after a call to `CLOSELOG` with that *index* parameter.
5	Incorrect *mode* parameter passed to a user logging intrinsic.	For the `WRITELOG` intrinsic, valid *mode* parameters are 0, 1, and 2. For all other intrinsics, valid *mode* parameters are 0 and 1 only.
6	A request to open the logfile was denied and the User Logging process was suspended (the User Logging process is stopping or attempting recovery from an exception condition, for example, a power failure).	The `:SHOWLOGSTATUS` command or the `LOGINFO` intrinsic will give the current state of the User Logging process. If the process is stopped, it must be restarted by the system supervisor before the user can open the file.
7	Illegal capability. The user must have LG or OP capabilities to use the User Logging intrinsics.	See your account or system manager to obtain the proper capabilities.

Table C-1. User Logging Error Codes (Cont.)

Code	Meaning	Corrective Action
8	Incorrect password was passed to a User Logging intrinsic.	The creator of the *logid* can obtain the password by using the `;PASS` option of the `:LISTLOG` command.
9	An error occurred while writing to the log file. The File System error message will be printed on `$STDLIST`.	Obtain the File System error from the `$STDLIST` and take the appropriate action.
12	The system is out of disc space, the logging process cannot proceed.	Take the usual action for obtaining disc space, then restart the logging process.
13	`OPENLOG` failed. The maximum number of users that can access a User Logging process at one time has been reached. This number is configurable at system startup only.	Temporary solution: Wait for a user, who is accessing the logfile, to finish. Long term solution: At the next system startup, have the number of users that can access a User Logging process increased.
14	Invalid access to a log file (security violation). The user, group, or account calling the intrinsic does not match the user, group, or account for the log index passed.	Each user wanting to access a User Logging process, and its associated logfile, must have obtained access to it through a call to `OPENLOG`.
15	End-of-file on logfile (the logfile is full and `AUTO` was not on to automatically switch to a new logfile).	`:RENAME` the logfile and `:BUILD` a new one with the old name, type `:LOG` *logid*,`START` to restart the logging process. or `:BUILD` a new logfile with a new name, use the `:ALTLOG` command to point the *logid* to the new logfile, and use `:LOG` *logid*, `START` to restart the logging process.
16	The *logid* does not exist.	To see the *logid*s that exist, use the `:LISTLOG` command. To add a *logid*, use the `:GETLOG` command.
17	Parameter (either *itemnum* or *itemval*) missing for the `LOGINFO` intrinsic.	Check the `LOGINFO` parameters, the *itemnum* and *itemval* must occur in pairs.
18	Invalid item number passed to the `LOGINFO` *itemnum* parameter.	Refer to the *MPE/iX Intrinsics Reference Manual* (32650-90028) for a complete list of valid item numbers that can be returned to the *itemnum* parameter of the `LOGINFO` intrinsic.

Index

D

E

F

G

I

L

M

N

O

P

R

S

T

U

V

W

X