PERKIN-ELMER

OS/32
PATCH

Reference Manual

48-016 F00 R01

The Perkin-Elmer Corporation, Data Systems Group, 2 Crescent Place, Oceanport, New Jersey 07757

Printed in the United States of America

TABLE OF CONTENTS

CHAPTERS (Continued)

APPENDIXES

FIGURES

TABLES

PREFACE

This manual is a guide to using Perkin-Elmer OS/32 Patch. Patch allows the user to apply software changes to object or image code without reassembling the source module.

Users of Patch should be familiar with OS/32, the Perkin-Elmer 32-bit processor machine instruction formats, object and image module formats.

Chapter 1 describes the capabilities of OS/32 Patch. Chapter 2 discusses the procedure for building, loading and starting a Patch image load module. Chapter 3 provides a detailed description of all Patch commands. Chapters 4 and 5 describe the concepts of image and object patching, respectively. The four appendixes include a Patch command summary, Patch information and error message summary, and descriptions of the object and image code formats.

Revision 01 of this manual introduces three new commands: COMMAND, VERIFY and VARIABLE. This revision also introduces enhancements to the BIAS, EXAMINE, MODIFY, TSW and TRANSFER commands to support the use of variables. Changes have also been made to the RANGE and EXPAND commands to support the use of program variables. In addition, several new object loader control items have been added. The loading of overlay files created by TET in Patch (software number 03-196 R02) differs from Patch (software number 03-196 R00).

For information on the contents of all Perkin-Elmer 32-bit manuals, see the 32-Bit Systems User Documentation Summary.

CHAPTER 1
OS/32 PATCH

1.1 INTRODUCTION

Patch is a program development tool that allows the user to change object or image versions of programs without reassembling the source module. Patch provides a convenient means of applying software changes to object or image code.

The capabilities provided are:

- modification of object and image modules,
- addition of code or data,
- a history feature that records all changes made, optionally labeled with Patch control IDs, and
- manipulation of object libraries and compound overlay files.

Patch executes as a disk-based reentrant program. It can be run in interactive or batch mode. The following sections describe the Patch features.

1.2 IMAGE PATCHING

All image modules produced by OS/32 Link and TET can be patched, including tasks, operating system images, overlays, resident libraries, preinitialized task commons (TCOMs) and shared segments.

An image module consists of one or more loader information blocks (LIBs) followed by image code segments that may consist of impure, pure and overlay segments. If the image module is a task with tree-structured overlays, there will also be an overlay descriptor table (ODT). These segments contain the program as it appears in memory when it is loaded. An LIB contains information such as segment sizes, entry points for resident libraries, and task parameters and options set by Link or TET. LIBs are described in Appendix D.

Patch provides commands to change the LIB information and the data within segments. It also allows segments to be expanded to accommodate additional code or data.

1.3 OBJECT PATCHING

All object modules produced by the Common Assembly Language (CAL) assembler can be patched, including named and unnamed object modules and block data subprograms.

An object module consists of a sequence of loader items as described in Appendix C. Loader items can define the name and transfer address of a task, absolute (unrelocatable) data, address constants, common areas, common references, external references and other items.

Patch provides commands to examine the code generated by these loader items and to create additional loader items. The user is responsible for ensuring that additional loader items do not overlay chains of external references.

1.4 OBJECT LIBRARIES AND COMPOUND OVERLAY FILES

Patch enables the user to process files that contain more than one module. Files containing more than one object module are called object libraries. Patch is capable of modifying compound overlay files that consist of a root segment and at least one overlay area.

With single modules, Patch reads the original module from an input file and writes the patched version to an output file. With object libraries and compound overlay files, Patch allows the user to select modules to be patched and to decide whether or not to include intervening modules in the output file. For instance, it is possible to create a new object library with several modules replaced by patched versions. It is also possible to produce a file with only the patched modules.

NOTE

See Section 4.4 for an explanation of the difference between Patch (software number 03-196 R02) and Patch (software number 03-196 R00) in the loading of overlays created by TET.

1.5 HISTORY FEATURE

Patch maintains a record of all changes made during a session. In addition, a Patch control ID can be associated with each session of changes.

Particular Patch commands display the patched modules, number of revisions and associated Patch control IDs. For any patched module, the complete record of all changes made by Patch can be displayed.

The Patch control ID consists of a positive integer from zero to 32,767 and up to four alphanumeric characters. This Patch control ID can be issued by Perkin-Elmer with any Perkin-Elmer supplied patches or by the user according to installation requirements.

1.6 PATCH REQUIREMENTS

Patch requires these system components:

- Minimum hardware support required for OS/32
- Operating system of R06.2 or higher
- Disk device
- Print device
- Temporary file support

1.7 STATEMENT SYNTAX CONVENTIONS

These statement syntax conventions are used in all instruction formats. Examples of Patch commands are shown for each convention.

Examples:

Capital letters, parentheses and punctuation marks must be entered exactly as shown:

```
BIAS A000:P
```

Lower-case letters represent parameters or information provided by the user:

```
RANGE adr ,adr
```

Underlining indicates that only the underlined portion of the entry is required:

```
PAUSE
```

An ellipsis represents an indefinite number of parameters or a range of parameters:

```
MODIFY data [,...data]
```

Braces represent required parameters from which one must be chosen:

```
DISPLAY {D}
        {P}
```

Brackets represent an optional parameter that may be chosen:

```
HELP [command name]
```

Commas separate parameters and substitute missing positional parameters:

```
OBJECT [fd1] [,fd2] [,LIBRARY]
```

Commas inside brackets must be entered if the optional parameter is chosen:

```
IDNO [n] [,charstring]
```

Braces inside brackets represent optional parameters from which one may be chosen:

```
LOG [{ON }]
    [{OFF}]
```

Commas preceding braces inside brackets must be entered if one of the optional parameters is chosen.

```
          {name   }
          {*OBJECT}
GET       {*TASK  }  [,{NOCOPY}]
          {*OS    }   {COPY  }
```

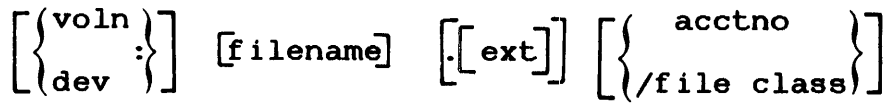
An equal sign separating the keyword from parameters must be entered to associate a parameter with the keyword.

SHARED NAME= sname

1.7.1 File Descriptors (fds)

All fds are entered in a standard format.

Format:



Parameters:

voln: is a 1- to 4-character alphanumeric string specifying the name of a disk volume. The first character must be alphabetic and the remaining alphanumeric. If the volume name is omitted, the default is:

- temp volume for temporary files,
- spool volume for spool files,
- user volume for all other files in a multi-terminal monitor (MTM) environment, or
- system volume for all other files in an operating system environment.

dev: is a 1- to 4-character alphanumeric string specifying a device name. The first character must be alphabetic and the remaining alphanumeric.

filename is a 1- to 8-character alphanumeric string specifying the name of a file. The first character must be alphabetic and the remaining alphanumeric. If a filename is specified with a device name, the filename is ignored.

.ext — is a 1- to 3-character alphanumeric string specifying the extension to a filename. If the period (.) is specified and the extension is omitted, the default is blanks. If the period and extension are omitted, a default extension appropriate to the particular Patch command in which the fd appears is appended to the filename.

acctno — is a decimal number ranging from 0 through 65,535 specifying the account number associated with the file. Account numbers 1 through 65,535 (excluding 255) are used by MTM. Account number 255 is reserved for the Authorized User Utility. Account 0 is used for system files and is the default for all operator commands.

file class — is a 1-character alphabetic string specifying the type of file class when files are used in an MTM environment. The file class types are:

- /P for a private file
- /G for a group file
- /S for a system file

An n specifies account number rather than class designator (P, G and S). If the file class is omitted, the default is P in an MTM environment. In an operating system environment, S is the only file class that can be specified.

CHAPTER 2
STARTING PATCH

2.1 BUILDING THE PATCH IMAGE LOAD MODULE

The following sequence of commands can be used to build Patch as a sharable segmented image load module from an object module named PATCH.OBJ:

```
LOAD LINK
START
ESTABLISH TASK
INCLUDE PATCH.OBJ
OPTION WORK=n,SEGMENTED
BUILD PATCH
END
```

The argument of the WORK parameter in the OPTION command can be calculated by using the following formula.

Formula:

$$n = w + e [+ X'50']$$

Where:

n	is a hexadecimal number specifying the sum of variables w and e that should be a minimum value of X'50' to provide space for the START parameters.
w	is a hexadecimal number specifying the number of bytes occupied by the largest program to be patched in main storage.
e	is a hexadecimal number specifying the total number of bytes all Patch EXPAND commands will add to the current task size.

The size of the workspace created during the building of the image load module can be overridden at load time.

2.2 LOADING PATCH

Patch must be built as an image load module before it can be loaded (see Section 2.1).

2.2.1 Loading Patch into an Operating System Environment

The following explains how to load Patch into an operating system environment.

Format:

LOAD taskid [,[fd],[segsize increment]]

Parameters:

taskid	specifies the name of the task after it is loaded into the foreground segment in main storage.
fd	is the file descriptor of the Patch image load module to be loaded into main storage. If this parameter is omitted, the default is taskid.TSK. If the Patch task was built as in the previous section, the task ID is PATCH.
segsize increment	is a decimal number, in kilobytes, specifying the additional area to be added to the task's impure segment. This value overrides the OPTION WORK value if it was specified when the task was built.

2.2.2 Loading Patch into a Multi-Terminal Monitor (MTM) Environment

The following explains how to load Patch into an MTM environment.

Format:

LOAD fd [,segsize increment]

Parameters:

fd	is the file descriptor of the Patch image load module to be loaded into main storage.
segsize increment	is a decimal number, in kilobytes, specifying the additional area to be added to the task's impure segment.

Functional Details:

The segsize increment field is optional and is used to provide additional storage for workspace. If the workspace is not large enough to contain the program to be patched, a message is displayed to inform the user that a temporary file is allocated for workspace. Use of workspace improves Patch response time. The necessary increment can be calculated using the following formula.

Formula:

$$\text{segsize increment} = \frac{nr*rs}{1024} + 1 + exp$$

Where:

nr	is a decimal number specifying the number of records in the largest program to be patched.
rs	is a decimal number of 126 for the record size of an object module and 256 for the record size of an image module.
exp	is a decimal number specifying the expansion size of the memory area to be used for additional data (see Section 3.9 for information on the EXPAND command).

2.3 STARTING PATCH

After Patch is loaded, the START command starts execution and causes the command, list and message devices to be assigned.

Format:

START [,COMMAND=fd_1] [,LIST=fd_2]

Parameters:

COMMAND= fd_1 specifies the input device from which commands are to be entered. If this parameter is omitted, the default is the console device (CON:). If the specified command input device is interactive, messages are sent to the command device. If the specified command input device is batch, messages are sent to the list device.

LIST= fd_2 specifies the output list device. If the LIST parameter is omitted and the command device is interactive, the list output is sent to the command device. If the LIST parameter is omitted and batch is the command device, the default is PR:.

If the list device is changed in interactive mode by using the LIST command, the device to which messages are sent remains unchanged. If the list device is changed in batch mode by using the LIST command, the device to which messages are sent changes to the new list device (see Section 3.15 for information on the LIST command).

Functional Details:

The list device is used by the DUMP, REVISION and TABLE commands to display various types of information (see Sections 3.6, 3.28 and 3.32, respectively).

CHAPTER 3
PATCH COMMANDS

3.1 INTRODUCTION

Commands are listed in alphabetical order in Table 3-1. The conventions used in describing the syntax are in Section 1.3.

TABLE 3-1 PATCH COMMAND AND DESCRIPTION SUMMARY

COMMAND	MEANING
BIAS	Sets the bias address for the EXAMINE and MODIFY commands.
BLOCK	Specifies the name of a common block.
COMMAND	Provides the ability to change the input command device.
DISPLAY	Displays the current Patch parameters to the device to which all messages are output.
DUMP	Displays the specified contents of an object or image module to the list device.
END	Terminates execution of Patch.
EXAMINE	Displays the contents of a specified location in a module.
EXPAND	Adds a Patch area to the end of a specified module.
GET	Specifies a module to be patched.
HELP	Displays the parameters for a specified command and briefly describes its usage.
IDNO	Labels patched modules with a Patch control ID.
IMAGE	Specifies image patching.

TABLE 3-1 PATCH COMMAND AND DESCRIPTION SUMMARY (Continued)

COMMAND	MEANING
LIB	Displays the contents of the loader information blocks (LIBs) for the currently selected image module being patched.
LIST	Changes the list device.
LOG	Enables or prevents copying of Patch commands to the list device.
MAXLU	Redefines the maximum number of logical units a task can use.
MODIFY	Changes the contents of specified locations in a module.
MXSPACE	Redefines the maximum amount of system space that a task can use.
NAME	Renames an object module.
NEWIDNO	Begins a new history record and assigns a Patch control ID for the new set of patches.
OBJECT	Specifies object patching.
OPTION	Redefines task options that were initially defined at Link or TET time.
OVERLAY	References tree-structured overlay modules while accessing the root segment.
PAUSE	Suspends Patch processing.
PRIORITY	Redefines initial and maximum priorities for a task.
RANGE	Computes the relative displacement between two addresses in a format suitable for use in an RX2 format instruction.
REVISION	Displays Patch history of a module from the current input file.

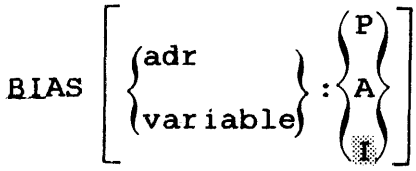
TABLE 3-1 PATCH COMMAND AND DESCRIPTION SUMMARY
(Continued)

COMMAND	MEANING
SAVE	Outputs a patched module to an output file.
SHARED	Changes access privileges and maximum or minimum size of a shared segment entry.
SEND STOP	Stops execution of the DUMP, EXAMINE, REVISION and TABLE commands.
TABLE	Displays a list of all of the modules in an input file.
TRANSFER	Redefines the transfer address of an object module.
TSW	Redefines the starting task status word (TSW) for a task.
VARIABLE	Allows assignment of an internal variable.
VERIFY	Provides the ability to verify the contents of a location.

3.2 BIAS COMMAND

This command sets the bias address for the specified segment, which becomes the current default segment descriptor.

Format:



Parameters:

adr	is a hexadecimal number from zero to FFFFFE aligned on a halfword boundary specifying an address in a pure, absolute or impure segment. The bias addresses for all segments are initially zero when Patch is started.
variable	is a 1- to 8-character string specifying the name of a previously defined internal Patch variable. See Section 3.35 for information on the VARIABLE command.
P	specifies the pure segment.
A	specifies the absolute segment.
I	specifies the impure segment. This segment is the default when Patch is first started.

Functional Details:

The BIAS command can be used at any time to establish bias addresses for each of the three segments. The address parameter in the EXAMINE and MODIFY commands is interpreted as an offset from the current bias address for the specified segment. If the address parameter of the BIAS command is omitted, the current bias for the specified segment is displayed. This segment descriptor becomes the default segment descriptor for subsequent EXAMINE and MODIFY commands.

If no parameters of the BIAS command are specified, the biases for all segments are displayed. The current default segment descriptor is preceded by an asterisk (*). The use of the default segment descriptor is explained in the EXAMINE and MODIFY command descriptions (Sections 3.8 and 3.18, respectively).

Examples:

```
BIAS 1234:I
```

The above example sets the impure bias to 1,234 and the current default segment descriptor to impure.

```
BIAS :P
```

This example displays the current pure bias and sets the current default segment descriptor to pure:

```
*PURE BIAS          0
```

```
BIAS
```

This example displays the current impure, pure and absolute biases and indicates the default segment descriptor with an asterisk:

```
*IMPURE BIAS     1234
 PURE BIAS          0
 ABSOLUTE BIAS      0
```

```
VARIABLE %NEWADD=1436
BIAS %NEWADD:I
```

The above example sets the value of the %NEWADD variable to 1,436 and establishes this as the impure bias. This form of the BIAS command also sets the current default segment descriptor to impure.

```
-----------
|   BLOCK   |
-----------
```

3.3 BLOCK COMMAND

This command specifies the name of a common block in an object block data subprogram.

Format:

BLOCK [name]

Parameter:

name	is a 1- to 8-character alphanumeric string specifying the name of the common block to be patched. If this parameter is omitted, the current selected common block name is displayed.

Functional Details:

In all subsequent EXAMINE and MODIFY commands, the address specified is interpreted as an offset from the beginning of the common block (see Sections 3.8 and 3.18).

Examples:

BLOCK ALPHA

The above example sets the current common block to ALPHA.

BLOCK

This example displays the current common block name.

3.4 COMMAND COMMAND

This command provides the ability to change the input command device.

Format:

COMMAND fd [,RETURN]

Parameters:

fd	is the file descriptor of the file or device from which Patch will accept commands.
RETURN	specifies that the user wishes to return to the original command device upon completion of the commands in the new device.

Functional Details:

Upon execution of this command, the file descriptor (fd) is checked for syntax. If the fd is correct, it is assigned as the command input device. Commands are read from this device until an end of data indicator (/* or ./) is encountered. At this time, the current command input device is closed and, if RETURN was specified, Patch reverts back to reading the original command file. COMMAND entered without any parameters will display the current command input device and whether or not RETURN is in effect. If this command is issued from a secondary command file while RETURN is in effect, the following message will be output.

'NESTING OF COMMAND FUNCTION ILLEGAL'

If an end of data indicator is encountered and RETURN is not in effect, it is ignored and the following message is generated.

'NO RETURN IN EFFECT'

NOTE

The END command overrides the RETURN option.

Example:

```
COMMAND MAG1:,RETURN
```

This example specifies that MAG1: becomes the input command device. Commands are read from the tape until an end of data indicator is encountered; Patch will then revert back to the previous command file.

3.5 DISPLAY COMMAND

This command displays information about current devices being used and various user-selected parameters. The information is displayed on the output device to which messages are sent. This information is divided into two groups:

- Device information
- Patch information

Format:

DISPLAY $\left\{\begin{matrix}D\\P\end{matrix}\right\}$

Parameters:

D — specifies that device information is to be displayed. Device information consists of the current:

- input and output files as set by the last IMAGE or OBJECT command,
- list device, and
- log mode status (see Section 3.16).

P — specifies that Patch information is to be displayed. Patch information consists of the current:

- common block name as set by the last BLOCK command,
- biases and default segment descriptor as set by the BIAS command,

- locations, segment IDs and sizes of Patch areas created by the EXPAND command, and
- Patch mode.

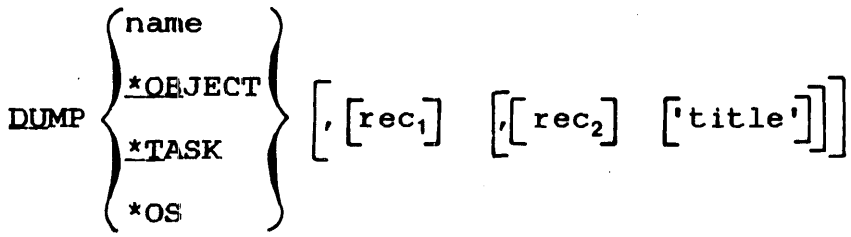
Functional Details:

See Sections 3.13 and 3.22 for descriptions of the IMAGE and OBJECT commands.

3.6 DUMP COMMAND

This command displays the specified contents of an object or image module to the list device.

Format:



Parameters:

name	is a 1- to 8-character alphanumeric string specifying the name of the module from the input file whose contents are to be dumped. If more than eight characters are entered for a resident library or task common module, an error message is displayed.
*OBJECT	specifies a named or unnamed object module in the input file. This module must be the only module within the input file.
*TASK	specifies a task image module that is a root segment of a compound overlay file produced by Link. This module must be the only module within the input file.
*OS	specifies an operating system image module.
rec	specifies a range of records to be displayed starting with the first specified record and ending with the last specified record. The defaults are the first and last record of the dump.
'title'	is a 1- to 50-character alphanumeric string to be output as a heading preceding the dump. If this parameter is omitted, a blank heading is the default.

Functional Details:

For object modules, DUMP produces history information and a listing of loader items. See Section 3.6.1 for the format for an object dump.

The input file from which a module is to be dumped must be first assigned by the IMAGE or OBJECT command. If the user fails to do so, an error message will be displayed.

If the module selected to be dumped to is currently being patched, the patched version of the program is displayed, along with the previous history information. The current position of an object library or a compound overlay file is not affected.

See Section 3.6.1 for an example of an object dump and Section 3.6.2 for an example of an image dump.

Examples:

```
DUMP *TASK,,'SAMPLE TASK DUMP'
```

The above example dumps all records of the task on the input file. Label the dump with the title given.

```
DUMP ALPHA,1,2
```

This example dumps the first two records of the module ALPHA. Use a blank heading.

3.6.1 Object Dump

An object dump contains the following:

- Data
- History records (if applicable)

Example:

```
      SEG    HEXADECIMAL          ASCII          LOADER        LOADER
ADR   DES    HALFWORD(S)           REP           ITEM NO       ITEM DES


000016:P   C9E0 012C 4280 8042 * ...,B..B *    (30)    34 BYTE(S) ABS DATA
00001E:P   C9E0 0190 4280 807C * ....B... *
000026:P   C5E0 01F4 4280 8304 * ....B... *
00002E:P   4300 8456 D310 4000 * C..V..@. *
000036:P   0000                * ..        *
                                            (13)    DEF START OF CHAIN (REF)
                               000034:P     (05)    PUR PROG ADR
                               LUS    000006(0B)    COMMON REF
                                            (19)    PERFORM FULLWORD CHAIN
000038:P   C9E0 0066 2134 D310 * ....14.. *    (25)     12 BYTE(S) ABS DATA
000040:P   4000 0000           * @...     *
                                            (13)    DEF START OF CHAIN (REF)
                               000040:P     (05)    PUR PROG ADR
                               LUS    000002(0B)    COMMON REF
                                            (19)    PERFORM FULLWORD CHAIN
000044:P   D210 4000 0B6D I                 (11)    3 BYTE ABS & 3 BYT IMP REL
00004A:P   081E CB10 0065 1112 * ........ *    (2D)     28 BYTE(S) ABS DATA
000052:P   58E1 8522 58D1 8522 * X.."X.." *
00005A:P   27D1 41F0 833E D310 * '.A..>.. *
000062:P   4000 0000           * @...     *
                                            (00)    END OF RECORD
```

Fields:

ADR	is the hexadecimal address of the first byte displayed on each line.
SEG DES	is the segment descriptor of I, P or A.
HEX HALFWORD	specifies one to four halfwords of hexadecimal data.
ASCII REP	is ASCII representation of absolute data items. Also included in this area are program entry points, external references and common references.
LOADER ITEM NO	is the loader control item.
LOADER ITEM DES	is the description of the loader control item.

3.6.2 Image Dump

For tasks, operating system and shared segments, the image dump contains:

- LIBs
- History records (if applicable)
- Data

Example:

```
    SEG                                                          ASCII
ADR DES          HEXADECIMAL HALFWORD(S)                      REP


2400:P   000F 1CF2 000F 1F2A 000F 1CF2 000F 1F42 * .......*.......B *
2410:P   000F 1CF2 000F 1CF2 000F 1CF2 000F 1CF2 * ................ *
2420:P   000F 1CF2 000F 1CF2 000F 1F6A 000F 1F74 * ................ *
2430:P   000F 1F74 000F 1F74 000F 1F7E 000F 1FA2 * ................ *
2440:P   000F 1FAA 000F 1F74 000F 1F74 494C 4C45 * .............ILLE *
2450:P   4741 4C20 4655 4E43 5449 4F4E 2020 2020 * GAL FUNCTION     *
```

Fields:

ADR	is the hexadecimal address of the first byte displayed on each line.
SEG DES	is the segment descriptor for I and P.
HEX HALFWORD	specifies one to eight hexadecimal halfwords.
ASCII REP	is ASCII representation of hexadecimal data.

3.7 END COMMAND

This command terminates execution of Patch.

Format:

```
END
```

Functional Details:

In interactive mode, a warning message is issued and a prompt is returned in expectation of another command under the following conditions:

- An END command has been entered without a corresponding SAVE command and the module was changed.
- The output file is not a null device and the module is not being patched in place.

If a second END command is then entered, Patch terminates execution.

Example:

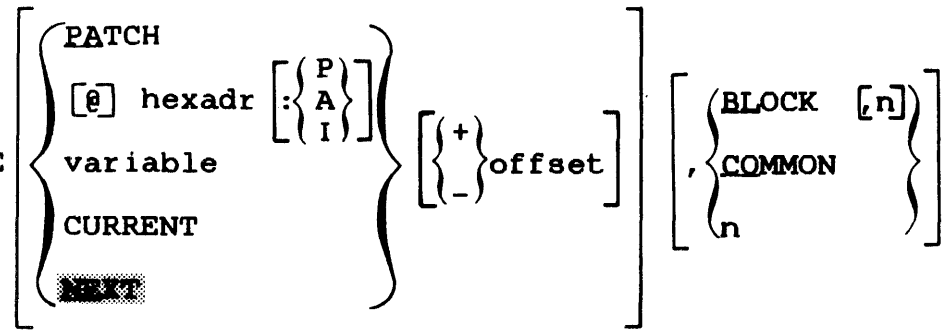
```
>END
SAVE CURRENT PROGRAM
>END
*01:01:01 PATCH:END OF TASK 0
```

In the above example Patch saves modified module.

3.8 EXAMINE COMMAND

This command is used to display the contents of specified locations in a module.

Format:



Parameters:

PATCH	specifies that the address of the last Patch area is to be referenced.
@	signifies that the bias is not to be added to the address.
hexadr	is a hexadecimal number from one to six digits aligned on a halfword boundary.
P	specifies a pure segment.
A	specifies an absolute segment.
I	specifies an impure segment.
variable	is a 1- to 8-character string specifying the name of a previously defined internal Patch variable. See Section 3.35 for information on the VARIABLE command.
CURRENT	specifies that the last address given in an EXAMINE or MODIFY command is to be referenced.
NEXT	specifies that the halfword following the last referenced halfword (via the EXAMINE or MODIFY command) is to be referenced.
+	specifies an arithmetic addition.

specifies an arithmetic subtraction.

offset — specifies an optional hexadecimal digit. If no address is specified, the address defaults to NEXT.

BLOCK — specifies that values within a block data subprogram are to be displayed.

COMMON — specifies that a COMMON reference is to be displayed.

n — is a decimal number of halfwords to be displayed. If this parameter is omitted, 1 is the default.

3.8.1 Image Mode

The EXAMINE command displays a number of halfwords, specified by the parameter n, starting at the location specified by the address parameter. The parameter n can be any positive decimal number and a value of 1 is assumed if n is not specified. This is the only form of the EXAMINE command used in image mode. The format of the response is:

```
 --------------------
|          | hex     |
| adr:seg  | halfword |
 --------------------
```

Fields:

adr — is the location of the starting address.

:seg — specifies the segment descriptor I, P or A.

hex halfword — represents from one to eight hexadecimal halfword values per line.

The impure segment descriptor is used when referencing task commons and overlays as well as the impure segments of task and operating system images.

The pure segment descriptor is used when referencing resident libraries, sharable segments and the pure segments of task and operating system images.

The absolute segment descriptor is not applicable in image mode.

3.8.2 Object Mode

The EXAMINE command is used in object mode in three different ways:

- to display halfword values in a module,
- to display common references, and
- to display halfword values in a common block within a block data subprogram.

Parts of an object module that have not been defined, such as uninitialized data arrays, are displayed as above with a U in the halfword value fields.

Examples:

```
EXAMINE 5A,40
```

The above example displays 40 halfwords beginning at address 5A plus the current bias.

```
EXA 5A:P,COM
```

This example displays the name and displacement of the common address reference in the instruction that starts at 5A plus the current pure bias.

```
EXA 5A,BLOCK,40
```

This example displays the 40 halfwords beginning at address 5A plus the current impure bias within the current common block.

```
VARIABLE %BEGADD=A10
EXAMINE %BEGADD:P,20
```

This example sets the value of the variable %BEGADD to A10, and then displays 20 halfwords beginning at address A10 plus the current bias.

3.8.3 Common Blocks

The object code for an instruction with a target address within a common block consists of two successive loader items. The first loader item contains the instruction with a target address of zero. This information can be displayed using the EXAMINE command. The second loader item contains the name of the common block referenced by the instruction and the displacement within that block of the target address. This information can be displayed using the EXAMINE command with the keyword COMMON as the second parameter. The address, in this case, specifies the beginning of the instruction that references the target address within a common block. The format of the display is:

```
 -------------------------
|          | common |       |
| adr:seg  | block  | displ |
 -------------------------
```

Fields:

adr	is the address of the first byte of the instruction referencing a common address.
:seg	specifies the segment descriptor P, A or I.
common block	is the name of the referenced common block.
displ	is a 1- to 6-digit hexadecimal number that specifies the displacement within the common block of the instruction target address.

3.8.4 Block Data Subprograms

When working with a block data subprogram, the BLOCK form of the EXAMINE command is used. However, a BLOCK command must be issued to select the current common block first. The address of the BLOCK command then specifies an offset within the current common block.

No segment descriptor should be used with an address, but the current impure bias is added to the address to obtain the effective offset within the current common block.

3.9 EXPAND COMMAND

This command allows data to be added to the end of the specified segment by creating a Patch area.

Format:

$$\text{EXPAND}\left[\left\{\begin{matrix}\text{P}\\\text{A}\\\text{I}\end{matrix}\right\},\text{n}\right],[\text{variable}]$$

Parameters:

P	specifies the pure segment.
A	specifies the absolute segment. This segment descriptor cannot be specified in image mode.
I	specifies the impure segment.
n	is a decimal number specifying the even number of bytes in the Patch area. For image modules, this number can be from 2 to 256. For object modules, this number can be from 2 to 100. If n is odd, it is rounded to the next highest even number.
variable	is a 1- to 8-character string specifying the name of a previously defined internal Patch variable. See Section 3.35 for information on the VARIABLE command.

Functional Details:

For impure segments of a task or an operating system image, the Patch area begins at the current UTOP. The UTOP and the TET expand/get area or the Link work area are moved accordingly. For image segments other than impure, the required number of bytes for all expands in a session are appended at the end of the segment. If expanding an overlay results in the need for a larger overlay area, the root segment should be expanded by the required amount. Patch areas for the root itself can then be created by the use of additional EXPAND commands. In image mode, this Patch area is initially all zero. In object mode, it is initially undefined.

A pure or impure segment being expanded must initially exist. However, in object mode, absolute segments can be created or expanded independent of prior existence. Block data subprograms cannot be expanded.

In tree-structured overlays created by Link, the overlays cannot be expanded. The root can be expanded and used as a Patch area for the overlays.

Multiple Patch areas can be generated.

Patch returns the starting address (with the segment descriptor) of the area and its size each time a segment is expanded. If no parameters are specified, Patch returns the starting address (with the segment descriptor) and size of each Patch area created.

Examples:

```
>EXPAND I,256
 PATCH AREA 318:I 256
```

The above example creates a Patch area of 256 bytes at the end of the impure segment.

```
>EXP P,100
 PATCH AREA 162A:P 100
```

This example creates a Patch area of 100 bytes at the end of the pure segment.

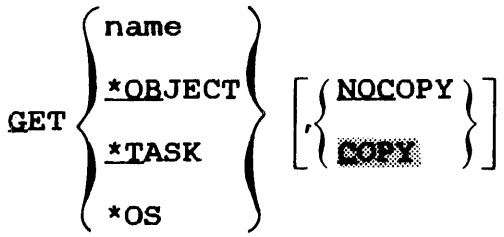
```
>VARIABLE %NEW
>EXPAND I,100,%NEW
```

This example establishes a variable (%NEW) and initializes it to zero. The EXPAND command then returns the Patch address and assigns this value to variable %NEW.

3.10 GET COMMAND

The GET command is issued for each module to be patched. Modules must be selected for patching in the same order that they appear in the compound file or object library.

Format:



Parameters:

name — is the name of an object module in a file by itself or in a library. It can be the name of:

- a resident library load module,
- a TCOM load module,
- a shared segment,
- an overlay load module in a file by itself or in a compound overlay file, or
- a tree-structured overlay.

*OBJECT — specifies a named or unnamed object module in a file by itself.

*TASK — specifies a task image module in a file by itself or the root segment in a compound overlay file.

*OS — specifies an operating system image load module.

NOCOPY specifies that any modules preceding the module specified by the GET command are not to be copied from the input file to the output file.

COPY specifies that any modules preceding the named module from the input file are to be copied to the output file. This option is effective only for object library and compound overlay files.

Functional Details:

The mode and input file must be set by the IMAGE or OBJECT command. If the user fails to do this, the following message is displayed:

```
NO FILES ASSIGNED
```

After setting the mode and the input file, the module to be patched can be selected by the GET command.

The first parameter specifies the name of the module to be patched. For unnamed object modules, task and operating system images, the special names *OBJECT, *TASK and *OS are used, respectively. When specifying the name of a resident library, TCOM or shared segment module, only the first eight characters of the name should be entered. Entering more than eight characters causes an error message to be output.

If the input file is an object library or a compound overlay file created by Link, the GET command selects the modules to be patched and also determines whether or not the other modules in the input file are to be copied to the output file. If NOCOPY is specified, any intervening modules are skipped. If COPY or no parameter is specified, all intervening modules are copied to the output file. Intervening modules are those modules between the module specified in the current GET command, and either the beginning of the input file or the module selected in the previous GET command.

Tree-structured overlays created by Link are part of the root segment and do not have their own LIB. A tree-structured overlay may be selectively accessed by name using the GET command. Only one overlay can be referenced this way. When the overlay is saved, all of the overlays and the root are saved. Therefore, multiple GETs of overlays will produce multiple images after the save. Since the tree-structured overlays are contained in the root segment, they are not a compound overlay file; therefore, the COPY parameter is not needed.

Both the root segment and tree-structured overlays can be referenced simultaneously by using GET *TASK and the OVERLAY command.

In interactive mode, if the GET command is issued before saving a currently modified module, a message reminding the user to save the currently modified module is output. If the user reissues the GET command, normal execution of GET proceeds. No warning message is issued if the output file is NULL or the INPLACE option is in effect (see Section 3.13). In batch mode, the message indicating that the previous module was not saved is issued, and Patch terminates.

Examples:

```
GET *OBJECT
```

The above example selects the unnamed object module in the input file to be patched.

```
GET ALPHA,COPY
```

This example selects ALPHA as the next module to be patched and copies the preceding modules to the output file.

3.11 HELP COMMAND

This command displays all the available Patch commands and gives a brief description of the command entered.

Format:

HELP [command name]

Parameter:

command name	is the name of a Patch command. The command can be entered in full or abbreviated form. A brief description of the use of the specified command is displayed. If this parameter is omitted, the list of Patch commands is displayed.

Functional Details:

The HELP information is contained in a file named PATCH.HLP provided with the OS/32 system. When a HELP command is entered, Patch attempts to find and assign this file to the:

- same volume and account number from which PATCH.HLP was loaded, or
- system volume and system account.

If both assign attempts fail, the following message is output:

ASSIGN HELP FILE TO LU 8

The user can then pause Patch and assign logical unit 8 (lu8), then continue the task and reenter the HELP command. The file can be preassigned by the user prior to the execution of Patch, and lu8 will not be closed.

Examples:

HELP IMAGE

The above example displays the parameters and a brief description of the IMAGE command.

H EXA

This example displays the parameters and a brief description of the EXAMINE command.

HELP

This example displays the following list of Patch commands:

```
BI(AS)          LIB             PR(IORITY)
BL(OCK)         LIS(T)          RA(NGE)
CO(MMAND)       LO(G)           RE(VISION)
DI(SPLAY)       MA(XLU)         SA(VE)
DU(MP)          MO(DIFY)        SEN(D) STOP
END             MX(SPACE)       SH(ARED)
EXA(MINE)       NA(ME)          TA(BLE)
EX(PAND)        NEW(IDNO)       TR(ANSFER)
G(ET)           OB(JECT)        TS(W)
H(ELP)          OP(TION)        VAR(IABLE)
ID(NO)          OV(ERLAY)       VE(RIFY)
IM(AGE)         PA(USE)
```

The letters not enclosed in parentheses indicate the command abbreviation.

3.12 IDNO COMMAND

This command labels patched modules with a Patch control ID, which consists of a control number and string.

Format:

IDNO [n] [,charstring]

Parameters:

n	is a decimal number from 0 to 32,767 specifying the control number.
charstring	is a 1- to 4-character alphanumeric string specifying the control string.

Functional Details:

If IDNO is entered without parameters, the current Patch control ID is displayed.

Special characters, other than carriage return (CR), can also be used. If less than four characters are entered, the string is padded with blanks. If no string is given and n is entered, charstring is set to all blanks.

The Patch control IDs associated with a patched module are also displayed by TABLE, REVISION and DUMP commands.

More than one Patch control ID can be entered for a module, but only the last one entered is effective.

The NEWIDNO command can be used to specify multiple Patch control IDs for changes made in a patching session (see Section 3.21).

Examples:

IDNO 12345,ABCD

The above example labels the current series of changes with Patch control ID 12345-ABCD.

IDNO 8,A/&

This example labels the current series of changes with Patch control ID 8-A/&.

IDNO ,ABCD

This example labels the current series of changes with Patch control ID 0-ABCD.

IDNO

This example displays current Patch control ID.

3.13 IMAGE COMMAND

This command specifies that the user is going to perform image patching.

Format:

IMAGE [fd_1,[fd_2] [,COMPOUND] [,NOHISTORY] [,INPLACE]]

Parameters:

fd_1	is the input file descriptor of the module to be patched. The input file must be a disk file with a record size of 256 bytes.
fd_2	is the output file descriptor to receive the patched module. If this parameter and the INPLACE parameter are omitted, the NULL: device is the default. If this parameter is omitted and INPLACE is specified, the default is the input file.
COMPOUND	specifies that the input file is a compound overlay file generated by TET.
NOHISTORY	specifies that no history records are to be maintained for image modules.
INPLACE	specifies that the changes are to be made directly to the input file. NOHISTORY is in effect.

Functional Details:

The input file is specified by fd_1 and contains the module to be patched. The patched version of the module is output to the file specified by fd_2. For both fds, the default extension is .TSK.

If INPLACE is specified, then the patches are made directly into the input file (fd_1). The output file (fd_2) should not be specified. If it is, it must match the input filename. No history records are maintained if INPLACE is specified and the EXPAND command cannot be used. The SAVE command is ignored while patching images in place. If the SAVE command is issued, "lu2 unassigned" is displayed.

If no history records are being maintained (NOHISTORY and INPLACE), then the task creation ID in the LIB is set to Patch and the time and date are changed to the current time and date.

The output file is created to be the same type and record size as the input file. If the input file is a contiguous file, Patch allocates an indexed file with filename fd_2 for output processing. When the user indicates that all processing of the input file is completed (by using the TERMINATE option in the SAVE command), the indexed output file is copied to a contiguous file with filename fd_2. If this copy operation fails (perhaps because a large enough contiguous file could not be allocated), a message is output and the indexed file is saved with filename fd_2.

The IMAGE command can be issued as many times as desired during execution of Patch.

If no parameters are present in the command line, then the current input and output assignments are displayed. If COMPOUND, NOHISTORY and/or INPLACE are specified, then those parameters are also displayed.

Examples:

```
->IMAGE PATCHOV1.TSK,,C
->IMAGE
- IMAGE INPUT ON M300:PATCHOV1.TSK/P
- IMAGE OUTPUT ON NULL:
- COMPOUND OVERLAY FILE


->IMAGE PATCHOV1.TSK,PATCHOV2.TSK
->IMAGE
- IMAGE INPUT ON M300:PATCHOV1.TSK/P
- IMAGE OUTPUT ON M300:PATCHOV2.TSK/P


->IMAGE PATCHSRC.TSK,,INPLACE
- NO HISTORY RECORDS MAINTAINED
->IMAGE
- INPUT ON M300:PATCHSRC.TSK/P
- OUTPUT ON M300:PATCHSRC.TSK/P
- NOHISTORY
- INPLACE
```

3.14 LIB COMMAND

This command displays the information contained in the LIBs of the image module currently selected for patching.

Format:

LIB [{n / 1}]

Parameters:

n	is a decimal number specifying the LIB to be displayed. If this parameter is omitted, 1 is the default.

Functional Details:

Only the information that is defined for the type of image module being patched is displayed. Some modules, such as resident libraries with many entry points, can have more than one LIB.

Although Patch changes the number of LIB records by including history records, these history records cannot be displayed with the LIB command. The REVISION command must be used.

Examples:

```
LIB
```

The above example displays the first LIB.

```
LIB 2
```

This example displays the second LIB.

```
-----------
|   LIST    |
-----------
```

3.15 LIST COMMAND

This command allows the user to change the list device. See Section 2.3 for a definition of the list device.

Format:

LIST [fd]

Parameter:

fd	is the file descriptor of the new list device. If this parameter is omitted, the current list device is displayed.

Functional Details:

If Patch is in batch mode, the message device is also changed (see Section 2.3).

Examples:

LIST PR:

The above example changes the current list device to PR:.

LIST

This example displays the current list device.

3.16 LOG COMMAND

This command is used either to copy or suppress copying of Patch commands to the list device.

Format:

LOG [{ON / OFF}]

Parameters:

ON	enables logging to the list device.
OFF	disables logging to the list device.

Functional Details:

Initially, the log feature is ON in batch mode and OFF in interactive mode. If the LOG command is given with no parameter, the current status of the log feature is displayed.

Examples:

```
LOG OFF
```

The above example turns the log feature OFF.

```
LOG ON
```

This example turns the log feature ON.

```
LOG
```

This example displays the current status of the log feature.

3.17 MAXLU COMMAND

This command allows the user to redefine the maximum number of logical units a task can use.

Format:

MAXLU [n]

Parameter:

n	is a decimal number from 1 to 254 specifying the new maximum number of logical units a task can use.

Functional Details:

Initially, this value is defined during link-edit time. The parameter gives the new value. If no parameter is given, the current value is displayed. This information is also displayed by the LIB command. This command is valid only in image mode while patching a task.

Examples:

MAXLU 20

The above example changes the maximum number of logical units for this task to 20.

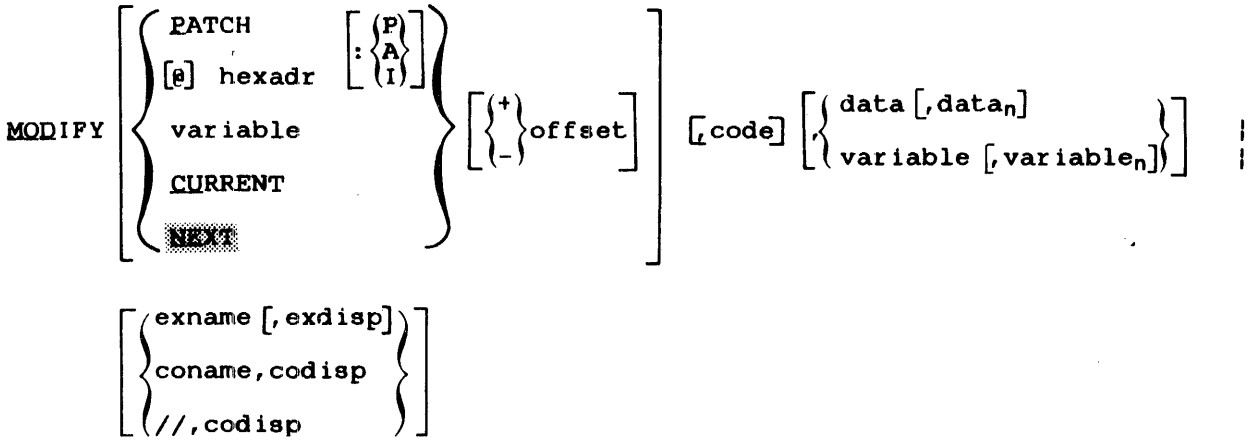
MAXLU

This example displays the current maximum number of logical units.

3.18 MODIFY COMMAND

This command enables the user to change the contents of specified locations in a module.

Format:



Parameters:

PATCH	specifies that the address of the last Patch area is to be referenced.
P	specifies a pure segment.
A	specifies an absolute segment.
I	specifies an impure segment.
@	signifies that the bias is not to be added to the address.
hexadr	is a hexadecimal number from one to six digits aligned on a halfword boundary.
variable	is a 1- to 8-character string specifying the name of a previously defined internal Patch variable. See Section 3.35 for information on the VARIABLE command.

CURRENT	specifies that the last address given in an EXAMINE or MODIFY command is to be referenced.
NEXT	specifies that the halfword following the last referenced halfword (via the EXAMINE or MODIFY command) is to be referenced.
+	specifies an arithmetic addition.
-	specifies an arithmetic subtraction.
offset	specifies an optional hexadecimal digit. If no address is specified, the address defaults to NEXT.
code	specifies the type of MODIFY operation. If this parameter is omitted, the default is ABS. See Table 3-2 for available codes.
data	specifies hexadecimal halfword values that modify the contents starting at the location specified as the address (see Table 3-2).
exname	is a 1- to 8-character alphanumeric string specifying the name of an external reference.
exdisp	is a 1- to 8-digit hexadecimal number specifying an offset from the address of an external reference.
coname	is a 1- to 8-character alphanumeric string specifying the name of a common block.
codisp	is a 1- to 6-digit hexadecimal number specifying a displacement within a common block.
//	is a special symbol specifying a blank common block.

Functional Details:

In all MODIFY commands the address specifies the location where changes are to be made. This address may have a segment descriptor or may refer to the default segment.

The starting address of the area to be changed is determined by the specified address plus the bias for the segment involved as set by the BIAS command.

The code specified determines how the remaining parameters are to be interpreted. It is the user's responsibility to avoid destroying chains of external references by attempting to overwrite an external reference.

In image mode, user-specified data are halfword hexadecimal values that are used to overwrite the data starting at the location specified by the address.

The I segment descriptor is used when referencing task commons and overlays as well as the impure segments of task and operating system images. The P segment descriptor is used when referencing resident libraries and the pure segments of task and operating system images. The A segment descriptor is not applicable in image mode.

In object mode, a MODIFY command results in additional loader items being appended to the object module. The particular type of loader item added is determined by the code parameter. Table 3-2 describes the codes and the additional parameters that can be specified. Any of the codes in Table 3-2 can be used in object mode.

TABLE 3-2 CODE TABLE

CODE	NUMBER AND TYPE OF ARGUMENTS	OBJECT CODE DESCRIPTION	NOTES
I	1 halfword value	2-byte impure relocation program address	Used for halfword address.
I	2 halfword values	4-byte impure relocation program address	Used for fullword address.
I	3 halfword values	3-byte absolute data, 3-byte impure relocation program address	Used for RX3 instruction.
P	1 halfword value	2-byte pure relocation program address	Used for halfword address.
P	2 halfword values	4-byte pure relocation program address	Used for fullword address.
P	3 halfword values	3-byte absolute data, 3-byte pure relocation program address	Used for RX3 instruction.

TABLE 3-2 CODE TABLE (Continued)

CODE	NUMBER AND TYPE OF ARGUMENTS	OBJECT CODE DESCRIPTION	NOTES
AI	2 halfword values	2-byte absolute data, 2-byte impure relocation program address	Used for halfword immediate instructions that reference an impure address.
AP	2 halfword values	2-byte absolute data, 2-byte pure relocation program address	Used for halfword immediate instructions that reference a pure address.
ABS	1-n halfword values where n is the maximum number of values that can be input on a command line	2-n bytes absolute data	This is the code default. It is also the only valid code for image module modifications.
TI	1 halfword value	2-byte impure translation table address	
TP	1 halfword value	2-byte pure translation table address	
EN	exname	Entry reference instruction	Exname is used to create an additional loader item that creates a new entry point.
WN	exname	Weak entry reference instruction	Exname is used to create an additional loader item that creates a new weak entry point.
DN	exname	Data entry reference instruction	Exname is used to create an additional loader item that creates a new data entry point.

TABLE 3-2 CODE TABLE (Continued)

CODE	NUMBER AND TYPE OF ARGUMENTS	OBJECT CODE DESCRIPTION	NOTES
EX	2 halfword values, exname, exdisp (optional)	External reference instruction	The two halfword values specify the first 24 bits of an RX3 instruction. A third halfword (=0) is added by Patch to complete the RX3 instruction. Exname is used to create an additional loader item that specifies the external address referenced by the RX3 instruction. Exdisp is an offset from the address of the external reference specified by exname.
WX	2 halfword values, exname exdisp (optional)	Weak external instruction	The two halfword values specify the first 24 bits of an RX3 instruction. A third halfword (=0) is added by Patch to complete the RX3 instruction. Exname is used to create an additional loader item that specifies the external address referenced by the RX3 instruction. Exdisp is an offset from the address of the external reference specified by exname.

TABLE 3-2 CODE TABLE (Continued)

CODE	NUMBER AND TYPE OF ARGUMENTS	OBJECT CODE DESCRIPTION	NOTES
CO	2 halfword values, coname or //, displacement	Common reference instructions	The two halfword values specify the first 24 bits of an RX3 instruction. A third halfword (=0) is added by Patch to complete the RX3 instruction. Coname (or // for blank) common and displacement are used to create an additional loader item specifying the common block name and the displacement within that block of the target address of the RX3 instruction.
BL	1-8 halfword values	Change data in a block data subprogram	The command uses the common block name currently set by the BLOCK command. In addition, the address is always biased by the current value of the impure bias.

NOTE

All halfword values are in hexadecimal.

3.19 MXSPACE COMMAND

This command allows the user to change the maximum amount of system space that a task can use.

Format:

MXSPACE [n]

Parameter:

n	is a 1- to 5-digit hexadecimal number specifying the new maximum number of bytes of system space that a task can use. If this parameter is omitted, the current value is displayed.

Functional Details:

Initially, this value is defined at TET or Link time. This information is also displayed by the LIB command. This command is valid only in image mode while patching a task.

Examples:

MXSPACE 15000

The above example changes the current maximum amount of system space for a task to 15,000 bytes.

MXSP

This example displays the current maximum amount of system space.

3.20 NAME COMMAND

This command renames an object module.

Format:

NAME [name]

Parameter:

name	is a 1- to 8-character alphanumeric string specifying the new name of an object module. The special characters period (.), dollar sign ($) and the commercial at sign (@) can also be used. If this parameter is omitted, the current name is displayed.

Examples:

```
NAME PROGA
```

The above example renames the current object module PROGA.

```
NAME
```

This example displays the current module name.

3.21 NEWIDNO COMMAND

This command starts a new history record for:

- a single Patch change, optionally assigning a Patch control ID,
- a set of Patch changes, optionally assigning a Patch control ID to each Patch change in the set, or
- a multiple set of Patch changes, optionally assigning a Patch control ID to the entire set.

Format:

NEWIDNO [n] [,charstring]

Parameters:

n	is a decimal number from 0 to 32,767 specifying the control number.
charstring	is a 1- to 4-character alphanumeric string.

Functional Details:

The identifier can be left out and defined later via the IDNO command, but must be defined before the next NEWIDNO command.

The current Patch control ID can be displayed by entering the IDNO command without parameters. The Patch control numbers associated with a patched module are displayed by DUMP, IDNO, REVISION and TABLE commands. If NEWIDNO is specified without parameters, a new set of changes is established without a Patch control ID.

Example:

A user has three sets of patches to make and wants to mark each set with its own Patch control ID. The first one is to fix problem number 1034, the second is an enhancement with a reference number 1279, and the third is another problem, number 92. The module is already in memory.

```
>IDNO 1034,PROB          Labels the first set of changes
                         with Patch control ID 1034-PROB.
>MOD 104:I,FFFF,FFFF     Enters patches.
>MOD 310:P,4300          Enters patches.
>NEWID                   Establishes new set of changes.
>MOD 1204:I,2391,0811    Enters patches.
>IDNO 1279,ENH           Labels the second set of changes
                         with Patch control ID 1279-ENH.
>NEWID 92,PROB           Establishes a third set of patches
                         and labels them with Patch control
                         ID 92-PROB.
>MODIFY 102C:P,400C,0312 Enters patches.
```

The REVISION command displays the following output for the patched task:

```
*TASK
 REV     1        1034-PROB
 07/24/79         14:48:46
      104:I FFFF FFFF
      310:P 4300


 REV     2        1279-ENH
 07/24/79         14:49:58
     1204:I 2301 0811


 REV     3          92-PROB
 07/24/79         14:52:15
     102C:P 400C 0312
```

3.22 OBJECT COMMAND

This command specifies that the user is going to perform object patching.

Format:

OBJECT [fd_1,[fd_2] [,LIBRARY]]

Parameters:

fd_1	is the input file descriptor of the module to be patched. The input file must be a disk file with a record size of 126 bytes.
fd_2	is the output file descriptor to receive the patched module. The output file may be NULL: or the name of a file that does not exist. If this parameter is omitted, the output file defaults to NULL:.
LIBRARY	specifies that the input file is an object library.

Functional Details:

For both fds, the default extension is .OBJ. If present, LIBRARY specifies that the input file is an object library.

The output file is created to be the same type and record size as the input file.

The OBJECT command can be issued as many times as desired during execution of Patch.

If no parameters are present in the command line, then the current input and output files are displayed. If LIBRARY is specified, a message is displayed indicating that the input file was designated as an object library.

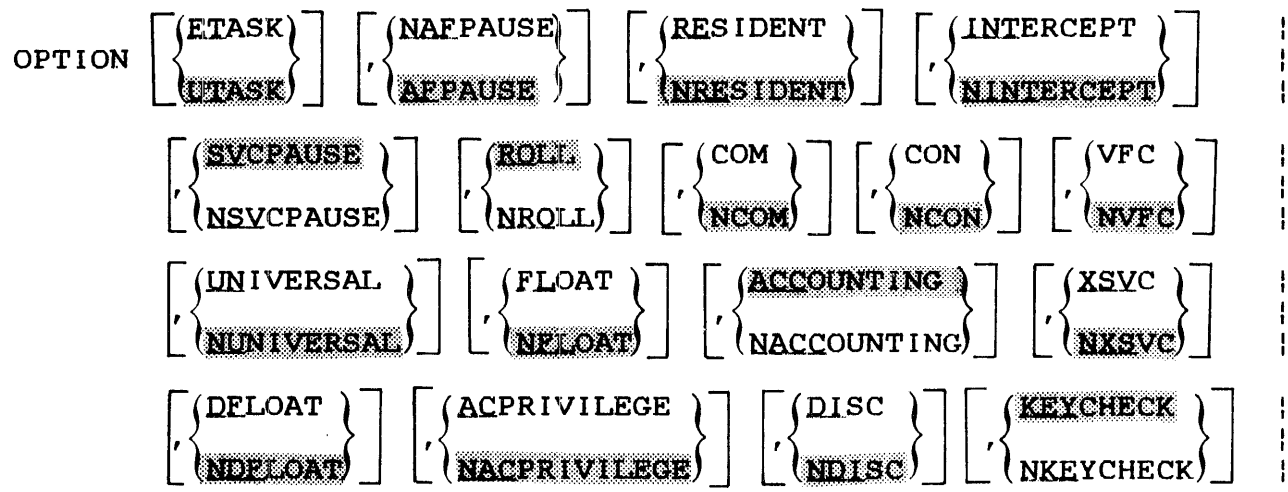
Examples:

```
->OBJECT PATCHOV1.LIB,,L
->OBJECT
- OBJECT INPUT ON M300:PATCHOV1.LIB/P
- OBJECT OUTPUT ON NULL:
- OBJECT LIBRARY FILE


->OBJECT PATCHOV1.OBJ,PATCHOV2.OBJ
->OBJECT
- OBJECT INPUT ON M300:PATCHOV1.OBJ/P
- OBJECT OUTPUT ON M300:PATCHOV2.OBJ/P
->
```

3.23 OPTION COMMAND

This command changes the task options that were initially defined at Link time. The parameters specify the new option settings.

Format:



Parameters:

ETASK	specifies that an executive task (e-task) image load module is to be built. An e-task must contain only positional-independent pure and impure code and cannot reference sharable segments.
UTASK	specifies that a user task (u-task) image load module is to be built. If both task parameters are omitted, UTASK is the default.
NAFPAUSE	specifies that the task is to continue if an arithmetic fault occurs during task execution.
AFPAUSE	specifies that the task is to pause if an arithmetic fault occurs during task execution. If both pause parameters are omitted, AFPAUSE is the default.
RESIDENT	specifies that the task is to remain in memory when it is terminated.

NRESIDENT	specifies that the task is to be removed from main storage when it is terminated. If both parameters are omitted, NRESIDENT is the default.
INTERCEPT	specifies that the task can intercept certain supervisor calls (SVCs) issued by another task before the SVC goes to the operating system for processing. If this parameter is omitted, NINTERCEPT is the default.
NINTERCEPT	specifies that the task cannot intercept the SVC of another task before the SVC goes to the operating system for processing. If both intercept parameters are omitted, the default is NINTERCEPT.
SVCPAUSE	specifies that all intertask communication and control macros entered are ignored and task execution is paused. If both pause parameters are omitted, SVCPAUSE is the default.
NSVCPAUSE	specifies that all intertask communication and control macros entered are ignored and task execution continued.
ROLL	specifies that a task can be rolled in and out of memory during task execution. If both roll parameters are omitted, ROLL is the default.
NROLL	specifies that a task cannot be rolled in and out of memory during task execution.
COM	specifies that a task can issue intertask communication.
NCOM	specifies that a task cannot issue intertask communication. If both communication parameters are omitted, NCOM is the default.
CON	specifies that a task can issue intertask control.
NCON	specifies that a task cannot issue intertask control. If both control parameters are omitted, NCON is the default.
VFC	specifies that a task uses the vertical forms control options in all I/O operations. If VFC is omitted, NVFC is the default.

NVFC — specifies that the task does not use the vertical forms control option in all I/O operations. If both the VFC parameters are omitted, NVFC is the default. Vertical forms control may still be invoked on a per lu basis and, if XSVC1 is specified, on a per I/O basis.

UNIVERSAL — specifies that a task can communicate with all other tasks in the system.

NUNIVERSAL — specifies that a task cannot communicate with all other tasks in the system. If both universal parameters are omitted, NUNIVERSAL is the default.

FLOAT — specifies that a task can execute single precision floating point (SPFP) instructions.

NFLOAT — specifies that a task cannot execute SPFP instructions. If both float parameters are omitted, NFLOAT is the default.

ACCOUNTING — specifies that the accounting function is enabled for a task. If both accounting parameters are omitted, ACCOUNTING is the default.

NACCOUNTING — specifies that the accounting function is disabled for a task.

XSVC — indicates that if the least significant bit (LSB) of a supervisor call 1 (SVC1) function code is set, an extended options fullword exists. This option must be specified to use such features as gapless mode on a 6250 magnetic tape drive or to control the use of VFC on an individual I/O basis.

NXSVC — indicates that if the LSB of an SVC1 function code is set, an image I/O is to be used. Currently, only the line printer and magnetic tape drivers use this option. ITAM drivers always operate as if XSVC1 is in effect. Other drivers always assume NXSVC1.

DFLOAT — specifies that a task can execute double precision floating point (DPFP) instructions.

NDFLOAT — specifies that a task cannot execute DPFP instructions. If both double float parameters are omitted, NDFLOAT is the default.

ACPRIVILEGE specifies that a u-task has extended file access privileges and can specify an account number instead of a file class for all SVC functions.

NACPRIVILEGE specifies that a u-task has no extended file access privileges. If both access privilege parameters are omitted, NACPRIVILEGE is the default.

DISC specifies that a u-task has an extended disk privilege and can assign a bare disk. If the disk is on-line, assignments for shared read-only (SRO) are allowed. All other assignments are rejected and a message is displayed. If the disk is marked off-line, all access privileges are allowed. See the OS/32 Programmer Reference Manual for a description of the access privileges.

NDISC specifies that a u-task has no extended disk privileges. If both disk privileges are omitted, NDISC is the default.

KEYCHECK specifies that file protection keys are checked for a privileged u-task or an e-task. If both keycheck parameters are omitted, the default is KEYCHECK.

NKEYCHECK specifies that no file protection keys are checked for a privileged u-task or e-task.

Functional Details:

This command is only valid when patching a task. Any option that is not specified remains unchanged. Illegal combinations of options cause the entire command to be rejected.

If no parameter is given, the current option settings are displayed. This information is also displayed by the LIB command.

Examples:

```
OPTION RE,NRO
```

The above example makes the task resident and unrollable, and all other parameters remain unchanged.

```
OPT NAF,NSVC
```

This example has the task continue on arithmetic faults and illegal SVC6.

```
OPTION
```

This example displays the current option settings.

3.24 OVERLAY COMMAND

This command is issued for each overlay to be patched. The GET command must be used to access the root segment before the OVERLAY command is issued. Overlays can then be examined and modified along with the root segment.

Format:

OVERLAY [name]

Parameter:

name	is the name of a tree-structured overlay.

Functional Details:

The OVERLAY command is valid only for tasks with tree-structured overlays generated by Link.

All overlays in the same path as the requested overlay are made accessible. If the requested overlay is in the same path as the previously requested overlay, both overlays will be accessible. If they are not in the same path, the previous overlay will not be available for patching.

If the name parameter is omitted, all accessible overlays are listed in the following format:

ovlyname	ovyadr	ovysiz	ovrecno	lvl	mlv	prntnode

Fields:

ovlyname	is the 1- to 8-character ASCII overlay name.
ovyadr	is the overlay start address in hexadecimal.
ovysiz	is the size in bytes of the overlay in hexadecimal.

ovrecno — is the starting record number of the overlay in the image file in decimal.

lvl — is the overlay level in decimal.

mlv — is the maximum overlay level in decimal specifying the highest level (numerically lowest) overlay that must be loaded with this overlay during program execution.

prntnode — is the overlay name of the parent node. It is used to indicate the overlay path.

Examples:

```
>OVERLAY OVERLAY2


>OVERLAY
 NAME        START   LENGTH   REC.NO   LVL   MLV   PARENT NODE
 OVERLAY2     5000      2E0      360     2     1   OVERLAY1
 OVERLAY1     2E00     3000      312     1     0   .ROOT
```

3.25 PAUSE COMMAND

This command causes Patch processing to be suspended. Multi-terminal monitor (MTM) or operating system commands can now be entered.

Format:

PAUSE

Functional Details:

A CONTINUE command causes Patch processing to resume.

Example:

PAUSE

This example suspends Patch processing.

3.26 PRIORITY COMMAND

This command allows the user to change the initial and maximum priorities of a task. This command is valid only in image mode while patching a task.

Format:

PRIORITY [inipri,maxpri]

Parameters:

inipri	is a decimal number from 10 to 249 specifying the new initial priority. The initial priority at which the task begins execution must be numerically greater than or equal to the maximum priority.
maxpri	is a decimal number from 10 to 249 specifying the new maximum priority.

Functional Details:

These priorities are initially defined at Link time. The maximum priority is the highest priority at which this task is allowed to execute.

If no parameters are given, the current values for these priorities are displayed. This information is also displayed by the LIB command.

Examples:

PRIORITY 128,100

The above example sets the initial priority of this task to 128 and its maximum priority to 100. Notice that the higher priority is numerically smaller than the lower one.

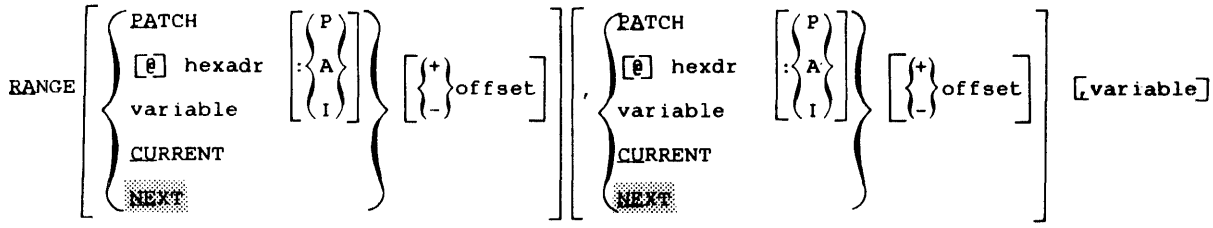
PRIORITY

This example displays the current initial and maximum priorities.

3.27 RANGE COMMAND

This command computes the relative displacement between two addresses in a form suitable for use in an RX2 format instruction.

Format:



Parameters:

PATCH	specifies that the address of the last Patch area is to be referenced.
P	specifies a pure segment.
A	specifies an absolute segment.
I	specifies an impure segment.
@	signifies that the bias is not to be added to the address.
hexadr	is a hexadecimal number from one to six digits aligned on a halfword boundary.
variable	is a 1- to 8-character string specifying the name of a previously defined internal Patch variable. See Section 3.35 for information on the VARIABLE command.
CURRENT	specifies that the last address given in an EXAMINE or MODIFY command is to be referenced.
NEXT	specifies that the halfword following the last referenced halfword (via EXAMINE or MODIFY) is to be referenced.

+	specifies an arithmetic addition.
-	specifies an arithmetic subtraction.
offset	specifies an optional hexadecimal digit. If no address is specified, the address defaults to NEXT.

Functional Details:

To obtain the relative displacement between an instruction and a target address, address specifies the instruction that requires the displacement value, and also specifies the target address. The value returned by the command can then be used to reference the target address.

Examples:

```
>RANGE E08,F00
```

The above example displays the displacement between E08 and F00.

```
RANGE:  80F4
```

In this example, if the instruction at address E08 references the address F00, then the value 80F4 that is returned by RANGE can be used in the instruction at E08 to reference either an instruction or data at address F00.

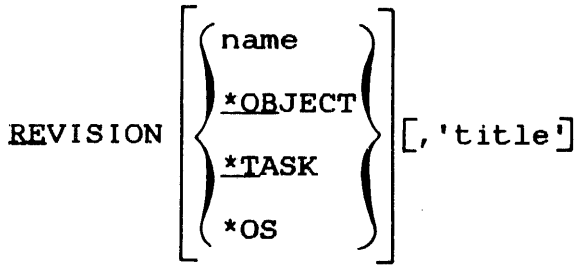
```
>VARIABLE %NEWRAN
>VARIABLE %RAN1=E08
>VARIABLE %RAN2=F00
>RANGE %RAN1,%RAN2,%NEWRAN
```

This example stores the hexadecimal value 80F4 into variable %NEWRAN and displays the displacement RANGE:80F4.

3.28 REVISION COMMAND

This command displays the previous Patch history of a module from the current input file to the list device.

Format:



Parameters:

name	is the name of a module in the input file.
*OBJECT	specifies a named or unnamed object module.
*TASK	specifies a task image module.
*OS	specifies an operating system module.
'title'	is a 1- to 50-character alphanumeric string specifying the heading to be output at the beginning of the display. If this parameter is omitted, a blank heading is used.

Functional Details:

The Patch history consists of the number of revisions, associated Patch control IDs and a list of all changes made to the module.

The first parameter selects the module by name. For unnamed object modules, task and operating system images, the special names *OBJECT, *TASK and *OS are used, respectively. When specifying the name of a resident library or task common module, only the first eight characters of the module name should be entered. Entering more than eight characters causes an error message to be output.

If the REVISION command is used for a module currently being patched, only the previous changes are displayed; none of the changes in the current session are displayed. The current position of an object library or a compound overlay file is not affected.

If the first parameter is not specified, a listing is produced of the names, number of revisions and associated Patch control IDs of all modules on the input file that have been patched. For the display, the names *TASK, *OS and *OBJECT are used for unnamed images (including operating systems produced by TET, Link and object modules, respectively).

Examples:

```
REVISION PROGA,'PROGA HISTORY'
```

The above example displays the Patch history for PROGA with heading PROGA HISTORY.

```
REV
```

This example displays all modules on the input file that have been patched.

3.29 SAVE COMMAND

This command is used to output the patched module to the output file specified in the IMAGE or OBJECT command after all desired patches are made.

Format:

SAVE [{NOCOPY / COPY}] [,TERMINATE]

Parameters:

NOCOPY	indicates that remaining modules in the input file are not to be transferred to the output file.
COPY	indicates that remaining modules in the input file are to be transferred to the output file. This parameter is the default option.
TERMINATE	indicates that no more Patch processing of the input file is to occur.

Functional Details:

After issuing the SAVE command, the user may Patch another module or terminate execution.

If the input file specified in the IMAGE or OBJECT command is an indexed file, the output file is an indexed file. If the input file specified in an IMAGE command is a contiguous file, the output file may or may not be a contiguous file. For a further explanation, see Section 3.13.

The SAVE command must be issued for each module that is modified in a compound overlay file or object library file.

Examples:

SAVE

The above example outputs the patched version of a module currently being modified to the output file. Copy the remainder of the input file to the output file and terminate Patch processing of an object library or a compound overlay file. This is the same as issuing SAVE COPY,TERMINATE.

SAVE NOCOPY

This example outputs the patched version of a module to the output file. Do not copy the remainder of an object library or compound overlay file from the input file to the output file.

SAVE, TERMINATE

This example outputs the patched version of a module currently being modified to the output file. Copy remaining modules of an object library or compound overlay file. If appropriate, change the task image file from index to contiguous. Terminate Patch processing of an object library or a compound overlay file.

3.30 SEND STOP COMMAND

This command allows a user to stop the execution of the DUMP, REVISION, EXAMINE or TABLE commands.

Format:

SEND STOP

Functional Details:

The SEND STOP command is recognized when Patch is executing either as a foreground task where the current task is the Patch task, or as a terminal task under an MTM system that supports intertask communication.

The SEND STOP command is issued in response to an operating system or MTM prompt. A prompt is obtained by depressing the BREAK key several times.

Issuing a SEND STOP command causes Patch to terminate the command that was issued just prior to the SEND STOP command. After processing the SEND STOP command, Patch is ready to accept the next command.

Examples:

Under MTM:

SEND STOP

stops execution of the current command. Patch is resumed and ready to accept the next command from the user.

Under the operating system:

TASK PATCH

selects Patch as the current task.

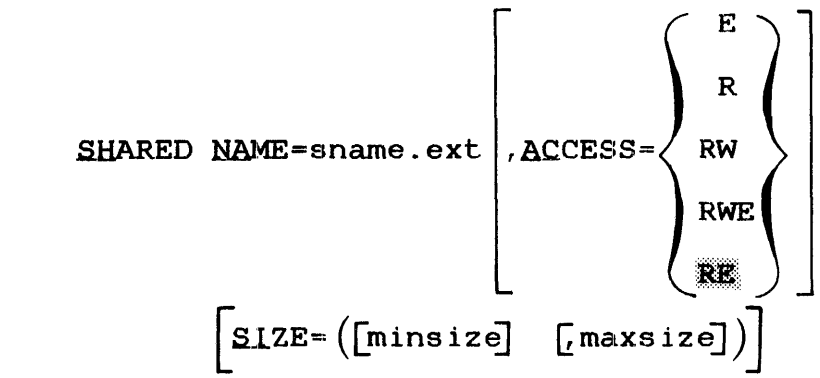
SEND STOP

stops execution of the current command.

3.31 SHARED COMMAND

This command enables the user to change the access privileges and the maximum or minimum size of a shared segment entry.

Format:



Parameters:

NAME= sname.ext is a 1- to 11-character alphanumeric string specifying the name of a shared segment.

ACCESS= E specifies that the access privilege of the sharable segment allows task execution within the sharable segment.

R specifies that the access privilege of the sharable segment allows access of data within the sharable segment. Execution or modification of data is not allowed.

RW specifies that the access privilege of the sharable segment allows access to data and modification of data within the sharable segment. Task execution is not allowed.

RWE specifies that the access privilege of the sharable segment allows access to data, modification of data, and task execution within the sharable segment.

RE specifies that the access privilege of the sharable segment allows access to data and task execution within the sharable segment.

Modification of data is not allowed. If the ACCESS= parameter is omitted, the default is RE.

SIZE= minsize is a 1- to 6-digit hexadecimal number specifying the minimum size of a sharable segment.

maxsize is a 1- to 6-digit hexadecimal number specifying the maximum size of a sharable segment.

Example:

```
SH NA= F7RTL.SEG,AC=RE
```

This example changes access privileges for the shared segment, F7RTL.SEG.

3.32 TABLE COMMAND

This command displays the modules on the input file to the list device. If they have been patched, the number of revisions and associated Patch control IDs are also displayed.

Format:

TABLE ['title']

Parameter:

'title'	is a 1- to 50-character alphanumeric string to be output as a heading at the beginning of the display. If this parameter is omitted, a blank heading is used.

Functional Details:

The names *OBJECT and *TASK are used for unnamed object and image modules. *TASK is used both for tasks and operating system images produced by TET. Operating system images produced by Link are identified as *OS.

For tasks containing tree-structured overlays, the overlay names are displayed with their starting address, length, first record numbers, level, maximum level and name of parent node.

Examples:

```
TABLE 'SAMPLE TABLE LISTING'
```

The above example produces a labeled listing of the input file.

```
TABLE
```

This example produces an unlabeled listing of the input file.

3.33 TRANSFER COMMAND

This command changes the transfer address of an object module.

Format:

TRANSFER [{ adr / variable } : { P / A / I }]

Parameters:

adr	is a 1- to 6-digit hexadecimal number specifying the new transfer address. If this parameter is omitted, the current transfer address is displayed.
variable	is a 1- to 8-character string specifying the name of a previously defined internal Patch variable. See Section 3.35 for information on the VARIABLE command.
P	specifies the pure segment.
A	specifies the absolute segment.
I	specifies the impure segment.

Functional Details:

This command can be used even if no transfer address had been originally specified.

Examples:

```
TRANSFER 1234:P
```

The above example sets the transfer address of this module to 1234 in the pure segment.

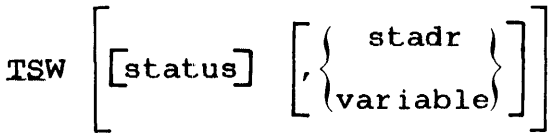
```
TR
```

This example displays the current transfer address.

3.34 TSW COMMAND

This command changes the starting TSW for a task.

Format:



Parameters:

status	is a 1- to 8-digit hexadecimal number specifying the status portion of the TSW.
stadr	is a 1- to 6-digit hexadecimal number specifying the start address of the task.
variable	is a 1- to 8-character string specifying the name of a previously defined internal Patch variable. See Section 3.35 for information on the VARIABLE command.

Functional Details:

Initially, the starting TSW is defined at Link time. If either status or start address is unspecified, the value for that parameter remains the same. If no parameters are specified, the current TSW is displayed. This information is also displayed by the LIB command. This command is valid only in image mode while patching a task.

Examples:

```
TSW 0
```

The above example changes the status portion of the initial TSW of this task to 0.

```
TSW ,100
```

This example changes the starting address of this task to 100.

```
TSW 0,100
```

This example makes both of the above changes.

```
TSW
```

This example displays the current initial status and starting address.

3.35 VARIABLE COMMAND

This command provides the ability to define an internal variable and optionally assign a value to that variable. Up to 20 variables can be created. Variable names can also be deleted in order to free space for new variable names.

Format:

VARIABLE vname [=value | ,DELETE]

Parameters:

vname	is a 1- to 8-character string specifying the name of the internal Patch variable to be defined. The first character must be a percent sign (%), the second character alphabetic and the remaining alphanumeric.
=value	is a hexadecimal number from 0 to FFFFFE aligned on a halfword boundary.
DELETE	specifies that the variable name entered is not needed in the Patch session any longer.

Functional Details:

Upon execution of the VARIABLE command, Patch scans the syntax of the vname parameter. If the syntax is correct, Patch then checks for the value. If a value is assigned to the variable, it is checked for boundaries. If the variable does not exist, it is set up by Patch and the specified value is assigned to it. A variable name that did not previously exist and is entered without a value is set up and initialized to zero. A previously defined variable entered without a value results in the current value being displayed to the user. The VARIABLE command with no parameters results in a list of all variable names and their values.

When a variable name is invalid the message 'INVALID VARIABLE NAME' is generated. Any value specified that is not valid results in the message 'INVALID VALUE SPECIFIED'. When a variable delete command is specified and the variable does not exist, the message 'VARIABLE NOT DEFINED' is generated.

If a request to add a previously defined variable is entered, the message 'DUPLICATE VARIABLE NAME' is generated. When a user attempts to list all the variables and no variables are defined, the message 'NO ENTRIES IN VARIABLE TABLE' is generated. When a VARIABLE command is entered and the maximum of 20 variables has been defined, the message 'VARIABLE TABLE FULL' is generated.

Examples:

```
>VARIABLE %NEWADD=1234
```

In the above example, the variable %NEWADD is set up and the hexadecimal value of 1234 is assigned to it.

```
>VARIABLE %NEW1,DELETE
```

In this example, the variable %NEW1 is deleted.

```
>VARIABLE %NEW1
```

In this example, the variable %NEW1 is set up and the binary value of 0 is assigned to it.

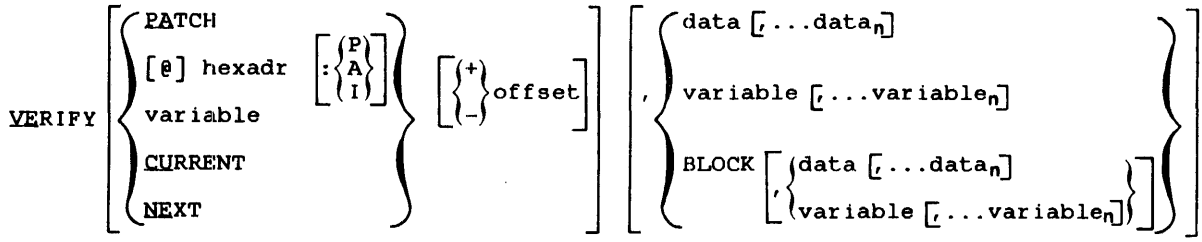
```
>VARIABLE

%NEWADD  1234
%NEW1       0
```

This example displays all of the variable names and their corresponding values.

3.36 VERIFY COMMAND

The VERIFY command provides the ability to verify the contents of a specified location in a module.

Format:



Parameters:

PATCH	specifies that the address of the last Patch area is to be referenced.
P	specifies a pure segment.
A	specifies an absolute segment.
I	specifies an impure segment.
@	signifies that the bias is not to be added to the address.
hexadr	is a hexadecimal number from 1 to 6 digits aligned on a halfword boundary.
variable	is a 1- to 8-character string specifying the name of a previously defined internal Patch variable. See the VARIABLE command in the previous section.
CURRENT	specifies that the last address given in an EXAMINE or MODIFY command is to be referenced.
NEXT	specifies that the halfword following the last referenced halfword (via an EXAMINE or MODIFY command) is to be referenced.
+	specifies an arithmetic addition.

	-	specifies an arithmetic subtraction.
	offset	specifies an optional hexadecimal digit. If no address is specified, the address defaults to NEXT.
	BLOCK	specifies that values within a block data subprogram are to be displayed.
	data	specifies hexadecimal halfword values that modify the contents starting at the location specified as the address (see Table 3-2).

Functional Details:

The VERIFY command uses the same format as the EXAMINE and MODIFY commands. The parameters hexadr and variable specify the starting location of the data to be verified. The expected value at the starting location is specified by two bytes of hexadecimal data or a previously defined variable name. Subsequent values or variable names are specified for sequential locations after the starting location. A value that is not a valid hexadecimal value or a variable name that is invalid will cause an appropriate error message in the interactive mode. An error in batch mode will cause Patch to terminate with a nonzero end of task code.

In an interactive mode, if the contents of the locations are verified against the expected values, the message 'VERIFY NO ERROR' is returned. If the contents of the locations do not match the expected values, the message 'VERIFY ERROR:LOC=nnnnnn EXPECTED=nnnn ACTUAL=nnnn' is returned. In batch mode, no message is output.

Examples:

```
>VERIFY 134:I,1234,5678
 VERIFY NO ERROR
```

The above example verifies that the two halfwords beginning at location X'134' of the impure segment contain the values 1234 and 5678.

```
>VERIFY %ONE:I,%TWO,%THREE
 VERIFY NO ERROR
```

This example verifies that the values of the variables %TWO and %THREE are contained in the two halfwords beginning at the location of %ONE.

CHAPTER 4
PATCHING IMAGE MODULES

4.1 INTRODUCTION

This chapter introduces the concepts of image patching through examples of image patching command sequences. The greater than symbol (>) is a prompt from Patch. Lines not starting with this symbol are responses from Patch. See Chapter 3 for detailed information on each command.

4.2 PATCHING A TASK IMAGE MODULE

Suppose you have an unsegmented task image in a file called TASKA.TSK and you want to change byte 43 of subroutine SUB1 from 0 to 1. All addresses and contents are in hexadecimal. Assume you have loaded and started Patch as shown in Chapter 2. The steps involved are:

- setting image mode,
- loading TASKA,
- locating SUB1 within TASKA,
- modifying byte 43 of SUB1, and
- saving the new version of TASKA.

The following example illustrates the steps involved in patching TASKA.TSK.

Example:

```
>IMAGE TASKA.TSK,NEWTASKA.TSK
>GET *TASK
>BIAS 3F00:I
    *IMPURE BIAS 3F00
>EXAMINE 42
    3F42:I 4000
>MODIFY 42,4001
>EXAMINE 42
    3F42:I 4001
>SAVE
>END
```

The IMAGE command sets the mode (image versus object), the input file and the output file. The output file must not already exist. The output file is created by Patch with the same record size and file type as the input file.

The GET command specifies that the module to be patched is a task image and causes the module to be read into the work area. For overlays, resident libraries and task commons, the module name to be patched would be specified. For operating system images, *OS would be used.

The BIAS command sets the impure bias to 3F00. If you want to modify routine SUB1, you would examine the Link or TET map for TASKA and find that subroutine SUB1 starts at 3F00. By setting the bias to 3F00, you can address locations relative to the beginning of the subroutine, using addresses from the subroutine listing (if they start from 0). All addresses have 3F00 added to them until the bias is changed; the bias is initially zero.

The I means that the bias refers to the impure segment. I is also used as the default segment descriptor for EXAMINE and MODIFY commands. The message output by Patch in this example (*IMPURE BIAS 3F00) verifies that the bias for the impure segment is set to 3F00. The asterisk (*) indicates that the default segment is the impure segment.

The EXAMINE command causes the contents of the specified location to be displayed. This command and its response also illustrate the biasing mentioned above. All addresses are even and the basic unit of data is a halfword. The parameter of the EXAMINE command specifies the address of the halfword to be displayed. The output on the next line consists of the address and the contents of the halfword location. The address has the bias added in and a segment descriptor (I) appended. For an unsegmented task, the segment descriptor I always appears, indicating an impure segment. The address parameters in the EXAMINE and MODIFY commands in this example do not need segment descriptors because the default is the impure segment as established in the previous BIAS command.

The MODIFY command changes the halfword contents, at location 3F42, from 4000 to 4001. Because of the halfword orientation, the contents of both bytes 42 and 43 are specified.

It is good practice to examine a location after it was changed to ensure that the correct location is being changed and that the change has occurred correctly.

The SAVE command causes the updated task image to be copied to the file NEWTASKA.TSK. At this point, another program can be patched by starting with a new IMAGE or OBJECT command (see Chapter 5).

The END command terminates Patch.

4.3 ADDING CODE TO IMAGE MODULES

Assume that TASKB.TSK contains a segmented task image. In addition to changing code in the pure segment, you should insert additional code, making the segment and the task larger. You should also label this change so that, subsequently, it will be possible to tell that this change was made. Finally, you should save the result in NEWTASKB.TSK.

The technique for inserting code has two steps. First, replace the two halfwords, before the insertion point, with a branch to a patch area. Second, put the replaced code in the patch area, followed by the code to be inserted, and then a branch back to the original code. The following example illustrates changing and adding code to a pure segment.

Example:

```
>IMAGE TASKB.TSK,NEWTASKB.TSK
>GET *TASK
>BIAS 1234:P
    *PURE BIAS   1234
>EXPAND P,10
    PATCH AREA  3456:P  10
>EXAMINE 42,2
     1276:P 0834 0A35
>RANGE 1276,3456
    RANGE:A1DC
>MODIFY 42,4300,A1DC
>EXAMINE CURRENT,2
    1276:P 4300 A1DC
>RANGE 345C,127A
    RANGE:DE1A
>MODIFY PATCH,834,A35,2631,4300,DE1A
>EXAMINE CURRENT,5
    3456:P 0834 0A35 2631 4300  DE1A
>IDNO 12345,PEDS
>SAVE
>END
```

The IMAGE command sets the mode, input and output files.

The GET command specifies that the module to be patched is a task image and causes the module to be read into the work area.

The BIAS command sets the bias to 1234 in the pure segment (the location where the additional code is to be inserted). It also sets the default segment descriptor for succeeding EXAMINE and MODIFY commands to P for pure.

The EXPAND command creates a patch area at the end of the pure segment which is initially all zero. The response PATCH AREA... gives the address of the beginning of the patch area, its segment descriptor and its size in bytes.

The second parameter in the first EXAMINE command specifies the decimal number of halfwords to be displayed, starting at the address specified by the first parameter. The two halfwords to be moved to the patch area are displayed.

The RANGE command gives the relative displacement between two addresses in a form suitable for subsequent use in RX2 instruction. The first parameter is the address of the instruction requiring the displacement value. The second parameter is the target address. An error message is output if the two addresses are not within RX2 range.

The first MODIFY command can change as many halfwords as can fit on the MODIFY command line, starting at the address specified in the first parameter. In this example, the two values constitute an RX2 instruction that is a branch to the patch area.

The second EXAMINE command is used to verify the modification. The first parameter, CURRENT, specifies that the address specified in the last MODIFY or EXAMINE command is to be used again.

You have inserted a branch to the patch area where new code is to be added. The code replaced by the branch instruction must be the first code inserted in the patch area.

The Patch parameter of the second MODIFY command specifies the address of the start of the pure patch area. Pure is the default segment; therefore, the segment descriptor was left out. The segment descriptor can be included in the format, PATCH:P.

The instructions replaced at 1256:P, a new instruction and a branch back to 127A:P, are put into the patch area and visually verified. The negative displacement from the patch area back into the original code was calculated with the RANGE command. You have now effectively inserted an instruction after the instruction originally at 1278.

The IDNO command associates a positive number (0 to 32,767) and a maximum of four alphanumeric characters with the patches made in the session. This label, called a patch control ID, can be displayed by any REVISION or TABLE command with NEWTASKB.TSK as the input file (see Chapter 3).

The SAVE command saves the modified task image to NEWTASKB.TSK.

The END command terminates Patch.

4.4 MODIFYING COMPOUND OVERLAY FILES CREATED BY TET

In Patch (software number 03-196 R00), the beginning of the overlay area is always shown to be at impure location zero, although the actual overlay start address is not at location zero. The correct overlay start address can be found by issuing the LIB command. The address field of the instructions displayed by the EXAMINE command has been based by TET using the overlay start address displayed in the LIB command.

The following example illustrates the loading of overlay files created by TET in Patch R00.

Example:

```
*LOAD PATCH
*START
 PERKIN-ELMER OS/32 PATCH R00-00
>IMA COBOL.OVY/S,COBOL.OV1,COMPOUND
>GET CBL006
>LIB
  SEGMENT TYPE                5 OVERLAY
  NO. OF LIB'S                2
  HISTORY RECORDS             1
  SEGMENT SIZE              295
  OVERLAY START             500
  OVERLAY NAME            CBL006
  TASK CREATION I.D.      TET32 R03-05
  DATE ESTABLISHED        20/02/80
  TIME ESTABLISHED        19:59:27
>EXA 10:I
    10:I 510
```

In Patch (software number 03-196 R02), the correct overlay start address is found by issuing the LIB or EXAMINE CURRENT command after the GET command is issued. The overlay start address must be used in the BIAS command to make addressing compatible with Patch R00. The following example illustrates the loading of overlay files created by TET in Patch R02.

Example:

```
*LOAD PATCH
*START
 PERKIN-ELMER OS/32 PATCH R02-00
>IMA COBOL.OVY/S,COBOL.CV1,COMPOUND
>GET CBL006
>EXA CU
      500:I 0000
>LIB
   SEGMENT TYPE                5 OVERLAY
   NO. OF LIB'S                2
   HISTORY RECORDS             1
   SEGMENT SIZE              295
   OVERLAY START             500
   OVERLAY NAME            CBL006
   TASK CREATION I.D.      TET32 R03-05
   DATE ESTABLISHED        20/02/80
   TIME ESTABLISHED        19:59:27
>BI    500 :I
   IMPURE BIAS  500
>EXA   10
    10:I 510
```

Assume that you want to patch the second and fourth overlays in a compound overlay file named OVYFILE.OVY that has five overlays named OVERLAYA through OVERLAYE. The result is a new compound overlay file named OVYFILE2.OVY with the new versions of OVERLAYB and OVERLAYD and the old versions of OVERLAYA, OVERLAYC and OVERLAYE.

Example:

```
>IMAGE OVYFILE.OVY,OVYFILE2.OVY,COMPOUND
>GET OVERLAYB,COPY
>EXAMINE 1000
    1000:I 1234
>MODIFY 1000,5678
>EXAMINE 1000
    1000:I 5678
>SAVE NOCOPY
>GET OVERLAYD,COPY
>EXAMINE 1000
    1000:I 1234
>MODIFY 1000,5678
>EXAMINE 1000
    1000:I 5678
>SAVE COPY,TERMINATE
>END
```

The third parameter of the IMAGE command specifies that the input file is a compound overlay file.

In addition to specifying the program to be patched, the GET command causes all overlays before OVERLAYB to be copied to the output file. Omitting the COPY keyword would still result in OVERLAYA being in the new overlay file since COPY is the default option.

The first EXAMINE command displays the contents of the first halfword starting at location X'1000'.

The MODIFY command changes the contents of location X'1000' to 5678.

The second EXAMINE command verifies the change made.

The first SAVE command saves the patched task image of OVERLAYB to OVYFILE2.OVY. NOCOPY indicates that the remaining modules in the input file are not to be transferred to the output file.

The second GET command causes OVERLAYC to be copied to the output file and specifies OVERLAYD as the next program to be patched. The previous patching sequence explanation also applies to OVERLAYD. Overlays must be selected for patching in the same order that they appear in the compound overlay file. The second SAVE command specifies that the rest of the input file is to be copied to the output file and that processing of this overlay file is to terminate.

4.5 MODIFYING TREE-STRUCTURED OVERLAYS

Tree-structured overlays differ from compound overlay files. The task with tree-structured overlays is just one image (with one set of LIBs). The compound overlay consists of a task image and one or more overlay images, each with its own set of loader information blocks (LIBs).

When patching tasks that have tree-structured overlays, the user only issues one GET. The user can specify the overlay to be modified by using the OVERLAY command.

Example:

```
>IMAGE TASKOVLY,TASKOVY2
>GET *TASK
>OVERLAY OVERLAYB
>EXAMINE 1000
  1000:I 1234
>MODIFY CURRENT,5678
>EXAMINE CUR
  1000:I 5678
>OVERLAY OVERLAYD
>EXAMINE 2100,3
 2100:I 4300 4001 023C
>MODIFY CURRENT,4320
>EXAMINE 2100,3
 2100:I4320 4003 0200
>SAVE
>END
```

The GET command specifies that a task image is to be patched.

The first OVERLAY command selects OVERLAYB for patching.

The first EXAMINE command displays the contents of the first halfword starting at location X'1000'.

The MODIFY command changes the contents of location X'1000' to 5678.

The second EXAMINE command verifies the change made.

The second OVERLAY command selects OVERLAYD for patching.

Assume that you want to patch only one overlay in a task. The name of the overlay is OVERLAYB. The result is a new task named TASKOVY2 with the new version of OVERLAYB and the old version of the task, OVERLAYA, OVERLAYC, OVERLAYD and OVERLAYE.

Example:

```
>IMAGE TASKOVLY,TASKOVY2
>GET OVERLAYB
>EXAMINE 1000
  1000:I 1234
>MODIFY 1000,5678
>EXAMINE 1000
  1000:I 5678
>SAVE
>END
```

The GET command selects OVERLAYB for patching.

The first EXAMINE command displays the contents of the first halfword starting at location X'1000'.

The MODIFY command changes the contents of location X'1000' to 5678.

The second EXAMINE command verifies the change made.

The SAVE command saves the root segment and all the overlays, including the patched OVERLAYB, to the file TASKOVY2.TSK.

CHAPTER 5
PATCHING OBJECT MODULES

5.1 INTRODUCTION

This chapter, through examples of object patching command sequences, introduces concepts of object patching. See Chapter 3 for more detailed information on each command.

5.2 PATCHING AN OBJECT MODULE

Assume that you want to change byte 43 of SUB1 from 0 to 1 in the object code. The steps involved are:

- setting object mode,
- loading SUB1 for patching,
- modifying byte 43, and
- saving the new version of SUB1.

The following example illustrates the steps involved in patching SUB1.

Example:

```
>OBJECT SUB1.OBJ,NEWSUB1.OBJ
>GET SUB1
>EXAMINE 42
    0042:I 4000
>MODIFY 42,ABS,4001
>EXAMINE 42
    0042:I 4001
>SAVE
>END
```

The OBJECT command sets the mode (object versus image), the input file and the output file. The output file must not already exist and will be created with the same record size and file type as the input file.

The GET command specifies the name of the object module to be patched and reads it into the work area. If the object module was unnamed (no PROG statement in the source), *OBJECT would be used as the parameter of GET.

The EXAMINE command specifies the address of the halfword to be displayed. Note that all addresses are even and the basic unit of data is a halfword. The output on the next line consists of the address and the contents of the halfword location. The address has a segment descriptor appended (I indicating impure). The address parameters in the EXAMINE and MODIFY commands in this session do not need segment descriptors, because the default segment is the impure segment.

The MODIFY command changes the halfword value at 42 from 4000 to 4001. Because of the halfword orientation, the value of byte 42 is also specified. The keyword 'ABS' specifies that the type of data to be inserted is absolute.

Neither EXAMINE command is necessary, but it is good practice to ensure that the correct location is being changed and that the change has occurred correctly.

The SAVE command causes the updated object program to be copied to the file NEWSUB1.OBJ. At this point, another program can be patched by starting with a new OBJECT or IMAGE command. See Chapter 3 for image patching.

The END command terminates Patch.

5.3 PATCHING A BLOCK DATA SUBPROGRAM

If the change made in the previous section was to be made within a common block named COMMONA in a block data subprogram, a slightly different procedure would be followed. The following example illustrates the use of the BLOCK command in conjunction with the BL code in the MODIFY command.

Example:

```
>OBJECT SUB2.OBJ,NEWSUB2.OBJ
>GET SUB2
>BLOCK COMMONA
    COMMON BLOCK COMMONA
>EXAMINE 42,BLOCK
    0042:I 4000
>MODIFY 42,BL,4001
>EXAMINE 42,BLOCK
    0042:I 4001
>SAVE
>END
```

The OBJECT and GET commands perform as described in Section 5.2.

The BLOCK command selects the common block named COMMONA. The addresses of subsequent EXAMINE and MODIFY commands (with BLOCK keyword and BL code, respectively) are treated as offsets from the beginning of COMMONA.

The EXAMINE and MODIFY commands display and change the contents of specified locations within a block data subprogram. Offsets within common blocks are determined by the address specified in the EXAMINE or MODIFY command plus the current impure bias. In this example the impure bias is zero, which is the initial value for all biases.

The SAVE and END commands save the patched object module and terminate patch.

5.4 ADDING CODE TO OBJECT MODULES

Assume that SUB3.OBJ contains an object module with both pure and impure code. In this case, you would like to insert some pure code, making the object module larger.

The technique used has two steps. First, replace the two halfwords before the insertion point with a branch to a patch area. Second, put the replaced code in the patch area, followed by the code to be inserted, and then a branch back to the original code. You would also like to label this change so that it is possible to tell that a patch has been applied. Finally, the result is saved on NEWSUB3.OBJ. The following example illustrates changing and adding code to a pure segment.

Example:

```
>OBJECT SUB3.OBJ,NEWSUB3.OBJ
>GET SUB3
>EXPAND P,14
    PATCH AREA 1234:P 14
>EXAMINE 42:P,2
    0042:P 0834 0A35
>RANGE 42,1234
    RANGE:91EE
>MODIFY 42:P,ABS,4300,91EE
>EXAMINE 42:P,2
    0042:P 4300 91EE
>BIAS 1234:P
    PURE BIAS 1234
>RANGE 123E,46
    RANGE:EE04
>MODIFY 0,ABS,834,A35
>MODIFY 4,CO,5A30,4000,COMMONA1,8
>MODIFY A,ABS,4300,EE04
>EXAMINE 0,5
    1234:P 0834 0A35 5A30 4000 0000
>EXAMINE 4,COMMON
    1238:P COMMONA1 8
>EXAMINE A,2
    123E:P 4300 EE04
>IDNO 12345,PEDS
>SAVE
>END
```

The OBJECT and GET commands are similar to those used in the previous two examples. The EXPAND command creates a patch area at the end of the pure code, the contents of which are initially undefined. The response gives the address of the beginning of the patch area, its segment descriptor and its size in bytes.

The second parameter in the first EXAMINE command specifies, in decimal, the number of halfwords to display, starting at the address specified by the first parameter. Here, the two halfwords to be moved to the patch area are displayed.

The first RANGE command gives the relative displacement between two addresses in a form suitable for subsequent use in an RX2 instruction. The first parameter is the address of the instruction. The second parameter is the target address. An error message is output if the two addresses are not within RX2 range.

The first MODIFY command with ABS data type changes two halfword values starting at the address specified in the first parameter. In this example, the two values constitute an RX2 instruction, which is a branch to the patch area.

Now you have inserted a branch to the patch area where new code is to be added. The code replaced by the branch instruction must be the first code inserted in the patch area.

After verifying the change, the pure bias is set to the beginning of the patch area. This allows locations to be addressed relative to the beginning of the patch area in subsequent EXAMINE and MODIFY commands. (Initially the bias is 0 and the default segment descriptor is I for impure.)

The BIAS command also sets the default segment descriptor for succeeding EXAMINE and MODIFY commands to P for pure.

The second RANGE command computes the displacement from the end of the patch area back to the original code. The next MODIFY command puts the replaced code into the patch area. The ABS keyword is used because the replaced code contains no relocatable data.

The third MODIFY command adds an RX3 instruction that references a location (8) within a common block named COMMONA1. Since RX3 instructions are represented by loader items different from absolute data, a different code is used in the second parameter. The code 'CO' indicates a common reference with the next two parameters giving the absolute part of the RX3 instruction, followed by the common block name and displacement of the RX3 target address. Other codes are used for RX3 instructions that reference pure, impure or external addresses. There are also codes for address constants and for data in block data subprograms.

The fourth MODIFY command appends the instruction that branches back to the original code. Since this branch is an RX2 instruction, it can be treated as absolute data.

Each of the last three EXAMINE commands corresponds to and verifies one of the previous MODIFY commands. The fifth EXAMINE command illustrates how to display a common reference.

The IDNO command associates a positive number (0 to 32,767) and up to four alphanumeric characters with the patches made in this session. The label, called a patch control ID, can be displayed by the REVISION or TABLE command with NEWSUB3.OBJ as the input file (see Chapter 3).

The SAVE and END commands save the patched object module and terminate Patch.

5.5 MODIFYING OBJECT LIBRARIES

Assume that the second and fourth object modules in an object library named OBJLIB.LIB are to be patched. The object library has five programs named PROG1 through PROG5. The result is to be a new object library named OBJLIB2.LIB with the new versions of PROG2 and PROG4 and the old versions of PROG1, PROG3 and PROG5. The following illustrates patching the object library OBJLIB.LIB.

Example:

```
>OBJECT OBJLIB.LIB,OBJLIB2.LIB,LIBRARY
>GET PROG2,COPY
>EXAMINE 1000
    1000:I 1234
>MODIFY 1000,ABS,5678
>EXAMINE 1000
    1000:I 5678
>SAVE NOCOPY
>GET PROG4
>EXAMINE 1000
    1000:I 1234
>MODIFY 1000,ABS,5678
    1000:I 5678
>EXAMINE 1000
>SAVE,TERMINATE
>END
```

The third parameter of the OBJECT command specifies that the input file is an object library.

In addition to specifying the program to be patched, the GET command causes all programs before PROG2 to be copied to the output file. Leaving out the COPY keyword would still result in PROG1 being in the new library file since COPY is the default option.

The next three commands (MODIFY, EXAMINE and SAVE) illustrate a sample patching sequence followed by a command to save the patched version of PROG2. NOCOPY indicates that the remaining modules in the input file are not to be transferred to the output file.

The second GET command causes PROG3 to be copied to the output file and specifies PROG4 as the next module to be patched. Modules must be selected for patching in the same order that they appear in the object library.

The same patching sequence (EXAMINE, MODIFY, EXAMINE and SAVE) is applied to PROG5. The SAVE command specifies that the rest of the input file is to be copied to the output file and that no more processing of this library will occur. If the NOCOPY, TERMINATE parameters had been specified in the SAVE command, PROG5 would not be in the new library file.

APPENDIX A
COMMAND SUMMARY

```
BIAS [ { adr      } : { P } ]
     [ { variable }   { A }
                      { I }
```

```
BLOCK [name]
```

```
COMMAND fd [,RETURN]
```

```
DISPLAY { D }
        { P }
```

```
DUMP { name     }
     { *OBJECT  } [,[rec1] [,[rec2] ['title']]]
     { *TASK    }
     { *OS      }
```

```
END
```

```
EXAMINE [ { PATCH                                  }
        [ { [@] hexadr [:{ P }]                    }
        [ { variable       { A }                   } [{ + }offset]
        [ { CURRENT        { I }                   }
        [ { NEXT                                   }
                                                       [ { BLOCK [,n]  } ]
                                                       [ { COMMON      } ]
                                                       [ { n           } ]
```

```
         ┌ ┌P┐    ┐
EXPAND   │ {A},n  │ ,[variable]
         └ └I┘    ┘
```

```
     ┌ name    ┐
GET  │ *OBJECT │  [,{NOCOPY}]
     │ *TASK   │    { COPY }
     └ *OS     ┘
```

```
HELP [command name]
```

```
IDNO [n] [,charstring]
```

```
IMAGE [fd1,[fd2] [,COMPOUND] [,NOHISTORY] [,INPLACE]]
```

```
LIB [{n}]
     {1}
```

```
LIST [fd]
```

```
LOG [{ON }]
     {OFF}
```

```
MAXLU [n]
```

```
        ┌ ┌ PATCH                ┐              ┐
        │ │ [e] hexadr [:{P}]    │              │
        │ │               {A}    │              │
        │ │               {I}    │              │
MODIFY  │ { variable             } [{+}offset]  │ [,code] [,{data [,data_n]                 }]
        │ │ CURRENT              │   {-}        │          {variable [,variable_n]}
        └ └ NEXT                 ┘              ┘

        [{exname [,exdisp]}]
        [{coname,codisp   }]
        [{//,codisp       }]
```

MXSPACE [n]

NAME [name]

NEWIDNO [n] [,charstring]

OBJECT [fd_1,[fd_2] [,LIBRARY]]

OPTION [{ETASK / UTASK}] [,{NAFPAUSE / AFPAUSE}] [,{RESIDENT / NRESIDENT}] [,{INTERCEPT / NINTERCEPT}]
[,{SVCPAUSE / NSVCPAUSE}] [,{ROLL / NROLL}] [,{COM / NCOM}] [,{CON / NCON}] [,{VFC / NVFC}]
[,{UNIVERSAL / NUNIVERSAL}] [,{FLOAT / NFLOAT}] [,{ACCOUNTING / NACCOUNTING}] [,{XSVC / NXSVC}]
[,{DFLOAT / NDFLOAT}] [,{ACPRIVILEGE / NACPRIVILEGE}] [,{DISC / NDISC}] [,{KEYCHECK / NKEYCHECK}]

OVERLAY [name]

PAUSE

PRIORITY [inipri,maxpri]

RANGE [{PATCH / [@] hexadr [:{P / A / I}] / variable / CURRENT / NEXT} [{+ / -}offset]] [,{PATCH / [@] hexdr [:{P / A / I}] / variable / CURRENT / NEXT} [{+ / -}offset]] [,variable]

REVISION [{name / *OBJECT / *TASK / *OS}] [,'title']

```
        [ { NOCOPY } ]
SAVE    [ { COPY   } ]  [,TERMINATE]

SEND STOP

                               [          { E   } ]
                               [          { R   } ]
SHARED NAME=sname.ext          [ ,ACCESS= { RW  } ]
                               [          { RWE } ]
                               [          { RE  } ]

        [SIZE=([minsize] [,maxsize])]

TABLE ['title']

            [ {   adr    }   { P } ]
TRANSFER    [ {          } : { A } ]
            [ { variable }   { I } ]

      [             [  { stadr    } ] ]
TSW   [ [status]    [, {          } ] ]
      [             [  { variable } ] ]

                 [ { =value  } ]
VARIABLE vname   [ {         } ]
                 [ { ,DELETE } ]

        [ { PATCH                     } ]  [  { data [,...data_n]                                   } ]
        [ { [@] hexadr  [ :{P,A,I} ]  } [{+}        ]   [  {                                                     } ]
VERIFY  [ { variable                  } [{-} offset ]   [ , { variable [,...variable_n]                           } ]
        [ { CURRENT                   } ]  [  {                                                     } ]
        [ { NEXT                      } ]  [  { BLOCK [ ,{ data [,...data_n]         } ]           } ]
                                              [  {       [  { variable [,...variable_n] } ]           } ]
```

APPENDIX B
PATCH MESSAGE SUMMARY

aaaaa:s OUT OF RANGE

The address specified by aaaaa:s is outside the boundaries of the selected program. Use the EXPAND command to extend the boundaries of the program.

ABSOLUTE BIAS nnnnn

This is an information message given in response to a BIAS or DISPLAY command. The current absolute bias (nnnnn) is displayed and the message is preceded by an asterisk (*) if the default segment descriptor is :A.

ARG #nn - ADR ERROR

The address part of argument nn contains no data or contains nonhexadecimal characters.

ARG #nn - ADR OUT OF RANGE

The address specified by the nnth argument is negative or greater than 2^{24} -1.

ARG #nn - INCONSISTENT

The nnth argument is inconsistent with a previous argument. For example, the last record is less than the first record in a DUMP command, or conflicting options are specified in an OPTION command.

ARG #nn - INVALID NAME FORMAT

The nnth argument is not a valid name for either a program or a COMMON block. Names must be from 1 to 8 alphanumeric characters starting with an alphabetic character. For names, the commercial at sign (@), dollar sign ($) and a period (.) are acceptable as alphabetic characters.

ARG #nn - INVALID PARAMETER NAME

The nnth argument is not one of the parameter choices for that argument.

ARG #nn - INVALID SEGMENT ID

The nnth argument contains a segment descriptor that is not P, A or I.

ARG #nn - MISSING

The nnth argument is missing and is required.

ARG #nn - NO DATA ENTERED

The nnth argument is null and is required.

ARG #nn - NOT HALFWORD ADDRESS

An odd address is specified in argument nn. An even address is required.

ARG #nn - SEGMENT ID MISSING

The nnth argument is an address specified without a segment descriptor and a segment descriptor is required.

ARG #nn - SYNTAX ERROR

An invalid delimiter or improper argument format (e.g., an alphabetic character in a decimal number) is detected in argument nn.

ARG #nn - TOO MANY CHARACTERS

The character string specified in argument nn exceeds the maximum number of characters allowed.

ARG #nn - VALUE OUT OF RANGE

Argument nn is not within the range specified for that argument.

ARGUMENT(S) MISSING

The command just issued does not have all of its required arguments.

CANNOT CHANGE PROGRAM NAME

The object program being modified does not have a name. The name can only be changed if one was originally present.

COMMON BLOCK nnnnnnnn

This is an information message given in response to a BLOCK command without arguments or a DISPLAY P command. The name of the currently selected common block (nnnnnnnn) is given. The word UNDEFINED is used if no COMMON block has been selected.

COMMON BLOCK NOT DEFINED

The command just issued assumed a current common block when one was not defined. Use the BLOCK command to select a current common block.

COMMON BLOCK NOT IN PROGRAM

The currently selected common block does not exist in the current program.

CONTIGUOUS FILE ALLOCATION FAILED

An attempt to convert the output file from an indexed file to a contiguous file has failed. The indexed version of the output file is saved. (The output file can be converted at a later date to a contiguous file using OS/32 COPY.) See Section 3.29 on the SAVE command. This is an information message and does not cause termination of Patch.

CONTIGUOUS FILE RENAME ERROR

The output file has been converted from an indexed file to a contiguous file, but an error has been encountered during the rename operation. In batch mode, Patch terminates and the output is lost. In interactive mode, Patch pauses, and when continued, retries the rename operation.

DESTINATION ADR. OUT OF RX2 RANGE

The addresses specified in a RANGE command exceed the range of an RX2 instruction.

DISPLACEMENT OUT OF RANGE

In attempting to MODIFY a common reference instruction, the displacement value exceeds the size of the specified common block.

DUPLICATE VARIABLE NAME

A request to add a previously defined variable was entered.

FILE ERROR: xx LU nn eeee
USE IS ffff

There has been a supervisor call 7 (SVC7) error on logical unit (lu) nn. The error status is xx and specifies the type of error described by eeee. Possible error types are:

ASSIGNMENT ERROR
BUFFER ERROR
FILE DESCRIPTOR ERROR
I/O ERROR
NAME ERROR
PRIVILEGE ERROR
PROTECT ERROR
SIZE ERROR
TYPE ERROR
VOLUME ERROR

The field ffff indicates the usage of the specified lu. Possible values are:

COMMAND INPUT
INPUT FILE
LIST OUTPUT
MESSAGE OUTPUT
OUTPUT FILE
SCRATCH FILE

FILE IS NOT COMPOUND

The IMAGE command just issued identified the input as a compound overlay file when it is not.

NO RETURN IN EFFECT

If an end of data indicator is encountered and RETURN is not in effect, the indicator is ignored and this message is generated.

IMAGE FILES UNASSIGNED

This is an information message given in response to an IMAGE command without arguments or a DISPLAY D command. No files have been assigned by a previous IMAGE command.

IMAGE INPUT ON fd1
IMAGE OUTPUT ON fd2

This is an information message given in response to an IMAGE command without arguments or a DISPLAY D command. The input and output files specified in the last IMAGE command are given by fd1 and fd2, respectively.

IMPURE BIAS nnnnn

This is an information message given in response to a BIAS or DISPLAY P command. The current impure bias (nnnnn) is given, and the message is preceded by an asterisk (*) if the default segment descriptor is :I.

INVALID COMMAND - IMAGE + LIBRARY

An image file cannot be a LIBRARY but can be COMPOUND.

INVALID COMMAND INPUT SPECIFICATION

The file descriptor (fd) specified as the command input device in the START parameters is invalid or contains a syntax error.

INVALID COMMAND MNEMONIC

The command just issued is not a legal command.

INVALID COMMAND - OBJECT + COMPOUND

An object file cannot be COMPOUND but can be a LIBRARY.

INVALID COMMAND SEQUENCE

The command just issued requires another command to be entered first (see Section 6.2).

INVALID COMMAND SYNTAX

The syntax of the command just entered is wrong. Check for illegal arguments and delimiters. The command mnemonic must be separated from its first argument by a blank.

INVALID FD

A device name (other than NULL:) was given as an input or output file in an IMAGE or OBJECT command.

INVALID KEYWORD

A keyword other than COMMAND or LIST is specified as a START parameter.

INVALID LIST OUTPUT SPECIFICATION

The fd specified as the list device in the START parameters is invalid or contains a syntax error.

INVALID VALUE SPECIFIED

Any value specified that is not a valid hexadecimal value or exceeds the boundaries will generate this message.

INVALID VARIABLE NAME

Variable name is invalid due to syntax error(s).

I/O ERROR: xx LU nn eeee
USE IS ffff

There is an SVC1 error on lu nn. The error status is xx and specifies the type of error described by eeee. Possible error types are:

DEVICE UNAVAILABLE
END OF FILE
END OF MEDIUM
ILLEGAL/UNASSIGNED LU
PARITY/RECOVER ERROR
UNRECOVERABLE ERROR

The field ffff indicates the usage of the lu. Possible values are:

COMMAND INPUT
INPUT FILE
LIST OUTPUT
MESSAGE OUTPUT
OUTPUT FILE
SCRATCH FILE

LIST DEVICE fd

This is an information message given in response to a LIST command without arguments or a DISPLAY D command. The current list device is specified by fd.

LOG MODE = status

This is an information message given in response to a LOG command without arguments or a DISPLAY D command. The current status (ON or OFF) of command logging is displayed.

LU nn UNASSIGNED
USE IS ffff

lu nn has been closed. The field ffff specifies the usage for this lu. Possible values are:

COMMAND INPUT
INPUT FILE
LIST OUTPUT
MESSAGE OUTPUT
OUTPUT FILE
SCRATCH FILE

MISSING OR INVALID PROG. NAME

A program name is incorrect or not specified when it is required.

NESTING OF COMMAND FUNCTION ILLEGAL

This message will be generated if a COMMAND command is issued from a secondary command file while return is in effect.

NO ABS SEGMENT

An address with an absolute segment descriptor (:A) has been specified and there is no absolute code in the program. Use the EXPAND command to allow absolute code to be entered.

NO COMMON REF AT THIS ADR

An EXAMINE command with a common keyword is specified, but the address given is not the beginning of an instruction that references common.

NO ENTRIES IN VARIABLE TABLE

An attempt was made to list all variables when no variables were defined.

NO EXPAND REQUEST FOR sssss

This is an information message given in response to an EXPAND command without arguments, an EXAMINE or DISPLAY with an invalid Patch argument, an EXPAND command during INPLACE patching or a DISPLAY P command. There have been no EXPAND requests for the segment specified by sssss.

NO FILES ASSIGNED

The command just issued requires assignment of an input or output file by an IMAGE or OBJECT command and none have been assigned.

NO IMPURE SEGMENT

An address with an impure segment descriptor (:I) has been specified and there is no impure segment.

NO PATCH HISTORY FOUND

This is an information message given in response to a REVISION command. An attempt has been made to display the Patch history of a program or a file of programs and no history was found.

NO HISTORY RECORDS MAINTAINED

No history records are maintained for inplace patching. This is displayed when INPLACE keyword is specified in the IMAGE command.

NO PROGRAM LOADED FOR PATCHING

The command just issued requires a program and none has been selected. Use the GET command to select a program.

NO PURE SEGMENT

An address with a pure segment descriptor (:P) has been specified and there is no pure segment.

NO ROOM TO EXPAND

The workspace does not contain enough space to allow the requested EXPAND. Existing data can be modified but no more EXPANDs can be done. The current program can be SAVEd and the saved version used as input in a new session or the whole session can be repeated using a larger workspace size or a scratch file. Reducing the workspace size below the size of the program forces allocation of a scratch file (see Section 2.1).

NO ROOM TO RECORD HISTORY

The Patch history workspace does not contain enough space to allow the requested command. The current program should be SAVEd and the saved version used as input in a new session.

NOT IN COMMAND REPERTOIRE

The argument specified in a HELP command is not the name of a valid command. To obtain a list of Patch commands, enter HELP without any arguments.

OBJECT FILES UNASSIGNED

This is an information message given in response to an OBJECT command without arguments or a DISPLAY D command. No files have been assigned by a previous command.

OBJECT INPUT ON fd
OBJECT OUTPUT ON fd

This is an information message given in response to an OBJECT command without arguments or a DISPLAY D command. The input and output files specified in the last OBJECT command are given by fd and fd , respectively.

OPTION INPLACE PROHIBITS EXPAND

An attempt to use the EXPAND command was made while the option INPLACE was specified.

PATCH AREA aaaaa:s zzzz

This is an information message in response to an EXPAND or DISPLAY P command. The starting address (aaaaa:s) and the size (zzzz) of each Patch area created by an EXPAND command are displayed.

PATCH MODE ERROR

An attempt has been made to issue a command that is not valid for the type of program being patched.

PROGRAM NOT FOUND

The program specified in a GET command could not be found on the input file. When *OS is specified, and the specified operating system file was created using DISCINIT, an error can result if an older version of DISCINIT was used. Patch requires that the current version of DISCINIT (R02-03 or higher) be used to create the OS file.

PURE BIAS nnnnn

This is an information message given in response to a BIAS or DISPLAY P command. The current pure bias (nnnnn) is given and the message is preceded by an asterisk (*) if the default segment descriptor is P.

RANGE: nnnn

This is an information message given in response to a RANGE command. The requested RX2 displacement is nnnn.

RECORD NOT FOUND

A record number specified in a DUMP command is not within the input file limits.

REQUESTED LIB NOT FOUND

The loader information block (LIB) requested does not exist.

SAVE CURRENT PROGRAM

An attempt has been made to exit Patch (END command), or to select another program for patching (GET command) without saving the previous program. This is a warning message when issued in interactive mode. If GET or END is issued again, normal processing occurs. In batch mode, Patch terminates execution after outputting the warning message.

SCRATCH FILE ALLOCATED

In memory, workspace reserved by the LOAD command or by the OS/32 Link OPTION WORK command is not large enough to hold the program to be patched. A scratch file has been allocated for Patch workspace. This is an information message only.

TEMPFILE ASSIGNMENT FAILS

An error was encountered while attempting to assign a temporary scratch file.

TOO MANY ARGUMENTS

The command just entered has more than the maximum number of arguments for that command.

USE REVISION CMD FOR HISTORY RECORDS

An LIB command has attempted to display a history record. The REVISION command must be used.

VARIABLE NOT DEFINED

A VARIABLE DELETE command has been specified for a variable that does not exist.

VARIABLE TABLE FULL

An attempt to use the VARIABLE command has been specified when the maximum of 20 variables has been defined.

VERIFY ERROR : LOC=nnnnnn EXPECTED=nnnn ACTUAL=nnnn

The contents of the location(s) do not match the expected value(s).

VERIFY NO ERROR

The contents of the location(s) are verified against the expected value(s).

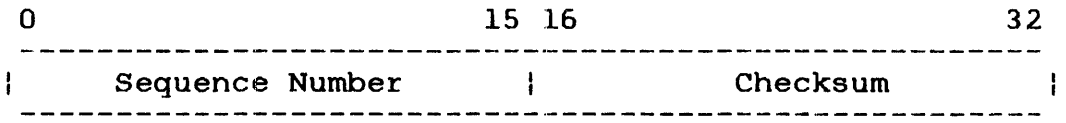
TABLE B-1 END OF TASK CODES

CODE	MEANING
0	Normal termination
1	Invalid keyword in START command
2	Invalid command input file/device
3	Command device assign error
4	Command device fetch attributes error
5	List device assign error
6	Message device assign error
7	Not used
8	Invalid list output file/device
9	START command syntax error or too many parameters
10	Input/output (I/O) error in batch mode
11	I/O error in interactive mode
12	Unrecoverable error in batch mode
20	Disk I/O error in interactive mode during history creation

APPENDIX C
PERKIN-ELMER 32-BIT OBJECT FORMAT

This appendix describes the format of object code produced by Common Assembly Language (CAL).

Modules in Perkin-Elmer 32-bit object format are divided into records. Each record contains 126 bytes of information. The first four bytes of each record are organized as follows:



The sequence numbers are sequential negative integers -1, -2, -3, etc., represented in two's complement form. The first record in a program must have sequence number -1. Subsequent records must be in proper order to be loaded.

The checksum is an Exclusive-OR sum of all halfwords in the record, except itself, plus a halfword of all ONEs.

The remainder of the record is a sequence of items; an item is a byte of loader information. There are two types of items: loader items and data items. Each loader item is followed by a certain number (which might be zero) of data items. The loader items and their meanings are listed in Table C-1.

History records are appended at the end of the object data.

TABLE C-1 LOADER ITEM DEFINITIONS

LOADER ITEM	MEANING	NUMBER OF DATA ITEMS FOLLOWING
0	End of record	None
1	End of program	None
2	Reset sequence number	None
3	Block data indicator	8-byte name, 3-byte displacement, any absolute data item (20-5B)
4	Absolute program address	3-byte address
5	Pure relocatable program address	3-byte address
6	Impure relocatable program address	3-byte address
7	2 bytes of pure relocatable data	2-byte address
8	2 bytes of impure relocatable data	2-byte address
9	4 bytes of pure relocatable data	4-byte address
A	4 bytes of impure relocatable data	4-byte address
B	Common reference	8-byte address, 3-byte displacement
C	EXTRN	8-byte name, followed by item 4, 5 or 6
D	ENTRY	8-byte name, followed by item 4, 5 or 6
E	Common definition	8-byte name, followed by a 3-byte length
F	Program label	8-character name
10	3 bytes absolute and 3 bytes pure relocatable	6 bytes
11	3 bytes absolute and 3 bytes impure relocatable	6 bytes
12	Load program transfer	Item 4, 5 or 6
13	Define start of chain (reference)	Item 4, 5 or 6
14	Load chain definition address	Item 4, 5 or 6
15	2 bytes absolute and 2 bytes pure relocatable	4 bytes
16	2 bytes absolute and 2 bytes impure relocatable	4 bytes

TABLE C-1 LOADER ITEM DEFINITIONS (Continued)

LOADER ITEM	MEANING	NUMBER OF DATA ITEMS FOLLOWING
17	Short form EXTRN	8-byte name and item 4, 5 or 6
18	Length of impure and pure segments	3-byte impure length and 3-byte pure
19	Perform fullword chain	None
1A	Perform halfword chain	None
1B	No operation	None
1C	2-byte pure translation table address	2 bytes
1D	2-byte impure translation table address	2 bytes
1E	Not used	N/A
1F	1 byte absolute data	1 byte
20	2 bytes absolute data	2 bytes
21	4 bytes absolute data	4 bytes
22	6 bytes absolute data	6 bytes
23	8 bytes absolute data	8 bytes
.	.	.
.	.	.
.	.	.
5B	120 bytes absolute data	120 bytes
5C	Define pure location counter	1-byte location number, 8-byte section name and 8-byte pool name
5D	Define impure location counter	1-byte location number, 8-byte section name and 8-byte pool name
5E	No operation	None
5F	Load program address	1-byte location number and 3-byte relocate address
60		2 bytes
61	4 bytes relocatable data	4 bytes
62	2 bytes ABS/ 2 bytes relocation	4 bytes
63	3 bytes ABS/ 3 bytes relocation	6 bytes
64	Load translate table address	1-byte location number and 2 bytes data

TABLE C-1 LOADER ITEM DEFINITIONS (Continued)

LOADER ITEM	MEANING	NUMBER OF DATA ITEMS FOLLOWING
65	Extended extrn reference	8-byte external symbol name, 1-byte flag, xxxx xx00 standard extern, xxxx xx01 weak extrn, xxxx xx10 include extrn, 4-byte offset item 4, 5 or 6
66	Extended entry	8-byte entry symbol, 1-byte flag, xxxx xx00 standard entry, xxxx xx01 data entry, xxxx xx10 weak entry item 4, 5 or 6
67	LINK commands	1-byte length and 1 to 80 characters of command
68	Declare common block	8-byte block name, 8-byte pool name and 3-byte length

APPENDIX D
PERKIN-ELMER 32-BIT IMAGE FORMAT

This appendix describes the format of image modules that can be loaded by OS/32 MTR03-01 or higher.

Modules in Perkin-Elmer 32-bit image format consist of 256-byte records. These records contain one or more loader information blocks (LIBs) followed by one or two image segments. Figure D-1 illustrates this format.

The format of an LIB depends on the type of image module. The formats for tasks, operating system images, resident libraries, task commons and overlays are given in Tables D-1 through D-7.

Patch history records are inserted following the LIBs, and before the image segments.

The image segment(s) contain the module as it appears in memory.

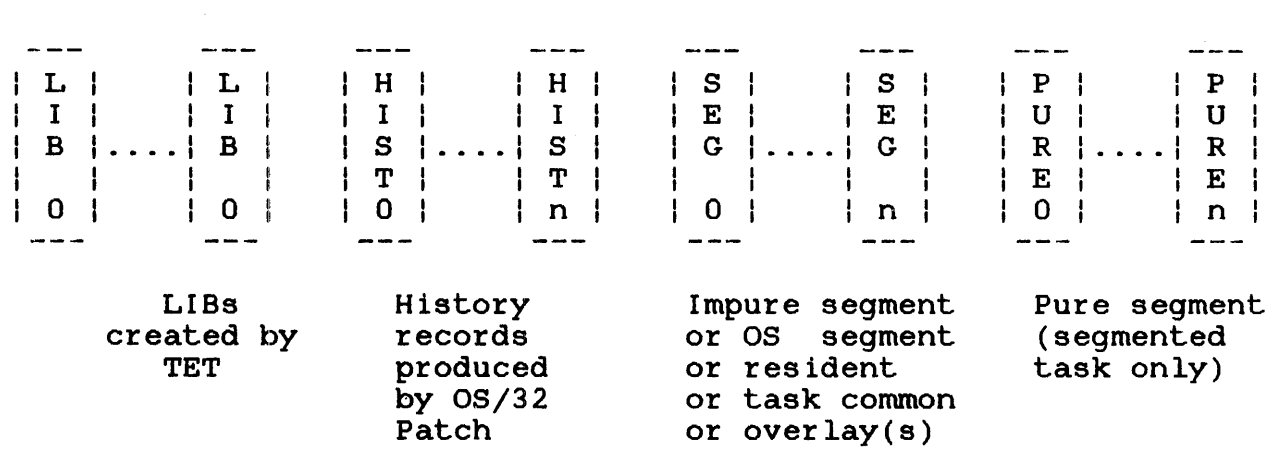


Figure D-1 Image Module Format

TABLE D-1 TASK AND OS IMAGE LIB PRODUCED BY TET

BYTE OFFSET	NUMBER BYTES	USAGE
0(0)	1	Type of module (=1 for task)
1(1)	1	Number of LIBs (including history records)
2(2)	1	Maximum number of logical units
3(3)	1	Not used
4(4)	1	Maximum priority
5(5)	1	Initial priority
6(6)	1	Pure segment register
7(7)	1	Number of task commons
8(8)	1	Number of resident libraries
9(9)	1	Not used
10(A)	2	Task options
12(C)	4	Size of impure segment in sectors
16(10)	4	Start address of overlay area
20(14)	4	Maximum system space available
24(18)	8	Initial task status word (TSW) (status + start address)
32(20)	4	Size of pure segment in sectors
36(24)	4	Starting record number of pure segment
40(28)	4	Not used
44(2C)	16	TET ID (name and revision level)
60(3C)	2	Number of history records
62(3E)	2	Starting record number of history records
64(40)	16	Not used
80(50)	16	Date/time established
96(60)	4	Not used
100(64)	4	CTOP (end of impure segment)
104(68)	4	UTOP (end of impure code and overlay area)
108(6C)	12/entry	Resident library names (name=11 bytes, segment register=1 byte)
After RL entries	16/entry	Task common names (name=11 bytes, access attribute=1 bit, unused=3 bits, segment register=4 bits, TCOM segment size=4 bytes)

NOTE

If there are enough resident library and/or task common entries, these fields may extend into a second LIB. Entries are aligned so that they do not cross LIB sector boundaries.

TABLE D-2 RESIDENT LIBRARY LIB PRODUCED BY TET

BYTE OFFSET	NUMBER BYTES	USAGE
0(0)	1	Type of module (=3 for resident library)
1(1)	1	Number of LIBs (including history records)
2(2)	1	Not used
3(3)	1	Resident library segment register
4(4)	8	Not used
12(C)	4	Size of segment in sectors
16(10)	4	Number of entry points
20(14)	12	Not used
32(20)	11	Segment name
43(2B)	1	Not used
44(2C)	16	TET ID (name and revision level)
60(3C)	2	Number of history records
62(3E)	2	Starting record number of history records
64(40)	16	Not used
80(50)	16	Date/time established
96(60)	4	Not used
100(64)	4	CTOP (end of segment)
104(68)	4	UTOP (end of code)
108(6C)	12/entry	Entry points (name=8 bytes, offset=4 bytes)

NOTE

If there are enough entry points, these fields may extend into additional LIB. Entries are aligned so that they do not cross LIB sector boundaries.

TABLE D-3 TASK COMMON LIB PRODUCED BY TET

BYTE OFFSET	NUMBER BYTES	USAGE
0(0)	1	Type of module (=4 for task common)
1(1)	1	Number of LIBs (including history records)
2(2)	10	Not used
12(C)	4	Size of segment in sectors
16(10)	16	Not used
32(20)	11	Segment name
43(2B)	1	Not used
44(2C)	16	TET ID (name and revision level)
60(3C)	2	Number of history records
62(3E)	2	Starting record number of history records
64(40)	16	Not used
80(50)	16	Date/time established
96(60)	4	Not used
100(64)	4	CTOP (end of segment)
104(68)	4	UTOP (end of data)
108(6C)	148	Not used

TABLE D-4 OVERLAY LIB PRODUCED BY TET

BYTE OFFSET	NUMBER BYTES	USAGE
0(0)	1	Type of module (=5 for overlay)
1(1)	1	Number of LIBs (including history records)
2(2)	10	Not used
12(C)	4	Size of overlay in sectors
16(10)	4	Start address of overlay area
20(14)	12	Not used
32(20)	8	Overlay name
40(28)	4	Not used
44(2C)	16	TET ID (name and revision level)
60(3C)	2	Number of history records
62(3E)	2	Starting record number of history records
64(40)	16	Not used
80(50)	16	Date/time established
96(60)	160	Not used

TABLE D-5 LIB PRODUCED BY LINK

BYTE OFFSET	NUMBER BYTES	USAGE
0(0)	1	Type of module (=7 for task)
1(1)	1	Number of LIBs (including history records)
2(2)	1	Maximum number of logical units
3(3)	1	Not used
4(4)	1	Maximum priority
5(5)	1	Initial priority
6(6)	1	Pure segment register
7(7)	1	Number of shared segments
8(8)	4	Task options
12(C)	4	Size of impure segment in sectors
16(10)	4	Address of overlay reference table in the root node
20(14)	4	Maximum system space available
24(18)	8	Initial TSW (status + start address)
32(20)	4	Size of pure segment in sectors
36(24)	4	Starting record number of pure segment
44(2C)	16	Task establishment ID
60(3C)	2	Number of history records
62(3E)	2	Starting record number of history records
64(40)	2	Number of overlay levels
66(42)	2	Number of overlay nodes
68(44)	4	Highest segmentation register used
72(48)	2	Starting record number of the overlay descriptor table (ODT)
74(4A)	2	Starting record number for impure segment
76(4C)	2	Number of shared segment entry point/commons
78(4E)	2	Not used
80(50)	16	Date/time established
96(60)	2	Maximum queued input/output (I/O) requests
98(62)	2	Length in bytes of the ODT
100(64)	4	CTOP (end of segment)
104(68)	4	UTOP (end of code)
108(6C)	20/entry	Sharable segment entry (name=11 bytes, segmentation register number=1 byte, access privileges=1 byte, minimum size=3 bytes, reserved=1 byte, maximum size=3 bytes)
After shared segment entries	16/entry	Common entry (name=8 bytes, type=1 byte, size=3 bytes, reserved=1 byte, address=3 bytes)

NOTE

If there are enough entry points, these fields may extend into additional LIB. Entries are aligned so that they do not cross LIB sector boundaries.

TABLE D-6 OPERATING SYSTEM IMAGE LIB PRODUCED BY LINK

BYTE OFFSET	NUMBER BYTES	USAGE
0(0)	1	Type of module (=8 for OS)
1(1)	1	Number of LIBs (including history records)
2(2)	10	Not used
12(C)	4	Size of segment in sectors
16(10)	4	Not used
20(14)	4	Maximum system space available
24(18)	8	Initial TSW (status + start address (=60))
32(20)	12	Not used
44(2C)	16	Task establish ID
60(3C)	2	Number of history records
62(3E)	2	Starting record number of history records
64(40)	10	Not used
74(4A)	2	Starting record number for segment
76(4C)	4	Not used
80(50)	16	Date/time established
96(60)	4	Not used
100(64)	4	CTOP (end of segment)
104(68)	4	UTOP (end of code)
108(6C)	148	Not used

TABLE D-7 SHARED SEGMENT IMAGE LIB PRODUCED BY LINK

BYTE OFFSET	NUMBER BYTES	USAGE
0(0)	1	Type of module (=9 for shared segment)
1(1)	1	Number of LIBs (including history records)
2(2)	5	Not used
7(7)	1	Number of shared segments
8(8)	4	Not used
12(C)	4	Size of segment in sectors
16(10)	4	Not used
32(20)	11	Segment name
43(2B)	17	Not used
44(2C)	16	Task establish ID
60(3C)	2	Number of history records
62(3E)	2	Starting record number of history records
64(40)	10	Not used
74(4A)	2	Starting record number for segment
76(4C)	2	Number of shared segment entry point/commons
78(4E)	2	Not used
80(50)	16	Date/time established
96(60)	4	Not used
100(64)	4	CTOP (end of segment)
104(68)	4	UTOP (end of code)
108(6C)	20/entry	Sharable segment entry (name=11 bytes, segmentation register number=1 byte, access privileges=1 byte, minimum size=3 bytes, reserved=1 byte, maximum size=3 bytes)
After shared segment entries	16/entry	Common entry (name=8 bytes, type=1 byte, size=3 bytes reserved=1 byte, address=3 bytes

NOTE

If there are enough entry points, these fields may extend into additional LIBs. Entries are aligned so that they do not cross LIB section boundaries.

INDEX

PERKIN-ELMER

PUBLICATION COMMENT FORM

We try to make our publications easy to understand and free of errors. Our users are an integral source of information for improving future revisions. Please use this postage paid form to send us comments, corrections, suggestions, *etc.*

1. Publication number_____

2. Title of publication_____

3. Describe, providing page numbers, any technical errors you found. Attach additional sheet if neccessary.

_____

_____

_____

4. Was the publication easy to understand? If no, why not?

_____

5. Were illustrations adequate? _____

_____

_____

6. What additions or deletions would you suggest? _____

_____

_____

7. Other comments: _____

_____

_____

From _____ Date _____

Position/Title _____

Company _____

Address _____

_____

_____

STAPLE

STAPLE

FOLD

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 22 OCEANPORT, N.J.

POSTAGE WILL BE PAID BY ADDRESSEE

PERKIN-ELMER

Data Systems Group
106 Apple Street
Tinton Falls, NJ 07724

ATTN:
TECHNICAL SYSTEMS PUBLICATIONS DEPT.

FOLD

FOLD

STAPLE

STAPLE