PERKIN-ELMER

OS/32 LINK

Reference Manual

48-005 F00 R01

The Perkin-Elmer Corporation, Computer Systems Division 2 Crescent Place, Oceanport, New Jersey 07757

TABLE OF CONTENTS

CHAPTERS (Continued)

CHAPTERS (Continued)

APPENDIXES (Continued)

TABLES (Continued)

PREFACE

This manual describes the new linkage editor called the Perkin-Elmer OS/32 Link, which provides the user with the ability to link one or more object modules producing an image load module. These load modules can be tasks, sharable segments, or operating systems. The user should be familiar with the Perkin-Elmer OS/32 and with the Perkin-Elmer OS/32 Multi-Terminal Monitor (MTM) if Link is used in an MTM environment.

Chapter 1 provides an introduction and overview of the features of Link. Chapter 2 describes how to build, load, and start the linkage editor. Chapter 3 lists and describes the link-edit commands. Chapter 4 guides the new user and explains the fundamentals of producing image load modules. Appendix A is a command summary. Appendix B is a message summary. Appendix C compares Link to TET. Appendix D lists and describes the TET commands.

Revision 01 of this manual adds support for these new features: trap event, vertical forms control, and supervisor call (SVC) interceptions. It also clarifies the LIBRARY command and includes changes to the OPTIONS and SHARED commands. This manual can be used with the OS/32 R06 and higher software release and with Revision 00-01 of OS/32 LINK.

The user can refer to these publications:

MANUAL TITLE	PUBLICATION NUMBER
OS/32 Operator Reference Manual	48-030
OS/32 Application Level Programmer Reference Manual	48-039
OS/32 Multi-Terminal Monitor (MTM) Reference Manual	48-043
32-Bit Systems User Documentation Summary	50-003

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

CHAPTER 1
OS/32 LINK

1.1 INTRODUCTION

The Perkin-Elmer OS/32 Link is a linkage editor that combines one or more object modules produced by Perkin-Elmer language processors and produces an image load module with all external references resolved. This image load module then can be loaded by the LOAD command.

1.2 LINK FEATURES

This linkage editor can build image load modules in sizes up to 16Mb. Tasks can be unsegmented or segmented into pure and impure segments which allows the user to share the pure code when more than one copy of the task is loaded. Task options such as floating-point requirements, workspace size, and priority can be specified. Link also can build an operating system image from the object module produced by the Perkin-Elmer OS/32 Library Loader. The resulting image load module can be loaded into main storage using the Perkin-Elmer OS/32 Bootstrap Loader (BOOT) or Loader Storage Unit (LSU).

Link automatically selects modules referenced by a user's task from specified object libraries, and links them to the main task. The library search is performed so that the order of the modules in the library has no effect on the link process.

The linkage editor can display maps with the addresses of program modules, entry points, common blocks, and overlays. A map can be a build summary, or can have symbols and addresses listed in alphabetical or address order. In addition, a cross reference listing of entry points and program modules referencing them can be included.

Link commands such as OPTION or POSITION, imbedded in an object module, are executed whenever the module is linked to a main task. The linkage editor can disable execution of these commands if a particular Link command (NDCMD) is specified by the user.

Sharable segments, such as resident libraries or task commons, are independently linked load modules that can be concurrently used by more than one task. This linkage editor can build sharable segments and link a task to the sharable segment.

Link also supports a tree-structured overlay feature. If the user specifies that routines are to be located in an overlay at link time, the linkage editor provides for automatic loading when the routine is referenced at execution time. Therefore, the overlay structure does not have to be defined in the source module. A Link command can place common areas in the root node or a specific node of the overlay structure.

1.3 LINK REQUIREMENTS

These resources are required for Link execution:

- OS/32 5.2 or higher
- 1 disc device
- 64kb of main storage for Link

1.4 STATEMENT SYNTAX CONVENTIONS

These statement syntax conventions are used in all command and instruction formats:

CONVENTION	USAGE
Capital letters, parentheses, and punctuation marks	must be entered exactly as shown.
Lowercase letters n	represent parameters or information provided by the user.
Underlining $\underline{P}$AUSE	indicates only the underlined portion of the entry is required.
Ellipsis ... $param_1$,...,$param_5$	represents an indefinite number of parameters or a range of parameters.
Lettering with shading n	represents a default option.

CONVENTION	USAGE
Braces { }	represent required parameters from which one must be chosen.
Brackets [ ]	represent an optional parameter that can be chosen.
Commas ,	separate parameters and substitute missing positional parameters.
Braces inside brackets [{ }]	represent optional parameters from which one can be chosen.
Comma preceding braces inside brackets [,{ }]	must be entered if one of the optional parameters is chosen.
Comma inside brackets [,]	must be entered if the optional parameter is chosen.
Comma outside brackets except last parameter [,[ ],[ ] [,]]	must be entered in place of missing positional parameters and to separate optional parameters that are chosen. Commas are omitted for trailing parameters and a comma must be entered with the last specified parameter.
Equal sign separating keyword from parameters KEYWORD=param	must be entered to associate parameter with keyword.

1.4.1 Link Command Syntax

Multiple commands can be entered on one line if they are separated by semicolons (;). When multiple commands are entered on the same line, they are executed sequentially. If a syntax error is detected in a command, that command plus any subsequent commands on the same line are ignored.

If the specified parameters of a command in interactive mode exceed one line, enter a comma as the last character and a carriage return (CR) which causes this message to be displayed:

```
    CONTINUE>
```

Continue entering the remaining parameters on the same line following the greater than (>) symbol. In batch mode, parameters can be continued by entering a comma as the last character and continuing the parameters on the following line.

Comments can be specified by entering an asterisk (*) before the comment string and a CR or semicolon at the end of the string. A comment can be the only data on a line or can follow a command on the same line. For example:

```
    *THIS IS THE LINK ROUTINE

    ESTABLISH TASK;*A TASK IS TO BE ESTABLISHED
```

1.4.2 File Descriptors

File descriptors, abbreviated as fd, are entered in a standard format.

Format:

$$\left[\begin{Bmatrix}\text{voln:}\\ \text{dev:}\end{Bmatrix}\right] \left[\text{filename}\right] \left[.\left[\text{ext}\right]\right] \left[/\begin{Bmatrix}\text{actno}\\ \text{file class}\end{Bmatrix}\right]$$

Parameters:

voln:	is a 1- to 4-character alphanumeric string specifying the name of a disc volume. The first character must be alphabetic and the remaining alphanumeric. If the volume name is omitted, the default is the: - volume specified by the Link VOLUME command, - volume specified by the operator or MTM VOLUME command, or - volume specified as the operating system or user default volume.

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 when a device name is specified, the filename is ignored.
.ext	is a 1- to 3-character alphanumeric string specifying the extension to a filename. If the period (.) and extension are omitted, a default extension appropriate to the particular command in which the fd appears is appended to the filename. If the period is specified and the extension is omitted, the default is blanks.
actno	is a decimal number from 0 through 255 specifying the account number associated with the file. Account numbers 1 through 240 are used by MTM; account numbers 241 through 255 are reserved. Account number 0 is used for system files and is the default for all operating system users. Specification of the account number as part of the fd can be entered when running Link from the system console.
file class	is a 1-character alphabetic string specifying the file class. The file classes are: - P for private file - G for group file - S for system file If the file class is omitted, the default is P when running Link from an MTM terminal and S when running Link from the system console.

Functional Details:

See the OS/32 Programmer Reference Manual for more information on file descriptors.

CHAPTER 2
STARTING LINK

2.1 BUILDING LINK

If the Perkin-Elmer supplied ready-to-execute version is to be used, no build is necessary. However, if a new version of Link is to be built, this sequence of commands builds Link as a sharable segmented image load module by using the Perkin-Elmer supplied version of Link.

```
    INCLUDE LINK
    OPTION SEGMENTED,WORK=8000
    BUILD LINK
    END
```

The reserved workspace must be a minimum of 8kb. The more workspace allocated, the less paging to and from disc. The less workspace allocated, the more paging to and from disc. The amount of workspace specified can be overridden at load time.

2.2 LOADING LINK

Link must be a segmented image load module before being loaded into main storage.

2.2.1 Loading Link from the System Console

This command loads Link from the system console.

Format:

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

Parameters:

taskid	is a 1- to 8-character alphanumeric string specifying the name of the task after it is loaded into the foreground segment in main storage.

fd	is the file descriptor of the device containing the linkage editor image load module to be loaded into main storage. If this parameter is omitted, the default is taskid.TSK.
segsize increment	is a decimal number in kb specifying the additional area to be added to the module's impure segment. This value overrides the WORK= option if specified when the module was built.

2.2.2 Loading Link from an MTM Terminal

This command loads Link from an MTM terminal.

Format:

LOAD fd [,segsize increment]

Parameters:

fd	is the file descriptor of the device containing the linkage editor image load module to be loaded into main storage.
segsize increment	is a decimal number in kb specifying the additional area to be added to the module's impure segment. This value overrides the WORK= option if specified when the module was built.

2.3 STARTING LINK

After Link is loaded into main storage this command starts its execution and assigns the command and log devices.

Format:

START [,COMMAND=fd_1][,LOG=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 command input

	device (CON:). If the command input device is interactive, all commands entered and messages generated are sent to the command input device. If the command input device is batch, the LOG parameter must be specified.
LOG=	fd_2 specifies the output device to which all commands entered and messages generated are recorded. If the command input device is batch, this parameter must be specified. If the log output device is a disc file, it must have been previously allocated.

Functional Details:

After the linkage editor is started, this message is iisplayed:

```
    LINK Rnn-nn
```

The revision number (Rnn) indicates the revision level of Link, and the update number (-nn) indicates the update level of Link. If the command input device is interactive, the greater than (>) symbol is then displayed as a prompt indicating the linkage editor is ready to accept commands.

CHAPTER 3
LINK COMMANDS

3.1 INTRODUCTION

There are three types of Link commands:

- Active
- Passive
- Environment

Active commands are executed as soon as they are entered and cause an immediate action to the image load module being built. Passive commands are executed when the build process occurs. Environment commands affect the link session instead of the image load module being built.

Table 3-1 lists all the Link commands, categorizes the type, and describes the function.

TABLE 3-1 LINK COMMANDS

COMMAND	TYPE			MEANING
	ACT	PAS	ENV	
BFILE			*	Backspaces a magnetic tape or contiguous file
BUILD	*			Starts building the image load module
DCMD	*			Enables execution of Link commands imbedded in object modules
END	*			Terminates the linkage editor
ESTABLISH	*			Specifies the type of image load module to be built

TABLE 3-1 LINK COMMANDS (Continued)

COMMAND	TYPE			MEANING
	ACT	PAS	ENV	
EXTERNAL		*		Specifies the name of the common block to be referenced outside a sharable segment
FFILE			*	Forward spaces a magnetic tape or contiguous file
INCLUDE	*			Specifies the object modules to be included in the image load module
LIBRARY		*		Specifies the object libraries to be searched for unresolved external references
LOCAL		*		Specifies entry points to be referenced only within a sharable segment
LOG			*	Enables logging all commands, messages, and maps to the log device
MAP		*		Generates a map when the image load module is built
NDCMD	*			Disables execution of Link commands imbedded in object modules
NLOG			*	Disables logging all commands, messages, and maps to the log device
OPTION		*		Sets task options
OVERLAY	*			Defines an overlay and specifies a level.
PAUSE			*	Pauses the linkage editor
POSITION		*		Moves a common block into a specific overlay node.
REWIND			*	Rewinds a magnetic tape or contiguous file

TABLE 3-1 LINK COMMANDS (Continued)

COMMAND	TYPE			MEANING
	ACT	PAS	ENV	
SHARED		*		Specifies a segment can be referenced by more than one task
TITLE			*	Specifies a title for the map
VOLUME			*	Specifies the default volume to be used for all subsequent file descriptors
WFILE			*	Writes a filemark on a magnetic tape or a contiguous file

* Indicates the type of Link command

3.2 BFILE COMMAND

The backspace file (BFILE) command is an environment command that backspaces a magnetic tape or contiguous file a specified number of filemarks.

Format:

```
    BFILE fd [,n]
```

Parameters:

fd	is the file descriptor of the device to be backspaced the specified number of filemarks.
n	is a decimal number specifying the number of filemarks to space backwards. If this parameter is omitted, 1 is the default.

Examples:

```
    BF MAG1:,2
```

3.3 BUILD COMMAND

The BUILD command is an active command that builds the image load module from the object modules specified in the INCLUDE command.

Format:

```
BUILD fd
```

Parameters:

fd	is the file descriptor that is to receive the image load module. If the extension is omitted, the default extensions are: .TSK for tasks .SEG for sharable segments .000 for operating systems

Functional Details:

The linkage editor attempts to allocate and assign the file specified in the BUILD command. If the file does not exist, the linkage editor allocates the file. However, if an error occurs during this process or the file is not specified in the BUILD command, this message is displayed:

```
ENTER FILE-DESCRIPTOR OF IMAGE>
```

Enter the fd of the device to receive the image load module. If the linkage editor is in batch mode and an fd is required, the build process is terminated. A pre-allocated empty indexed or contiguous file with sufficient space can be specified as the fd. If the file does not contain sufficient space, this message is displayed:

```
FILE EXISTS - DO YOU WANT IT OVERWRITTEN?>
```

If YES is entered, the file is deleted and re-allocated. If NO is entered, this message is displayed:

```
    ENTER FILE-DESCRIPTOR OF IMAGE>
```

Enter the fd to receive the image load module which causes the allocation/assignment process to be repeated.

NOTE

> Building an image load module on a contiguous file is significantly faster than building an image load module on an indexed file.

After these messages are displayed, the maps are generated if the MAP command was entered. If the MAP command was not entered, this message is displayed:

```
    MAP?>
```

If YES (Y) or NO (N) is entered, the following four messages are displayed:

```
    ENTER MAP FILE DESCRIPTOR>
```

Enter the fd of the device or file to receive the maps.

```
    SORTED ALPHABETICALLY?>
```

If YES is entered, a map with all symbols in alphabetical order is generated:

```
    SORTED BY ADDRESS?>
```

If YES is entered, a map with all symbols in address order is generated:

```
    CROSS REFERENCE?>
```

If YES is entered, a map with all symbols in alphabetical order and the names of all modules that reference each symbol is generated.

If NO was entered for all of these messages, only a build summary is generated. See section 3.13.

After the BUILD command is executed, the linkage editor is ready to build a new image load module.

Examples:

```
BU TASK

BU TASK.TSK
```

Messages:

n UNDEFINED EXTERNAL SYMBOL(S)

The specified number of undefined external symbols exist.

n MULTIPLE DEFINED ENTRY POINTS(S)

The specified number of entry points are defined more than once in the same path.

n AMBIGUOUSLY DEFINED ENTRY POINT(S)

The specified number of entry points were defined in parallel paths and referenced from a node common to both paths.

n COMMAND(S) ENCOUNTERED IN OBJECT CODE

The specified number of Link commands were encountered in the object modules included in the image load module.

3.4 DCMD COMMAND

The define command (DCMD) is an active command that enables execution of passive Link commands in object modules included in the image load module.

Format:

```
    DCMD
```

Functional Details:

When an object module with imbedded passive Link commands is included, the imbedded commands are treated as if they were entered after the INCLUDE command was entered. Imbedded LIBRARY commands are treated as if they were entered immediately before the BUILD command was entered.

Link commands can be imbedded in an object module if the CAL DCMD pseudo-op was used during an assembly. Only passive Link commands can be imbedded in object modules. Any active or environment commands imbedded in object modules are rejected and cause a message to be displayed. If a log device is specified, all Link commands in the object module are sent to the log device with a plus sign (+) in column 1. For example:

```
      ES TA
      INCLUDE MOD
     +OPTION FLOAT
      BUILD MOD
```

If an error occurs during execution of an imbedded command, a message is displayed. The format of the CAL DCMD pseudo-op is:

```
    DCMD C'linkedit command'
```

Examples:

```
    DCMD C'OPTION FLOAT'

    DCMD C'MAP PR:,ALPHA'
```

3.5 END COMMAND

The END command is an active command that terminates the linkage editor.

Format:

```
END
```

Functional Details:

If the END command is entered after passive Link commands are entered but before the BUILD command is entered, this message is displayed:

```
BUILD IMAGE FROM PREVIOUS INPUT?>
```

Enter YES if the image load module is to be built. Enter NO if no image load module is to be built and the task is to be terminated. See Table 3-2 for the list and meaning of the link end of task codes.

TABLE 3-2 LINK END OF TASK CODES

END OF TASK CODE	MEANING
0	Terminated normally
1	An error occurred but did not affect the building of the image load module.
2	An error occurred that affected the building of the image load module.
3	A severe error occurred that caused the linkage editor to abort.

3.6 ESTABLISH COMMAND

The ESTABLISH command is an active command that specifies the type of image load module to build. The three types of image load modules are:

- task,
- sharable segment, and
- operating system.

Format:

```
           ( TASK                                                              )
           |                                                                   |
           | OS                                                                |
           |                                                                   |
           |                      ( E   )                                      |
           |                      | R   |                 (m0000)              |
ESTABLISH <  SHARED [,ACCESS=    <  RE  > ] [,ADDRESS=   <       > ]           >
           |                      | RW  |                 (  *   )             |
           |                      ( RWE )                                      |
           |                                                                   |
           |     [,NAME=segment]                                               |
           (                                                                   )
```

Parameters:

TASK	specifies that a task image load module is to be built. If the ESTABLISH command is omitted, TASK is the default.
OS	specifies that an operating system image load module is to be built.
SHARED	specifies that a sharable segmented image load module is to be built.
ACCESS=	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.

	E specifies that the access privilege of the sharable segment allows 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.
	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.
ADDRESS=	m0000 is the starting address of the sharable segment. This address is the bias address used to relocate relocatable addresses in the sharable segment. The variable m is a hexadecimal number from 1 through BF. If the ADDRESS= parameter is omitted, or ADDRESS=* is specified, the sharable segment becomes address-independent and can be assigned a different starting address by each task that references it. If relocatable addresses are located in an address-independent sharable segment, they are relocated as though ADDRESS=00000 was specified and a warning message is issued.
NAME=	segname is a filename.ext that identifies the sharable segment after it is loaded into main storage. This name is matched against the name specified by the tasks that are to reference the sharable segment. If the NAME= parameter is omitted, the segment name becomes the filename.ext of the image load module.

Functional Details:

If the ESTABLISH command is entered after passive commands have been entered, this message is displayed:

```
    BUILD AN IMAGE FROM PREVIOUS INPUT?>
```

If YES is entered, a build is performed. If NO is entered, no build is performed and this message is displayed:

```
***ESTABLISHMENT ABORTED***
```

Examples:

```
ES OS
```

Establish an operating system image load module.

```
ES SHARED,ACCESS=RE,AD=F0000,NAME=SEG1
```

Establish a sharable segmented image load module.

```
ESTABLISH SHARED,ACCESS=RE,ADDRESS=A0000
```

Establish a reentrant library image load module.

```
ESTABLISH SHARED,ACCESS=RW,ADDRESS=*
```

Establish a task common image load module.

3.7 EXTERNAL COMMAND

The EXTERNAL command is a passive command that specifies the name of one or more common blocks to be referenced outside a sharable segment.

Format:

EXTERNAL common block name$_1$ [,...,common block name$_n$]

Parameters:

common block name	is the name of a common block to be referenced outside the sharable segment. See section 3.10.

Functional Details:

Common blocks are local to a sharable segment unless specified by the EXTERNAL command. External common blocks are matched against external common block references in the same way external references are matched against entry points in a segment.

3.8 FFILE COMMAND

The forward file (FFILE) command is an environment command that forward spaces a magnetic tape or contiguous file a specified number of filemarks.

Format:

```
    FFILE fd [,n]
```

Parameters:

fd	is the file descriptor of the device to be forward spaced the specified number of filemarks.
n	is a decimal number specifying the number of filemarks to space forward. If this parameter is omitted, 1 is the default.

Examples:

```
    FF MAG1:,2
```

3.9 INCLUDE COMMAND

The INCLUDE command is an active command that specifies the file containing the object modules and the specific names of object modules that are to be included in the image load module.

Format:

$$\underline{\text{IN}}\text{CLUDE } [\text{fd}] \left[\left[, \begin{Bmatrix} \text{module}_1 \\ * \end{Bmatrix} \right] \left[- \begin{Bmatrix} \text{module}_n \\ * \end{Bmatrix} \right] , \ldots , \text{module}_x \right]$$

Parameters:

fd	is the file descriptor of the file or device containing the modules to be included. If this parameter is omitted, a preassigned lu 1 or the fd specified in the last INCLUDE command entered is used. If the extension is omitted, the default is .OBJ.
module_1	is a 1- to 8-character alphanumeric string specifying the name of the next module of a range of modules to be included in the image load module. If an asterisk (*) is specified or this parameter is omitted, the next module, relative to the position of the file, is included.
module_n	is a 1- to 8-character alphanumeric string specifying the name of the last module of a range of modules to be included in the image load module. If this parameter is omitted, module1 is included. If an asterisk (*) or hyphen (-) with no module name is specified, all modules starting with module1 to the end of the file are included.

Functional Details:

If no module names are specified, all modules in the file are included.

Examples:

```
INCLUDE LIBRARY.OVY
```

Include all modules in fd LIBRARY.OVY.

```
INCLUDE LIBRARY,FIRST
```

Include the object module FIRST in fd LIBRARY.OBJ.

```
INCLUDE,SECOND-FOURTH
```

Include modules SECOND through FOURTH in the fd specified in the previous INCLUDE command.

```
INCLUDE LIBRARY.OBJ,-FOURTH,SIXTH,TENTH-*
```

Include modules FIRST through FOURTH, SIXTH, and TENTH through the end of LIBRARY.OBJ.

3.10 LIBRARY COMMAND

The LIBRARY command is a passive command that specifies object libraries to be searched at build time to resolve external references. The libraries are searched in the order they are named.

Format:

```
    LIBRARY fd1[,...,fdn]
```

Parameters:

fd	is the file descriptor of the library to be searched. If the extension is omitted, the default is .OBJ.

Functional Details:

The libraries specified are searched for entry points that match external references in the image load module being built. When a match is found, the object module is included. Only one pass is made through the list of libraries.

External references generated by the EXTRN pseudo-op are matched against library entry points. All external references generated from modules included from the library cause the library modules that resolve the external references to also be included regardless of the order of the modules.

Weak external references generated by the WXTRN pseudo-op are not matched against the library and are only resolved to entry points in modules explicitly included or to modules included from a library through external references that are not weak.

Non-linking external references generated by the INCLD pseudo-op are matched against module names in the library.

Weak entry points in the library generated by the WNTRY pseudo-op are ignored during the library search.

A module is selected from a library for the following two reasons:

1. The module is named in an INCLD pseudo-op.

2. The module contains an ENTRY or a DNTRY which can be resolved against an EXTRN in a previously included module.

Any weak entry points contained within this newly included module also become known to LINK. These weak entry points will be resolved against the list of unresolved externals, including both the standard and the weak externals.

Examples:

```
LI USER.LIB,F7RTL.OBJ
```

Specifies the user run time library and FORTRAN run time library to be searched.

3.11 LOCAL COMMAND

The LOCAL command is a passive command that specifies one or more entry points in a sharable segment can only be referenced within that segment. This command is valid only when establishing a sharable segment.

Format:

```
LOCAL entry point1 [,...,entry pointn]
```

Parameters:

entry point	is a 1- to 8-character alphanumeric string specifying the entry point to be referenced within that segment.

Functional Details:

When a sharable segment is built, all entry points can be externally referenced by tasks unless the entry points are made local to that segment by the LOCAL command.

Examples:

```
LOC ENTRY1
```

3.12 LOG COMMAND

The LOG command is an active command that specifies a new log device or starts the logging process if it was previously stopped. All command input, messages, and maps are sent to the log device.

Format:

```
    LOG fd
```

Parameters:

fd	is the file descriptor of the device or file to receive command input, messages, and maps.

Examples:

```
    LO PR:

    LO M300:LCGFILE
```

3.13 MAP COMMAND

The MAP command is a passive command that displays a map containing the names and addresses of symbols.

Format:

$$\underline{\mathrm{MA}}\mathrm{P}\ [\mathrm{fd}]\ \left[,\left\{\begin{array}{l}\underline{\mathrm{AL}}\mathrm{PHABETIC}\\ \underline{\mathrm{AD}}\mathrm{DRESS}\\ \underline{\mathrm{XR}}\mathrm{EF}\end{array}\right\}\right]$$

Parameters:

fd	is the file descriptor of the device to receive the map. If this parameter is omitted, the map is sent to the log device. However, if a log device was not previously specified, the maps are output to the command input device in interactive mode and PR: in batch mode. If the specified fd is not the same as the log device, the map is sent to both.
ALPHABETIC	specifies that the map is to contain all symbols in alphabetical order.
ADDRESS	specifies that the map is to contain all symbols in address order.
XREF	specifies that the map is to contain all the names of the modules that reference each symbol.

Functional Details:

When the MAP command is entered, a build summary map containing this information is generated:

- The name of the file to receive the image load module
- The number of logical records the image load module contains

- The size of each overlay which includes the size of its impure and pure segments, common blocks, overlay tables, and total size
- A virtual address map containing the address and size of each segment
- A list of any undefined symbols
- A list of any multiple defined symbols
- A list of any ambiguously defined symbols

Examples:

```
MAP PR:
```

The build summary is sent to the line printer.

An address map listing contains the:

- name of each symbol followed by an E (ENTRY), D (DNTRY), P (PROG), or C (COMMON),
- address of each symbol followed by a P (pure), I (impure), or A (absolute), and
- each overlay area grouped separately and in the order they were defined.

Examples:

```
MAP MAPFILE,ADDR
```

The address map is sent to the file named MAPFILE.

An alphabetic map listing contains the:

- name of each symbol followed by an E (ENTRY), D (DNTRY), P (PROG), or C (COMMON),
- address of each symbol followed by a P (pure), I (impure), or A (absolute), and
- the name of the node containing the symbol.

Examples:

```
    MAP ,ALPHA
```

The alphabetic map is sent to the log device.

The cross reference map listing contains the names of all common blocks and entry points followed by the name of the module in which they were defined and a list of all modules that reference them.

Examples:

```
    MAP PR:,XREF
```

3.14 NDCMD COMMAND

The NDCMD is an active command that disables execution of Link commands imbedded in object modules to be included in the image load module.

Format:

```
    NDCMD
```

Functional Details:

The DCMD command re-enables execution of Link commands imbedded in object modules. See section 3.4.

3.15 NLOG COMMAND

The no log (NLOG) command is an environment command and terminates logging.

Format:

```
    NLOG
```

Functional Details:

Logging can be restarted by the LOG command. See section 3.12.

3.16 OPTION COMMAND

The OPTION command is a passive command that sets task options that occur at execution time.

Format:

```
OPTION  [{ETASK }] [,{NAFPAUSE}] [,{RESIDENT  }] [,{SEGMENTED }]
        [{UTASK }] [ {AFPAUSE }] [ {NRESIDENT }] [ {NSEGMENTED}]

        [,{NROLL}] [,{COM }] [,{CON }] [,{NSVCPAUSE}]
        [ {ROLL }] [ {NCOM}] [ {NCON}] [ {SVCPAUSE }]

        [,{UNIVERSAL }] [,{DISC }] [,{ACP }] [,{FLOAT }]
        [ {NUNIVERSAL}] [ {NDISC}] [ {NACP}] [ {NFLOAT}]

        [,{DFLOAT }] [,LU=lu] [,SYSSPACE={ s  }]
        [ {NDFLOAT}]          [          {3000}]

        [,WORK=(({min}  , {max  }))] [,ABSOLUTE={ a }] [,IOBLOCKS {b
        [      (({ * }    {  *  }))] [          {100}] [          {1
        [      (({80 }    {40000}))]

        [,PRIORITY=([{ipri}] [,{mpri}])]
        [          ([{128 }] [ {128 }])]

        [,TSW=([{status}] [,{st adr}])] [,ENTRY=entry point symbol]
        [     ([{  0   }] [ {  0   }])]

        [,TCOSAVE={NONE   }] [,{XSVC1 }] [,{VFC }]
        [         {PARTIAL}] [ {NXSVC1}] [ {NVFC}]
        [         {ALL    }]

        [,{INTERCEPT }] [,{ACCOUNTING }] [,{KEYCHECK }]
        [ {NINTERCEPT}] [ {NACCOUNTING}] [ {NKEYCHECK}]
```

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 (arithmetic fault pause) 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 (nonresident) is the default.
SEGMENTED	specifies that the pure segment of a task can be shared when more than one copy of the task is loaded (except e-tasks).
NSEGMENTED	specifies that the pure segment of a task cannot be shared when more than one copy of the task is loaded (except e-tasks). If both segmented parameters are omitted, NSEGMENTED (nonsegmented) is the default.
NROLL	specifies that a task cannot be rolled in and out of memory during task execution.
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.
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 (no communication) 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 (no control) is the default.
NSVCPAUSE	specifies that all intertask communication and control macros entered are ignored and task execution continues.
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.
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, nonuniversal (NUNIVERSAL) is the default.
DISC	specifies that a u-task has an extended disc privilege and can assign a bare disc. If the disc is on-line, assignments for shared-read-only (SRO) are allowed. All other assignments are rejected and a message is displayed. If the disc 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 disc privileges. If both disc privileges are omitted, no disc (NDISC) is the default.
ACP	specifies that a u-task has extended file access privileges and can specify an account number instead of a file class for all file management functions.
NACP	specifies that a u-task has no extended file access privileges. If both access privilege parameters are omitted, no file access privileges (NACP) is the default.
FLOAT	specifies that a task can execute single precision floating point instructions.
NFLOAT	specifies that a task cannot execute single precision floating point instructions. If both float parameters are omitted, no float (NFLOAT) is the default.
DFLOAT	specifies that a task can execute double precision floating point instructions.

NDFLOAT	specifies that a task cannot execute double precision floating point instructions. If both double float parameters are omitted, no double float (NDFLOAT) is the default.
LU=	lu is a decimal number from 1 through 254 indicating the maximum quantity of logical units that can be assigned to a task.
SYSSPACE=	s is a 1- to 6-digit hexadecimal number indicating the maximum amount of system space that a task can use at run time. If this parameter is omitted, X'3000' is the default.
WORK=	min is a 1- to 6-digit hexadecimal number indicating the number of bytes of main storage to be added to the end of a task for workspace. Each time a number is specified, it is added to the current minimum value. If an asterisk (*) is specified, the minimum value is reset to zero. If this parameter is never specified, 80 bytes (X'50') is the default. When a sharable segment is being built, the default is 0. max is a 1- to 6-digit hexadecimal number indicating the maximum amount (kb) of main storage that can be added to the end of a task for workspace. If this parameter is omitted, 256kb (X'40000') is the default.
ABSOLUTE=	a specifies a 1- to 6-digit hexadecimal number indicating the number of bytes of main storage to reserve for absolute data. If this parameter is omitted, the default is 256 bytes (X'100').
IOBLOCKS=	b is a decimal number from 0 through 65,535 indicating the maximum number of I/O control blocks assigned to a task. Each I/O control block can contain one queued proceed I/O request. If this parameter is omitted, the default is one.
PRIORITY=	ipri is a decimal number from 11 through 254 indicating the initial priority of a task. The initial priority must be less than or equal to the specified maximum priority. If this parameter is omitted, the default is 128. mpri is a decimal number from 11 through 254 indicating the maximum priority of a task. If this parameter is omitted, the default is the value used for the initial priority.

TSW=	status is a 1- to 8-digit hexadecimal number indicating the initial setting of the status portion of a task's task status word (TSW). An OR operation is performed on all status word specifications to form the final status word for the image load module. If the asterisk (*) is specified, the current TSW is reset to zero. If this parameter is omitted, the default is zero.
	st adr is a 1- to 6-digit hexadecimal number indicating the starting address of the address portion of a task's TSW. This address overrides the starting address at assembly or compilation time and the starting address in the ENTRY= parameter if specified in a previous OPTION command.
ENTRY=	entry point symbol is the name of an entry point in the root node where task execution is to start. Specification of an entry point overrides the starting address specified at assembly or compilation time.
TEQSAVE=	NONE specifies that no register contents are saved and restored by OS/32 when entering and exiting a task event service routine. If this parameter and the PARTIAL parameter are omitted, ALL is the default.
	PARTIAL specifies that only the contents of registers containing event data are saved and restored when entering and exiting a task event routine. If this parameter and the NONE parameters are omitted, ALL is the default.
	ALL specifies that all register contents are saved and restored by OS/32 when entering and exiting a task event service routine. If this parameter, the NONE parameter, and the PARTIAL parameter are omitted, ALL is the default.
XSVC1	specifies that the meaning of the least significant bit of an SVC1 function code being set is that 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.
NXSVC1	specifies that the meaning of the least significant bit of an SVC1 function code being set is that 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.

VFC	specifies that a task uses the vertical forms control option in all I/O operations. If VFC is omitted, NVFC (no VFC) is the default.
NVFC	specifies that a task does not use the vertical forms control option in all I/O operations. If the VFC and NVFC 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.
INTERCEPT	specifies that this task can intercept certain SVCs of another task before the SVC goes to the operating system for processing. If this parameter is omitted, NINTERCEPT is the default.
NINTERCEPT	specifies that this task cannot intercept the SVC of another task before the SVC goes to the operating system for processing. If the INTERCEPT and NINTERCEPT parameters are omitted, the default is NINTERCEPT.
ACCOUNTING	specifies that OS/32 task accounting features are to be enabled. If this parameter is omitted, ACCOUNTING is the default.
NACCOUNTING	specifies that OS/32 task accounting features are disabled. If the ACCOUNTING and NACCOUNTING parameters are omitted, the default is ACCOUNTING.
KEYCHECK	specifies that the task option keys are checked for a privileged u-task or an E-task. If this parameter is omitted, the default is KEYCHECK.
NKEYCHECK	specifies that no task option keys are checked for a privileged u-task or an e-task. If this parameter and the KEYCHECK parameter are omitted, the default is KEYCHECK.

Examples:

```
    OP FL,RES,NAF,LU=10,WORK=3000,TSW=(,B020)
```

3.17 OVERLAY COMMAND

The OVERLAY command is an active command that defines an overlay area and specifies a level.

Format:

$$\underline{\text{OV}}\text{ERLAY overlay name}\left[,\left\{\begin{matrix}\text{level}\\ \boxed{1}\end{matrix}\right\}\right]$$

Parameters:

overlay name	is an 8-character alphanumeric string specifying the name of the overlay to be loaded into main storage. The name .ROOT is reserved for the root segment.
level	is a decimal number from 1 through 256 specifying the number of overlays between the overlay being defined and the root (inclusive). The number specified must be at most one greater than the previous level. If this parameter is omitted, the default is 1.

Functional Details:

This command is entered after all modules to be included in the root segment have been specified. Object modules to be positioned in an overlay area are included following the OVERLAY command. The sequence of defining overlays must specify the overlay and all its descendants before defining other overlays at the same level. Overlaid tasks generated by Link result in automatic loading of overlays (see Section 4.4). However, user-controlled loading of overlays is done by using SVC 5. See the OS/32 Application Level Programmer Reference Manual.

Examples:

```
INCLUDE ROOT.OBJ
  OVERLAY ONE,1
  INCLUDE A.OBJ
    OVERLAY THREE,2
    INCLUDE D.OBJ
    INCLUDE E.OBJ
    OVERLAY FOUR,2
    INCLUDE F.OBJ
  OVERLAY TWO,1
  INCLUDE B.OBJ
  INCLUDE C.OBJ
    OVERLAY FIVE,2
    INCLUDE G.OBJ
```

3.18 PAUSE COMMAND

The PAUSE command is an environment command that pauses the linkage editor.

Format:

```
    PAUSE
```

Functional Details:

The linkage editor can be continued by entering the CONTINUE command.

3.19 POSITION COMMAND

The POSITION command is a passive command that moves common blocks from their original location closer to the root segment and places them in a node that will not be initialized when an overlay is loaded.

Format:

$$\underline{PO}SITION\ \underline{CO}MMON=\left(\left\{\begin{matrix}[name_1, \ldots, name_n] \\ * \end{matrix}\right\}\right)\left[,\underline{TO}=\left\{\begin{matrix}nodename \\ .ROOT\end{matrix}\right\}\right]$$

Parameters:

COMMON=	name is a 1- to 8-character alphanumeric string specifying the name of the common block to be moved. If an asterisk (*) is specified, all common blocks are moved.
TO=	node name is a 1- to 8-character alphanumeric string specifying the name of the node to which the blocks are to be moved. If this parameter is omitted, the blocks are moved to the overlay node in which the POSITION command is encountered. If .ROOT is specified, the blocks are moved to the root segment.

Functional Details:

The placement of common blocks in a task is determined by the location in which the blocks are referenced. A common block is initially placed no closer to the root segment than any particular reference to the common block.

Examples:

```
ES TASK
INCLUDE ROOT
POSITION COMMON=(A,B)
OVERLAY OVLY1,1
INCLUDE SUB1
INCLUDE SUB2
OVERLAY OVLY2,1
INCLUDE SUB3
```

3.20 REWIND COMMAND

The REWIND command is an environment command that rewinds a magnetic tape or contiguous file.

Format:

```
    REWIND fd
```

Parameters:

fd	is the file descriptor of the device to be rewound.

Examples:

```
    RE MAG1:
```

3.21 SHARED COMMAND

The SHARED command is an active command that specifies the name of the sharable segment to be referenced by the image load module.

Format:

```
SHARED   [fd] [,NAME=segname]

                 ┌⎧ R  ⎫┐
                 │⎪ E  ⎪│ ┌            ⎧m0000⎫┐
       ,ACCESS=  │⎨ RE ⎬│ │,ADDRESS=   ⎨     ⎬│
                 │⎪ RW ⎪│ └            ⎩  *  ⎭┘
                 └⎩ RWE⎭┘
       [,STRUCTURE=(name1 [/size1] [,...,namen [/sizen]])]
       [,SIZE=([min [,max]])]
```

Parameters:

fd	is the file descriptor of the sharable segment. If this parameter is omitted, the default is a non-established task common segment defined by the operator TCOM command.
NAME=	segname is a filename.ext specifying the name of the sharable segment. If this parameter is omitted, fd must be specified, and the default is the name assigned to the sharable segment when it was created. This name is matched against the name of any sharable segments already in main storage when the task is loaded. If a sharable segment with this name is not found when the task is loaded, the segment name is treated as a file descriptor and is used to load a sharable segment.
ACCESS=	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.

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

RE specifies 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, default is RE.

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.

ADDRESS= m0000 is the starting address of the sharable segment. If the sharable segment specifies a file descriptor and the sharable segment is not address independent, the specified address must match the address specified in the sharable segment. If this parameter is omitted or not specified in the sharable segment, Link assigns an address to the sharable segment.

STRUCTURE= name is an 8-character alphanumeric string specifying the name of the task common block to be placed in the sharable segment.

size is a hexadecimal number specifying the length of the task common block. This number must be greater than or equal to the size of the common block. If this number is smaller than the current size of the task common block, a message is displayed and the size of the task common block is used. If this parameter is omitted, the default is the size of the task common block.

SIZE= min is a 1- to 6-digit hexadecimal number specifying the minimum number of bytes of main storage to be occupied by the sharable segment. If this parameter and the fd parameter are omitted, the default is the total number of bytes of all common blocks specified in the STRUCTURE= parameter or the size of segment as established. If this parameter is omitted and the fd parameter is specified, the default is the number of bytes specified when the sharable segment was built.

max is a 1- to 6-digit hexadecimal number specifying the maximum number of bytes of main storage to be occupied by the sharable segment. If this parameter and the fd parameter are omitted, the default is the total number of bytes of all common blocks specified in the STRUCTURE= parameter. If this parameter is omitted and the fd parameter is specified, the default is the number of bytes specified when the sharable segment was built.

Functional Details:

When the task referencing the sharable segment is loaded, the user-specified minimum and maximum values are compared with the actual size of the sharable segment. If the actual size is smaller than the specified minimum value, a message is displayed and the task is not loaded. If the actual size is larger than the specified maximum value, only the specified maximum value is available. If the sharable segment references other sharable segments, these references are automatically included in the image load module. However, these secondary references need not be declared again by using the SHARED command.

Examples:

```
ESTABLISH SHARED,NAME=SEGMENT.ACC,ACCESS=RW
INCLUDE COMX
BUILD COMX
END

ESTABLISH TASK
SHARED COMX,STRUCTURE=(COMX)
INCLUDE PROG1
BUILD PROG1
END

ESTABLISH SHARED,NAME=SEGMENT.ACC,ACCESS=RE,ADDRESS=E0000
INCLUDE LIB1
INCLUDE LIB2
BUILD LIBX
END

ESTABLISH TASK
SHARED LIBX
INCLUDE PROG1
BUILD PROG1
END
```

3.22 TITLE COMMAND

The TITLE command is an environment command that specifies the heading to be printed at the top of all maps.

Format:

```
    TITLE title
```

Parameters:

title	is a 60-character alphanumeric string specifying the title to be printed at the top of all maps. If the title contains a blank, comma, or semicolon, it must be enclosed within single quotation marks.

Functional Details:

The TITLE command remains in effect until a subsequent TITLE command is specified.

Examples:

```
    TI PERKIN-ELMER
    TI 'DEPARTMENT 3086'
```

3.23 VOLUME COMMAND

The VOLUME command is an environment command that specifies the volume to be used by the linkage editor when a volume is omitted in a file descriptor.

Format:

```
VOLUME [voln]
```

Parameters:

voln	is the name of the volume to be used by the linkage editor as the default. If this parameter is omitted, the current default volume is displayed on the command input device.

Functional Details:

The VOLUME command remains in effect until a subsequent VOLUME command is specified.

Examples:

```
VO M300
```

3.24 WFILE COMMAND

The WFILE command is an environment command that writes a filemark on a magnetic tape or contiguous file.

Format:

```
    WFILE fd [,n]
```

Parameters:

fd	is the file descriptor of the device to which a filemark is to be written.
n	is a decimal number specifying the number of filemarks to be written. If this parameter is omitted, 1 is the default.

Examples:

```
    WF MAG1:,2
```

CHAPTER 4
BUILDING EXAMPLES OF IMAGE LOAD MODULES USING LINK

4.1 INTRODUCTION

This chapter explains the basic concepts required to use the linkage editor through examples showing sample command build sequences. See Chapter 3 for detailed information on the Link commands.

4.2 BUILDING A SIMPLE TASK IMAGE LOAD MODULE

This example includes an object module with no external references called MOD1.CBJ, produced by the CAL Assembler, and builds a task image load module. For example:

```
INCLUDE MOD1
MAP PR1:
BUILD MOD1
END
```

The INCLUDE command specifies that all the object modules in the input file MOD1.OBJ are to be included in the build. The file extension .OBJ is the default extension for the INCLUDE command which is an active command and is executed immediately.

The MAP command specifies that a build summary is to be printed on the output device (PR1:). The MAP command is a passive command and is executed only when the BUILD command is entered.

The BUILD command builds the image load module and stores it in file MOD1.TSK. The file extension .TSK is the default extension for the BUILD command. The BUILD command is an active command and is executed immediately.

The END command is an active command and terminates the linkage editor.

4.3 BUILDING MORE COMPLEX TASK IMAGE LOAD MODULES

This section discusses building COBOL and FORTRAN task image load modules, using subroutine libraries, maps, the OPTION command, and object modules containing imbedded Link commands.

4.3.1 Building a COBOL Task Image Load Module

This example includes an object module containing external references called MOD2.OBJ produced by the COBOL compiler. The task image load module to be built is to include the single-precision floating point capability. A map is to be generated listing the names and locations of all modules and entry points in address order.

```
    INCLUDE MOD2
    LIBRARY COBOL.LIB
    OPTION FLOAT
    MAP PR1:,ADDRESS
    BUILD MOD2.TSK
    END
```

The INCLUDE command specifies that all the object modules in the input file MOD2.OBJ are to be included in the build.

The LIBRARY command specifies that the COBOL run time library file COBOL.LIB is to be searched, and any routines that contain entry points matching unresolved external references are to be included in the task image load module. The LIBRARY command is a passive command and causes the specified library to be searched when the build process occurs.

The OPTION command specifies that the single-precision floating point capability is to be included as part of the task image load module.

The MAP command specifies that a build summary and a listing of the names and locations of all modules and entry points in address order are to be generated.

The BUILD command builds the task image load module and stores it in file MOD2.TSK.

The END command terminates the linkage editor.

4.3.2 Building a FORTRAN Task Image Load Module

This example includes an object module containing external references called MOD3.OBJ produced by the OS/32 FORTRAN VII Compiler and builds an image load module. The image load module to be built is to include both single and double precision floating point capabilities, and additional workspace for the user and FORTRAN run time libraries. A map is to be generated listing the names and locations of all modules, common blocks, and entry points in alphabetical order. Also a cross reference of all entry points and the modules referencing them is to be generated.

```
    INCLUDE MOD3
    LIBRARY USERLIB,F7RTL
    OPTION DFLOAT,FLOAT,WORK=A00
    MAP PR1:,ALPHABETIC,XREF
    BUILD MOD3
    END
```

The INCLUDE command specifies that the main task in the input file MOD3.OBJ is to be included in the build.

The LIBRARY command specifies that the user library file USERLIB and FORTRAN run time library file F7RTL are to be searched in the order that they are named and that any routines containing entry points matching unresolved external references are to be included in the task image load module.

The OPTION command specifies that the single- and double-precision floating point capabilities and additional workspace for the run time libraries are to be included as part of the task image load module.

The MAP command specifies that a build summary and an alphabetical listing of the names and locations of all modules and entry points are to be generated. A cross reference of all entry points and modules referencing them is also to be generated.

The BUILD command builds the task image load module and stores it in file MOD3.TSK and the END command terminates the linkage editor.

4.3.3 Building a COBOL Task Image Load Module with Link Commands Imbedded in Object Modules

This example includes an object module, MOD4.OBJ, containing external references and imbedded Link commands produced by the COBOL compiler, and builds an image load module. The image load module to be built will include single- and double-precision floating point capabilities and additional workspace for the user and COBOL run time libraries. An alphabetical map will be generated listing the names and locations of all modules, common blocks, and entry points. Also a cross reference of all entry points and the modules referencing them will be generated. Execution of all imbedded Link commands is disabled in MOD4 by the NDCMD command and enabled by the DCMD command in the library modules. Multiple commands are entered on one line separated by a semicolon, and comment lines are used by preceding the comment with an asterisk.

```
NDCMD;*IGNORE IMBEDDED COMMANDS IN MOD4
INCLUDE MOD4;LIBRARY USERLIB,COBOL.LIB
OPTION DFLOAT,FLOAT,WORK=A00
MAP PR1:,ALPHABETIC,XREF
DCMD;*PROCESS IMBEDDED COMMANDS IN LIBRARY MODULES
BUILD MOD4
END
```

Link accepts passive commands that have been compiled or assembled into an object module. These commands are treated as if they occurred at the point where the module is included. Therefore, passive commands imbedded in object modules referenced by an INCLUDE command are treated as if they were entered immediately after the INCLUDE command. Commands imbedded in object modules referenced by a LIBRARY command are treated as if they were entered immediately before the next BUILD command. The NDCMD command causes all subsequent imbedded commands to be ignored and the DCMD command reenables this feature.

4.4 BUILDING OVERLAID TASK IMAGE LOAD MODULES

This section discusses building overlaid task image load modules using subroutines, root segments, overlay areas, root nodes, and overlay nodes. This overlay feature allows a task to be broken into sections so it can be executed using less main storage than its total size.

4.4.1 Building a Simple Overlaid Task Image Load Module

This example includes an object module called MOD5.OBJ which consists of a main task that calls three subroutines (SUBA, SUBB, and SUBC). These subroutines do not reference each other and overlay 10kb of the same main storage area if each subroutine is loaded only when needed. The main task occupies 10kb of main storage, and the largest overlay occupies 10kb of main storage which is a total of 20kb for the whole task. This task would occupy 40kb of main storage without using the overlay feature. The MAP command specifies that a build summary and a listing of the names and locations of all modules and entry points in address order are to be generated. It is assumed that all the routines are contained in file MSP.OBJ.

```
INCLUDE M300:MSP.OBJ,MOD5
OVERLAY A
INCLUDE ,SUBA
OVERLAY B
INCLUDE ,SUBB
OVERLAY C
INCLUDE ,SUBC
MAP PR1:,ADDRESS
BUILD MOD5
END
```

The INCLUDE command specifies that the object module MOD5.OBJ in the input file MSP.OBJ is to be included in the build. Because no overlays have been specified by the OVERLAY command, MOD5.OBJ becomes the main task (root segment) and is placed in the root node.

The first OVERLAY command defines an overlay area named A. The INCLUDE command specifies that the object module called SUBA is part of overlay A and will be automatically loaded into main storage if it is not already loaded when MOD5 calls SUBA. The overlay can be explicitly loaded by issuing an SVC 5 in assembly language or CALL IFETCH in FORTRAN.

The second OVERLAY command defines an overlay area named B. The INCLUDE command specifies that the object module called SUBB is part of overlay B and will be automatically loaded into the same main storage area previously occupied by overlay A, if SUBB is not already loaded when MOD5 calls it.

The third OVERLAY and INCLUDE commands define an overlay area named C and includes the object module called SUBC as part of overlay C.

The MAP command specifies that a build summary and a listing of the names and locations of the main task (root segment) and all subroutines (overlay areas) in address order are to be generated. A map of each overlay area is also produced.

The BUILD command builds the image load module called MOD5.TSK which consists of a root segment (MOD5.OBJ), an overlay area large enough to contain the largest overlay (A, B, and C), and the subroutines (SUBA, SUBB, and SUBC). The END command terminates the linkage editor.

4.4.2 Building a More Complex Overlaid Task Image Load Module

This example includes an object module called file LFP.OBJ. which consists of a main task that calls two subroutines (SUBA and SUBB). Subroutine SUBA calls two more subroutines (SUBA1 and SUBA2). Subroutine SUBB also calls two more subroutines (SUBB1 and SUBB2). In addition to SUBA and SUBB overlaying each other, SUBA1 and SUBA2 are to be overlaid when SUBA is in main storage, and SUBB1 and SUBB2 are to be overlaid when SUBB is in main storage. This overlay process can be accomplished by using another level of overlay areas. Figure 4-1 illustrates the overlay structure for this example.

```
                         -------------
                         |    MOD6     |
                         | (root node) |
                         -------------
                               |
                               |
                               |
                               |
                               |
          ---------------------|--------------------
          |                                        |
      ----------                              ----------
Level |  SUBA    |                          |  SUBB    |
  1   | (node A) |                          | (node D) |
      ----------                              ----------
          |                                        |
          |                                        |
          |                                        |
    ------|------                            ------|------
    |            |                           |            |
 ----------   ----------                ----------   ----------
Level | SUBA1    | |  SUBA2   |  |  SUBB1   | |  SUBB2   |
  2   | (node B) | | (node C) |  | (node E) | | (node F) |
 ----------   ----------                ----------   ----------
```

Figure 4-1 Sample Overlay Structure

A path is defined as a set of nodes (a group of routines loaded at one time is a node), one at each level, each of which is a descendant of the node at the previous level. For example, the root node, node D, and node E form a path. Only nodes in the same path may be in memory at the same time and, therefore, a routine may only call routines in nodes which are in the same path as the node containing the calling routine.

The overlay nodes may be different sizes and the total overlay area required at any one time is the total size of all the nodes in the current path. The size of the overlay area for the task is determined by the path with the largest total size.

It is assumed that all the subroutines are contained in file LFP.OBJ. Utility routines called in the task are in USERLIB.OBJ.

```
     INCLUDE M300:LFP.OBJ,MOD6
       OVERLAY A,1
       INCLUDE ,SUBA
         OVERLAY B,2
         INCLUDE ,SUBA1
         OVERLAY C,2
```

```
      INCLUDE ,SUBA2
    OVERLAY D,1
    INCLUDE ,SUBB
      OVERLAY E,2
      INCLUDE ,SUBB1
      OVERLAY F,2
      INCLUDE ,SUBB2
  LIBRARY USERLIB
  MAP PR1:,ADDRESS
  BUILD MOD6
  END
```

The INCLUDE command specifies that the object module MOD6 in the input file LFP.OBJ is to be included in the build. MOD6 becomes the main task (root segment).

The first OVERLAY command defines an overlay area named A with a depth level of one. The INCLUDE command specifies that the object module called SUBA is part of overlay A. All descendants of overlay A must be specified before any other overlays with a depth level of one are defined.

The second and third OVERLAY commands define overlay areas named B and C with a depth level of two which indicates that these overlays are descendants of overlay A.

The fourth OVERLAY command defines an overlay area named D with a depth level of one.

The fifth and sixth OVERLAY commands define overlay areas named E and F with a depth level of two which indicates that these overlays are descendants of overlay D.

The LIBRARY command specifies that the user library file USERLIB is to be searched and any routines containing entry points matching unresolved external references are to be included in the overlay structure being built. If a particular overlay area contains external references to a routine in the user library, a copy of that routine is placed in the referencing overlay area unless the referencing overlay area is a descendant of an overlay area that already contains a copy of that particular routine.

If modules SUBA1 and SUBA2 reference a routine called TAG located in the user library, a copy of routine TAG is included in overlay areas B and C. However, if modules SUBB and SUBB1 reference routine TAG, a copy of the routine is only included in overlay area D. If the main task MOD6 references routine TAG, a copy of the routine is only included in the root segment regardless of any other overlay areas referencing it. However, if two copies of a routine are to be included in two overlay areas (one being a descendant of the other), that routine must be explicitly included by the INCLUDE command.

The MAP command specifies that a build summary and a listing of the names and locations of the main task (MOD6) and all subroutines (SUBA, SUBA1, SUBA2, SUBB, SUBB1, and SUBB2) in address order are to be generated.

The BUILD command builds the image load module which consists of the root segment, overlay areas, and the subroutines. The END command terminates the linkage editor.

4.4.3 Moving Common Blocks

Normally, the placement of common blocks in a task is determined by where they are referenced. For example, if ALPHA is a common block referenced by routines in a particular node, ALPHA is included in that node.

If ALPHA is referenced by routines in more than one overlay node, ALPHA is included in the numerically highest level node of the path in with each node references ALPHA. This is subject to the restriction that ALPHA is not referenced in a numerically lower level node than the one in which it is placed.

If SUBA1 and SUBA2 both reference ALPHA, it is placed in node A. If routines SUBA2 and SUBB1 reference ALPHA, ALPHA is placed in the root node.

In some cases, it is desirable to place a common block in a node other than the one in which it would normally be placed which is where it is referenced. For example, placing a common block in the root node prevents the data in it from being reinitialized each time the node in which it is located is loaded.

This example moves a common block called BETA, which is referenced by routines in modules SUBA2 and SUBB1 in Figure 4-1, to the root node in the overlay structure by using the POSITION command.

```
INCLUDE M300:LFP.OBJ,MOD6
OVERLAY A,1
   .
   .
   .
LIBRARY USERLIB
POSITION COMMON=BETA,TO=.ROOT
   .
   .
   .
END
```

This command specifies that the common block named ALPHA is to be placed in the root node. Only one copy of a common block can occur in a task and an error results if an attempt is made to position a common block in a node that is at a numerically higher level or not in the same path as the node in which it would normally be placed.

4.5 BUILDING SHARABLE SEGMENTS

Sharable segments, such as blockdata modules and run time libraries, must be separately built by Link to be used or referenced by established tasks. This example includes two blockdata object modules called BDALPHA.OBJ and BDBETA.OBJ to initialize common blocks called ALPHA and BETA.

This example also includes an object file called F7RTL.OBJ to be included in a second build. The shared segment to be built is to include local and external entry points and additional workspace for the FORTRAN run time library.

```
ESTABLISH SHARED,ACCESS=RW,NAME=COMMONS
INCLUDE BDALPHA.OBJ
INCLUDE BDBETA.OBJ
EXTERNAL ALPHA,BETA
BUILD COMMONS.SEG
*THIS COMMAND SEQUENCE STARTS THE SECOND BUILD
ESTABLISH SHARED,ACCESS=RE,ADDRESS=F0000
INCLUDE F7RTL.OBJ
LOCAL .DI,.DO,.TGD,.TASKID,.HYDEX,.HYEXP
OPTION WORK=A00
BUILD F7RTL.SEG
END
```

The first ESTABLISH command specifies that the sharable segment to be built is called COMMONS.SEG with read/write access privileges. The ACCESS and NAME parameters provide information that is verified against the parameters specified in a referencing task's SHARED command or the defaults if no SHARED command is specified in a referencing task. For example, if a subsequent SHARED command in a referencing task specifies read-only access and a name of COMMONS, the access is allowed because it is a subset of the maximum access specified in the previous example and the name COMMONS matches the name specified in the previous example. A request for execute access would be rejected.

The first two INCLUDE commands include the blockdata object modules called BDALPHA.OBJ and BDBETA.OBJ.

The EXTERNAL command specifies that the two common blocks ALPHA and BETA are to be known outside the sharable segment.

Normally, common blocks are considered local. Note that either the STRUCTURE parameter in a subsequent SHARED command in a referencing task or the EXTERNAL command ,not both, are required to match up the common references in a task with the initialized common blocks in the sharable segment named COMMONS. The EXTERNAL command is passive.

The first BUILD command builds the sharable segment in file COMMONS.SEG.

The second ESTABLISH command specifies that a new sharable segment is to be built called F7RTL.SEG with read-execute access privileges only and a starting address of X'F0000' within the task referencing it. The ADDRESS parameter specifies that this segment is to start at F0000 in the address space of any task which references it. Segments that do not specify an address in either the SHARED command of the referencing task or the ESTABLISH command of the segment are address independent and may be allocated anywhere within the address space of tasks which reference them.

The third INCLUDE command includes all the FORTRAN run time library routines in F7RTL.OBJ in the sharable segment to be built.

The LOCAL command defines that the entry points .DI, .DO, .TGD, .TASKID, .HYDEX, and .HYEXP are local to the segment and cannot be referenced by tasks referencing the sharable segment.

The OPTION command specifies that additional workspace for the run time library is to be included in the sharable segment when any task references this segment.

The second BUILD command builds the sharable segment and stores it in file F7RTL.SEG. The END command terminates the linkage editor.

Sharable segments can also be created by the operator at the system console if it is to be used as an area for common blocks. These sharable segments do not require loading into main storage or initialization before they can be referenced by other tasks.

4.6 BUILDING A TASK IMAGE LOAD MODULE REFERENCING SHARABLE SEGMENTS

Link provides the capability of sharing one copy of a segment containing code and/or data areas among multiple tasks. In particular, shared common blocks allow data to be shared or communicated among tasks. Shared copies of run time libraries allow more efficient use of main storage.

This example builds a FORTRAN task. MOD7.OBJ is a FORTRAN program that references a sharable segment containing two common blocks called DELTA and GAMMA and the FORTRAN run time library.

```
INCLUDE MOD7
SHARED COMMON.SEG,NAME=COMMONS,ACCESS=R,
CONTINUE>STRUCTURE=(DELTA/1000,GAMMA)
SHARED F7RTL.SEG
MAP PR1:,ADDRESS
BUILD MOD7
END
```

The INCLUDE command specifies that the object module MOD7.OBJ is to be included in the build.

The first SHARED command specifies that COMMON.SEG is the file containing a sharable segment called COMMONS which consists of the two common blocks, DELTA and GAMMA. The access privileges are read-only. Because a comma is the last character entered on the line, the CONTINUE> prompt is displayed in interactive mode and the remaining parameters are entered. The STRUCTURE parameter specifies that the first 1000 bytes of the segment COMMONS is to be allocated for the common block DELTA, regardless of the size of DELTA in the program. The area after the first 1000 bytes is to be allocated for the common block GAMMA. The parameters in the SHARED command are compared against the information in the file COMMON.SEG. Any information not provided by the parameters is taken from the file or defaulted. At run time, the pre-initialized segment is loaded from the file.

The second SHARED command specifies that another shared segment is to be loaded from the file F7RTL.SEG. All of the other parameters default to information contained in the file.

The MAP command specifies that a build summary and a listing of the names and locations of all modules and entry points in address order are to be generated.

The BUILD command builds the task image load module and stores it in the file MOD7.TSK. The sharable segments are referenced to resolve external references and to determine the placement of common blocks. The sharable segments are stored as separate files and are not included as part of the load module. The END command terminates the linkage editor.

4.7 BUILDING AN OPERATING SYSTEM IMAGE LOAD MODULE

This example includes an object module called MTSYSTEM.OBJ with external references produced by the Library Loader and builds an image load module. A map is to be generated listing the names and locations of all symbols, tasks, and entry points in alphabetical and address order.

```
ESTABLISH OS
INCLUDE MTSYSTEM.OBJ
MAP PR1:,ADDRESS,ALPHABETIC
BUILD OS32R0n.000
END
```

The ESTABLISH command specifies that the image load module to be built is to be an operating system load module.

The INCLUDE command specifies that the input file MTSYSTEM.OBJ contains the object module to be included in the build.

The MAP command specifies that a build summary and a listing of the names and locations of all modules and entry points in alphabetical and address order are to be generated.

The BUILD command builds the operating system image load module and stores it in the file OS32R0n.000 which can be loaded into main storage by the BOOT or LSU. The END command terminates the linkage editor.

APPENDIX A
LINK COMMAND SUMMARY

```
BFILE fd [,n]

BUILD fd

DCMD

END
          ⎧                                                                 ⎫
          ⎪ TASK                                                            ⎪
          ⎪ OS                                                              ⎪
          ⎪                         ⎧ E   ⎫                                 ⎪
ESTABLISH ⎨                         ⎪ R   ⎪                                 ⎬
          ⎪ SHARED [,ACCESS=        ⎨ RE  ⎬ ][,ADDRESS= ⎧ m0000 ⎫ ]         ⎪
          ⎪                         ⎪ RW  ⎪             ⎩   *   ⎭           ⎪
          ⎪                         ⎩ RWE ⎭                                 ⎪
          ⎪    [,NAME=segment]                                              ⎪
          ⎩                                                                 ⎭

EXTERNAL common block name1 [,...,common block namen]

FFILE fd [,n]

INCLUDE [fd] [[, ⎧ module1 ⎫]  [- ⎧ modulen ⎫],...,modulex]
                 ⎩    *    ⎭      ⎩    *    ⎭

LIBRARY fd1 [,...,fdn]

LOCAL entry point1 [,...,entry pointn]
```

```
LOG fd

                ┌ ┌ALPHABETIC┐ ┐
MAP [fd]        │,│ADDRESS   │ │
                └ └XREF      ┘ ┘

NDCMD

NLOG

OPTION  [ ETASK ] [,NAFPAUSE] [, RESIDENT ] [, SEGMENTED ]
        [ UTASK ] [ AFPAUSE ] [ NRESIDENT ] [ NSEGMENTED ]

        [,NROLL] [,COM ] [,CON ] [,NSVCPAUSE]
        [ ROLL ] [ NCOM] [ NCON] [ SVCPAUSE ]

        [,UNIVERSAL ] [,DISC ] [,ACP ] [,FLOAT ]
        [ NUNIVERSAL] [ NDISC] [ NACP] [ NFLOAT]

        [,DFLOAT ] [,LU=lu] [,SYSSPACE={ s  }]
        [ NDFLOAT]                     {3000}

        [,WORK=(( min ) ,( max   ))] [,ABSOLUTE={ a }] [,IOBLOCKS {b
               ((  *  ) ,(  *    ))]            {100}             {1
               (( 80  ) ,( 40000 ))]

        [,PRIORITY=([{ipri}] [,{mpri}])]
                   ([{128 }] [ {128 }])]

        [,TSW=([{status}] [,{st adr}])] [,ENTRY=entry point symbol]
              ([{  0   }] [ {  0   }])]

        [,TCOSAVE={NONE   }] [,XSVC1 ] [,VFC ]
                  {PARTIAL}  [ NXSVC1] [ NVFC]
                  {ALL    }

        [,INTERCEPT ] [,ACCOUNTING ] [,KEYCHECK ]
        [ NINTERCEPT] [ NACCOUNTING] [ NKEYCHECK]
```

```
                          ┌   ┌level┐ ┐
OVERLAY overlay name      │ , │     │ │
                          └   └  1  ┘ ┘


PAUSE


                        ┌ ┌[name1 ,...,namen]┐ ┐  ┌      ┌ nodename ┐ ┐
POSITION COMMON=        │ │                  │ │  │ ,TO= │          │ │
                        └ └        *         ┘ ┘  └      └ .ROOT    ┘ ┘

REWIND fd


SHARED     [fd]  [,NAME=segname]
                 ┌┌ R  ┐┐
                 ││ E  ││                  ┌m0000┐
       ,ACCESS=  ││ RE ││  [,ADDRESS=      │     │ ]
                 ││ RW ││                  └  *  ┘
                 └└ RWE┘┘
       [,STRUCTURE=(name1 [/size]   [,...,namen [/sizen]])]
       [,SIZE=([min   [,max]])]

TITLE title


VOLUME  [voln]


WFILE fd [,n]
```

APPENDIX B
LINK MESSAGE SUMMARY

ACCESS PRIVILEGE CONFLICT IN SHARABLE SEGMENT

Access privileges of a segment being referenced should be a higher privilege level than the access privileges specified when the segment was defined.

ADDRESS OVERFLOW AT xxxxxx

A halfword relocatable address was larger than 64kb.

ATTEMPT TO POSITION x IN A DIFFERENT PATH

An attempt was made to position a common block that is in a different path than the node referencing it.

ATTEMPT TO POSITION x IN LOWER LEVEL NODE

An attempt was made to reposition a common block program in a lower level node.

BUILD NOT SUPPORTED ON THIS DEVICE

Other than an indexed or contiguous file was specified for building the image.

CHECKSUM ERROR FILE: x MODULE: y RECORD: z

An invalid checksum was detected while reading an object file.

COMMAND NOT PERMITTED

Command is not valid for the type of build or not permitted in a DCMD statement.

COMMON BLOCK x ENCOUNTERED IN MORE THAN ONE SHARABLE SEGMENT

The same common block was specified in more than one SEGMENT command.

COMMON BLOCK x, NOT REFERENCED

The common block named was never referenced.

COMMON BLOCK x SPECIFIED IN POSITION COMMAND IS PART OF SHARABLE SEGMENT

An attempt was made to reposition a common block that was part of a sharable segment by using the POSITION command.

CONTINUATION NOT PERMITTED

An attempt was made to continue a command imbedded in the object code.

ENTRY POINT x SPECIFIED IN ENTRY OPTION NOT FOUND

The ENTRY parameter of the OPTION command specified a nonexistant entry point or an entry point in other than the root node.

ENTRY POINT x, SPECIFIED IN LOCAL COMAND, NOT DEFINED

The entry point named was never defined.

ESTABLISHMENT ABORTED

A serious error occurred, and the image module cannot be built. Link is cleared as if a module was built.

EXTERNAL REFERENCE TO OVERLAY CONTAINS OFFSET AT xxxxxx

An external reference with offset cannot be resolved because the corresponding entry point is an overlay.

EXTRA RIGHT PARENTHESIS

Either an extra right parenthesis or a missing left parenthesis condition occurred.

fd NOT FOUND

An assignment error occurred while Link attempted to assign the specified file.

INCORRECT PARAMETER LENGTH

The length of the value of an operand was longer or shorter than expected.

INSUFFICIENT WORK SPACE

There was not enough workspace for Link. It will return to command mode and clear itself as if an image had been built.

INVALID CHARACTERS IN NAME

Invalid characters in an entry point, common block, or overlay node name were encountered.

INVALID COMBINATION OF OPERANDS

A particular combination of operands was invalid.

INVALID COMMAND

An invalid command was specified.

INVALID DELIMITER

A delimiter that was unknown was found at the end of a parameter or where a parameter should have been.

INVALID FILE-DESCRIPTOR

A syntax error occurred in the fd entered.

INVALID KEYWORD

Misspelled keyword.

INVALID NUMERIC VALUE

A numeric value was expected but was not encountered.

INVALID PARAMETER

An invalid parameter was specified in a command.

INVALID POINTER TO LOCATION xxxxxx ENCOUNTERED IN
REFERENCE CHAIN FOR xxxxxx AT LOCATION xxxxxx
THIS INVALID POINTER ERROR OCCURRED IN
- FILE: vol:filename.ext/a - MODULE: module
- RECORD: number - BYTE number

LINK encountered an invalid link in an address chain. When LINK resolves a chain of references, it traces back through the chain, link by link, replacing the chain pointer with the resolved address of the object. If a chain has a forward pointer within a module or if a pointer indicates an area outside of the module, LINK ceases to follow this chain, leaving the remainder of the chain unresolved, and prints the error message above.

ITEM NOT PERMITTED IN E-TASK FILE: x MODULE: y RECORD: z
BYTE: m

The loader item encountered is not allowed in an e-task establishment.

MISSING PARAMETER

A required parameter was not specified.

MISSING RIGHT PARENTHESIS

A left parenthesis was encountered for which no matching right parenthesis was encountered.

MODULE INCOMPLETE FILE: x MODULE: y

An end-of-file condition was detected before the end-of-program item in an object module.

MODULE xxxxxxx NOT FOUND

A module specified in an INCLUDE command was not found.

MORE THAN 192 SEGMENTATION REGISTERS REQUIRED

More than 192 segmentation registers are required.

n MULTIPLY DEFINED SYMBOLS

Entry points were encountered that were defined more than once in the same path.

n UNDEFINED EXTERNAL SYMBOLS

This message is output at build time if any standard external symbols remain unresolved.

*** nnn UNDEFINED WEAK EXTERNAL SYMBOL(S)***

This message is output at build time if any weak external symbols remain unresolved.

name SPECIFIED IN POSITION COMMAND NOT FOUND

A common block that was specified on a POSITION command could not be found.

NUMERIC VALUE OUT OF RANGE

A numeric operand was greater than the maximum permissable value or less than the minimum permissable value.

OBJECT CODE ERROR (n) FILE: x MODULE: y RECORD: z BYTE m

An object code error cccurred. If n=1, an invalid object code item exists in object record. If n=2, the object code item overflows record. If n=3, a load program address item was expected but not encountered.

OVERLAY DEFINED OUT OF ORDER

An OVERLAY command specifies a level that is not consistent with the rules for defining overlays.

RECORD LENGTH FOR MAP LESS THAN 64 DEVICE/FILE

The device or file specified for the output of the maps has a record length less that 64 bytes.

SEGMENT AT x OVERLAPS NEXT SEGMENT

An impure, pure, or sharable segment's end address was greater than the end address of another segment. See the build summary map for the names of the segments.

SEQUENCE ERROR FILE x MODULE: y RECORD: z

A sequence number error was detected while reading an object module.

TOO MANY OPERANDS

More operands than were expected were encountered.

VIRTUAL SYMBOL TABLE SPACE LIMIT EXCEEDED

More than 256K symbol table space required.

WARNING: ABSOLUTE SPACE LESS THAN 100

Less than 100 bytes of absolute code was reserved for the UDL.

WARNING: MORE THAN 16 SEGMENTATION REGISTERS REQUIRED

More than 16 segmentation registers were used, making this image loadable only on a processor with greater than 1MB of memory.

WARNING: n AMBIGUOUS REFERENCES

External references were encountered that could be resolved to more than one entry point.

WARNING: OVERRIDE SIZE FOR COMMON BLOCK x SMALLER THAN ACTUAL SIZE

The override size specified in the STRUCTURE parameter of the SEGMENT command was smaller than the largest definition of the common block.

WARNING: NAME OF SHARABLE SEGMENT x DOES NOT MATCH NAME SPECIFIED IN SHARED COMMAND

The name given to a sharable segment when it was linked does not match the name specified in the NAME parameter of the SHARED command. The name specified in the SHARED command is used.

x ERROR (y) ON z TO fd

An SVC 7 error occurred. Variable x is the type of error, y is the hexadecimal status, z is the SVC 7, and fd is the file. See Table B-1 for the error types and status.

TABLE B-1 SVC 7 ERROR TYPES AND STATUS

FUNCTION z	ERROR TYPE x	HEX STATUS y	MEANING
ALLOCATE ASSIGN CLOSE DELETE FETCH ATTRIBUTES	VOLUME	3	Volume was not specified.
	DISC SPACE	5	Insufficient disc space available to allocate or assign a file.
	PROTECTION KEY	6	File being assigned had nonzero protection keys.
	ACCESS PRIVILEGE	7	Specified access privileges could not be granted.
	SYSTEM SPACE	8	Insufficient system space available.
	SVC 7	9-FF	An SVC 7 error occurred other than the errors specified above.

x ERROR (y) ON z TO LU n FILE fd

An SVC 1 error occurred. Variable x is the type of error, y is the hexadecimal status, z is the function that was being performed, and n is the lu number. See Table B-2 for the error types and status.

TABLE B-2 SVC 1 ERROR CODES AND STATUS

FUNCTION z	ERROR TYPE x	HEX STATUS y	MEANING
READ WRITE COMMAND	DEVICE UNAVAILABLE	A0	Device has been turned off.
	END OF MEDIUM	90	End of tape or disc encountered.
	END OF FILE	88	End of tape or disc encountered.
	UNRECOVERABLE	84	An unrecoverable error occurred.
	RECOVERABLE	82	A recoverable error occurred.

x IS NOT A SHARABLE SEGMENT

A file named in the SEGMENT command as a sharable segment was not a valid sharable segment.

APPENDIX C
LINK TO TET COMPARISON

This table compares Link to TET, the utility previously used to establish and build image load modules under OS/32. The Link commands are listed with the corresponding TET commands as a guide to converting from TET to Link.

TABLE C-1 LINK TO TET COMPARISON

LINK COMMANDS	TET COMMANDS	MEANINGS
BFILE		
BUILD	BUILD	*
DCMD		
END	END	*
ESTABLISH	ESTABLISH	*
EXTERNAL		
FFILE		
INCLUDE	INCLUDE	*
LIBRARY	EDIT	*
LOCAL		
LOG	LOG	*
MAP	MAP	Generates a map with symbols in address order
	AMAP	Generates a map with symbols in alphabetical order
NDCMD		
NLOG	NLOG	*

TABLE C-1 LINK TC TET COMPARISON (Continued)

LINK COMMANDS	TET COMMANDS	MEANINGS
OPTION	OPTIONS	Specifies task options
	MAXLU	Specifies maximum number of task's logical units
	MXSPACE	Sets maximum size of task's system space
	ABSOLUTE	Sets size of absolute memory to precede impure segment
	QIO	Sets maximum number of I/O proceed requests that can be queued by a task
	PRIORITY	Sets task priorities
	GET	Adds additional task memory
	EXPAND	Adds additional task memory
	TSW	Sets task's initial TSW
OVERLAY	OVERLAY	*
PAUSE	PAUSE	*
POSITION		
REWIND	REWIND	*
SHARED **	RESOLVE	Resolves external references to a sharable segment
	TCOM	Defines a task common segment
	LBLCOM	Defines a labeled common segment
TITLE	JOB	*
VOLUME	VOLUME	*
WFILE	WFILE	*
COMMAND parameter in START command	REMOTE	

* Indicates that the meanings for both Link and TET commands are the same.

** Link does not recognize previously generated TET shared segments. These shared segments must be reestablished using Link.

APPENDIX D
TET COMMANDS

D.1 INTRODUCTION

This chapter describes the OS/32 Task Establisher Task (TET). Any task, reentrant library, or preinitialized task common must be established using TET before it can run under OS/32. The functions of TET and the commands used to control it, are fully described in this manual. Examples are provided to show how TET is used to establish tasks in various environments. The user should refer to the OS/32 Application Level Programmer Reference Manual for detailed information about task preparation within an OS/32 environment.

A task may be a single program or a group of programs linked together. TET processes object code programs, links external references, and produces a memory image task for loading and running under OS/32. External references to task common and to previously established reentrant library segments are also processed.

When a task is established by executing TET, the result is one or more load modules of memory image code that can be loaded directly into memory using the OS/32 Resident Loader. The command stream directing TET activity can be input in batch mode or interactively. An operator uses the commands to specify programs for inclusion in the task, as well as task options. The establishment procedure requires two passes of the object code. On the first pass, TET compiles a symbol table of external references and definitions. On the second pass, the actual load module is built.

TET can also be used to build a memory image of OS/32 on disc. The operating system image produced by TET may be loaded into memory by the 32-bit Direct Access Bootstrap Loader, or by the Loader Storage Unit (LSU).

D.2 SYSTEM REQUIREMENTS

TET requires 25kb of memory space, plus approximately 2kb for dynamic operations, and as much space as is required to house a dictionary of all external references and definitions in the programs of the task being established. TET may build task modules in memory or use a contiguous disc file as work storage.

If the task is built in memory, there also must be enough space to hold the largest load module built. This workspace can be allocated at establishment time or load time. In either case, the amount of memory for workspace can be approximated as follows:

FUNCTION	MEMORY
For each entry in program	16 bytes
For each program definition	16 bytes
For each TET command entered	16 bytes
If image built in memory	256 bytes and largest load module
Plus	2kb

To allocate this amount of working storage at establishment time, the TET EXPAND and GET commands are used. Refer to Section D.4.7 for information. To allocate this amount of working storage at task load time, the segment size increment field of the LOAD command should be used. For example:

```
LOAD TET,TET32.TSK,10
```

Required devices include input and output binary devices for the input object code and output image code, an ASCII device for TET command input, and an ASCII print device for error warning and messages to prompt the operator. ASCII device requirements can be met by multiple assignments of a CRT or TTY, but high-speed devices are recommended for binary input and output. A temporary file is a recommended option to hold pass one input programs for use during pass two. If a temporary file is not used, the object code input file is processed twice. It is also recommended that the task be established on disc, because building in memory may require a very large memory segment.

D.3 AN ESTABLISHED TASK

An established task consists of at least a main (impure) segment made up of one or more object code programs. A task can also include a sharable (pure) segment, one or more task common segments, and one or more reentrant library segments. Certain run time task conditions can be established by TET. These include limits on the task's use of system space, the number of logical units (LUs) it may assign, its priority, and its initial task status. Most present options can be overridden once a task is loaded into memory. Those options that cannot be changed, except by reestablishment of the task, are noted in the command descriptions.

Task address space is divided into one or more program segments, sets of contiguous logical program addresses starting on a 64kb boundary. All program segments are classified according to their contents. That is, they are either pure or impure and task common or reentrant library.

An established task may consist of:

- One main (impure) segment (with optional user overlay)
- One sharable (pure) segment
- One or more task common segments (up to 15)
- One or more reentrant library segments (up to 15)

The inclusion of a pure segment, reentrant library segments, or task common segments is a user option depending on the task to be established. A maximum of 16 segments is available for task establishment. Table D-1 shows the relationship of the segment numbers to the start addresses.

TABLE D-1 ADDRESS-SEGMENT RELATIONSHIP

SEGMENT NUMBER	STARTING PROGRAM ADDRESS OF THE SEGMENT (HEXADECIMAL)
0	Y'00000'
1	Y'10000'
2	Y'20000'
3	Y'30000'
4	Y'40000'
5	Y'50000'
6	Y'60000'
7	Y'70000'
8	Y'80000'
9	Y'90000'
10	Y'A0000'
11	Y'B0000'
12	Y'C0000'
13	Y'D0000'
14	Y'E0000'
15	Y'F0000'

Two types of common are supported under OS/32: local common and task common. Local common is contained entirely within the main segment. It is referenced via EXTRNS by the main segment program as well as the subroutines of a task (e.g., FORTRAN common). TET allocates space for local common in the task's impure segment as defined by the included programs. Task common allows common references among OS/32 tasks and is symbolically referenced the same as local common. A particular common block is designated as a task common by the TET TCOM command.

Overlays established for a task use a single overlay area. The area required by the largest overlay is noted and made part of the impure segment. All overlays of a task are loaded at the same address in the segment. Overlays of a task may be built onto a single file or different overlay files. Overlays may not call one another; i.e., nested overlays are not permitted.

A TET load module consists of a loader information block (LIB) followed by the memory image of the task in 256-byte records. The LIB contains data required by the operating system to load the task in preparation for execution.

D.4 TET COMMANDS

These TET commands can be specified at task establishment time:

ABSOLUTE	MXSPACE
AMAP	NOLOG
BUILD	OPTIONS
EDIT	OVERLAY
END	PAUSE
ESTABLISH	PRIORITY
EXPAND	QIO
GET	REMOTE
INCLUDE	RESOLVE
JOB	REWIND
LBLCOM	TCOM
LOG	TSW
MAP	VOLUME
MAXLU	WFILE

D.4.1 ABSOLUTE Command

The ABSOLUTE command is optional. If this command is omitted, a 256-byte user dedicated location (UDL) area precedes the impure task segment. If the ABSOLUTE command is included, it specifies an absolute amount of memory which should precede the impure segment in place of the UDL.

Format:

```
ABSOLUTE xxxxx
```

Parameter:

xxxxx	is a hexadecimal number of up to five digits specifying the number of bytes of absolute-addressed code or absolute data to be included in the task.

Functional Details:

ABSCLUTE establishes a bias of the specified number of bytes between address 0 and the start of relocatable code. In the absence of this command, TET automatically reserves 256 bytes (X'100') for the UDL. If less than 256 bytes is specified, a warning message is printed, but TET biases the impure segment by the specified amount. A task's UDL may be deleted with an ABS 0 command. This command applies to task establishment only.

D.4.2 AMAP Command

The AMAP command requests a map with symbols in alphabetical order rather than address order. Otherwise, it is identical to the MAP command.

Format:

```
AMAP [fd]
```

Parameter:

fd	is the file descriptor of the file or device that the map is written to.

D.4.3 BUILD Command

The BUILD command indicates the end of pass one and the beginning of pass two. This command causes the load module to be built.

Format:

```
        ⎧ TASK [,fd]                          ⎫
        ⎪ RL   [,[fd] [,[sname] [,segno]]]    ⎪
BUILD  ⎨ CVLY [,fd]                          ⎬
        ⎪ TCOM [,[fd] [,sname]]               ⎪
        ⎩                                     ⎭
```

Parameters:

TASK	indicates a task load module is to be built. The output is to the specified fd. If the previous ESTABLISH command specified pure, two segments are built, one for a pure and one for an impure segment. If pure was not specified, then only one impure segment is built. If fd is not specified, LU 2 is assumed.
RL	causes one sharable reentrant library segment load module to be built, with output to the specified fd. RL load modules require segment names. The resident loader checks the name when the console operator loads a task to see if the library segment in memory is the one required by the task. All library segments should be resident. The RL segment name can be specified with the sname parameter, which is in this format: filename .ext. If sname is not specified, TET uses the filename.ext portion of fd.
segno	specifies the segment number to be allocated for this library segment. The default is segment 15. RL segments must contain only pure relocatable code and must not reference task common or another RL segment.

OVLY	causes an overlay module to be output to the specified fd. A BUILD OVLY command must be entered for, and correspond sequentially to, each OVERLAY command entered during pass one. Overlays can be built on one file or separate files, and are loaded by SVC 5 run time calls by the main segment.
TCOM	causes a task common segment to be built and output to the file. BUILD TCOM should be used when establishing an initialized task common segment with a block data program or absolute data program. See Section D.4.24.
sname	in the BUILD TCOM command specifies an override segment name. If sname is omitted when building a block data TCOM, the common definition name is selected as the segment name. If sname is omitted when building a nonblock data TCOM, the segment name is derived from the first included file containing a program label.

Functional Details:

The BUILD command reads the input program file in the same order as the first pass. If a temporary file is being used, the system rewinds it, if possible. Otherwise, the operator is prompted by a message to reposition the device. If a temporary file is not being used, the operator is prompted to load each program in sequence.

If the file specified by fd or assigned to LU 2 is a contiguous disc file, TET builds the load module directly on that file. If the file is not a contiguous disc file, TET builds in memory and outputs the load module to the file. If the form of the fd parameter is valid for a disc file, but the file does not exist, TET allocates and assigns a contiguous file with a size sufficient to hold the task. The output file should not be preallocated (or assigned) and TET should allocate and assign the output file.

D.4.4 EDIT Command

The EDIT command is used to edit a file after inclusion of one or more programs. EDIT includes all programs having program labels referenced by the already included code. This command can be used when a task requires a number of subroutines contained within a subroutine library.

Format:

```
    ED̲I̲T [fd]
```

Parameter:

fd	is the optional file descriptor from which additional programs are to be included. If fd is not specified, LU 1 is assumed.

D.4.5 END Command

The required END command terminates TET processing and returns control to the operating system. If the operator wishes to stop in the middle of an establishment process, and END command must be issued and TET reentered via the operating system START command to begin another establishment. Restarting TET is necessary to initialize TET pointers and work areas.

Format:

```
    END
```

D.4.6 ESTABLISH Command

The ESTABLISH command initializes TET task processing.

Format:

```
            ⎧ TASK  ⎡⎧,PURE   ⎫⎤ ⎫
            ⎪       ⎣⎩,IMPURE ⎭⎦ ⎪
ESTABLISH   ⎨ TCOM  ⎡⎧,NOBDATA⎫⎤ ⎬
            ⎪       ⎣⎩,BDATA  ⎭⎦ ⎪
            ⎩ RL                 ⎭
```

Parameters:

TASK TCOM RL	indicate whether a task, reentrant library or task common segment, respectively, is to be established. The defaults for TASK and TCOM are IMPURE and BDATA, respectively.
PURE	indicates that any code assembled as pure, via the CAL assembler pseudo-op PURE, is established in a separate segment. The pure segment can then be shared by several copies of the same task at task execution.
IMPURE	specifies that the pure and impure code are established in the same impure segment. The pure code of such an established task is not sharable. If no parameter is specified, IMPURE is the default.
NOBDATA	pertains only when establishing TCOM. It allows any absolute data (e.g., DS, DC, or DB types of statements), not a block data program, to be included as part of the TCOM segment.
BDATA	specifies that a block data program is to be established. If no parameter is specified, BDATA is the default.

Functional Details:

If a text editor is established with the code body as pure and data areas and buffers as impure, several editing tasks can be run simultaneously, each sharing the pure code body of the editor. The loader checks that the necessary pure segment is already in memory when the task is loaded. The parameter PURE is valid only when establishing a task, and is ignored for RL and TCOM.

The ESTABLISH TCOM command is used to separately establish a task common segment with initialized data, i.e., block data or absolute data. TET does not allow mixing block data with absolute data (nonblock data) in a single TCOM segment establishment. An initialized TCOM segment can contain either block data only or nonblock data only. The distinction is made via the BDATA/NOBDATA optional parameter. If a task common segment does not contain initialized data, it need not be established and need only be declared by the TCOM command when establishing the task.

TET allocates contiguous memory for the impure code starting at segment 0. If the impure code is greater than 64kb, TET allocates the next contiguous segment for the impure task area. TET allocates contiguous memory for the pure code, starting at the highest available segment, after all library and task common segments are resolved.

D.4.7 EXPAND and GET Commands

The EXPAND and GET commands are used to add memory to a task beyond what is required to hold the code body. For example, if a task requires an area of memory for processing, such as for a symbol table, GET and EXPAND move CTOP upward. They increase the minimum memory area in which a task can run.

Formats:

```
EXPAND xxxx

GET yyyyy
```

Parameters:

xxxx	is a 4-digit decimal number specifying the number of 256-byte blocks to be reserved beyond the end of the defined task.
yyyyy	is a hexadecimal number of up to five digits, specifying the number of bytes to be reserved.

Functional Details:

If neither of these commands is specified, and overlays are produced, a default value of X'300' bytes is assumed (the amount necessary for executing a FORTRAN program). If a task has no overlays, the default allocation is 0.

The GET command can be given with a parameter of 0 to save space if no GET STORAGE calls are to be issued. The total number of bytes specified by GET and EXPAND is rounded upward to a 256-byte boundary. The GET and EXPAND commands can appear anywhere between the ESTABLISH and BUILD commands.

D.4.8 INCLUDE Command

The INCLUDE command is required to specify that program(s) from a file are to be included into a task (reentrant library or task common).

Format:

```
INCLUDE [fd] [,program label]
```

Parameters:

fd	is the optional file descriptor of the input device. If fd is not specified, LU 1 is assumed, in which case the fd should be preassigned to LU 1.
progam label	specifies the program name in the input file and causes this program to be located and included. If a progam label is not specified, the entire file is included up to end-of-medium or end-of-file.

Functional Details:

As the input file is read, TET creates a dictionary of external program references. A copy of the included program(s) is written to the temporary file, if present.

D.4.9 JOB Command

The JOB command allows the operator to title the TET output map. This command is permitted any time during the execution of TET.

Format:

```
    JOB title
```

Parameter:

title	is the title given to the TET map. Any characters are permitted in this 1- to 12-character field. All characters beyond the twelfth position are ignored.

D.4.10 LBLCOM Command

The LBLCOM command can be used to structure a task common segment for more than one named common block. The LBLCOM command causes TET to construct references to a task common segment for each named common that appears in a LBLCOM command and is referenced in the task. The LBLCOM command must be used in conjunction with a TCOM command. A TCOM command must be entered first for every task common segment; one or more LBLCOM commands may be entered with the same segment number.

Format:

$$\underline{LBL}COM\quad name_1\ /segno_1 \left[/ \begin{Bmatrix} \underline{RW} \\ \underline{RO} \end{Bmatrix} \left[/size_1 \right] \right]$$

$$\left[,name_n /segno_n \left[/ \begin{Bmatrix} \underline{RW} \\ \underline{RO} \end{Bmatrix} \left[/size_n \right] \right] \right]$$

Parameters:

name	specifies a 1- to 8-character name of a named common.
segno	is the segment number of the task common. If the segno is not the same as the segno in the TCOM command for that segment, the LBLCOM command is rejected. The message ILLEGAL SEGMENT NO is printed. If command input is not remote, TET accepts a LBLCOM command with the correct segment number.
RW and RO	specify the access privileges, read/write and read only, respectively. If RW and RO are omitted, the default is RW. The access privileges must be the same as specified in the TCOM command for that segment. Any conflict in access privileges causes the warning message ACCESS PRIVILEGE CONFLICT to be printed. The access privileges are set equal to those entered with the TCOM command for that segment. TET is then ready to accept the next command.

size	is a decimal number specifying the number of 256-byte blocks. If size is omitted or specified as 0, the common block's size is as defined within the task. The first task definition of the common block determines the size. The size of any subsequent task definition of the same common block must be less than or equal to the size of the first definition; otherwise, a TCOM TOO BIG is generated and task establishment is aborted. A specified size overrides the actual size as encountered in the common definition during task establishment. This size must be greater than or equal to any common definition for that named common in the task; otherwise, a TCOM TOO BIG is generated and task establishment is aborted.

Functional Details:

The name given to the task common segment is the name of the first named common block encountered in the task that matches the name field in a TCOM or LBLCOM command for that segment. The amount of memory associated with a task common segment at task load time is the sum of the common block sizes for that segment. If a TCOM or LBLCOM command specifies an override size for the first named common block that is encountered while building the task, the segment size is considered to be an override size by the system loader.

D.4.11 LOG Command

The LOG command causes all operator commands to be copied to a specified output unit. This command is generally used when the REMCTE command is used.

Format:

```
    LOG [fd]
```

Parameter:

fd	specifies the unit to contain the commands.

D.4.12 MAP Command

The MAP command outputs a display of the symbol table that TET has built during processing of the program. If the command is entered during pass one, the user may obtain a list of undefined symbols to determine which files are yet required. However, MAP is most useful at the end of pass two when the establishment is complete. The items are output in address order.

Format:

```
MAP [fd]
```

Parameter:

fd	is the device on which to display the contents of the dictionary. If fd is not specified, LU 3 is the default.

Functional Details:

The following list is a description of each item in a TET map. Individual headings are not printed unless there is an item to be printed under that heading. Sample maps are included in Appendix G.

CTOP	A hexadecimal value representing the last halfword location in the user's required memory space. This value is always the last halfword in a 256-byte block, i.e., xxxxFE.
UTOP	A hexadecimal value representing the first fullword location above the user's established task. It is, in effect, the next available location in the user's space.
MIN CORE SIZE	A decimal value representing the minimum memory size in kilobytes required by this segment; e.g., 4.00kb=X'1000' bytes (impure segment size in a task plus expand and get storage).

PROGRAM SEGMENTS	All program segments defined or referenced by this task are listed. For each segment, the segment name and size are listed. During pass two, the segment number is also listed for each segment. This section of the map does not appear when establishing reentrant libraries or task common segments.
PROGRAM LABELS	Entries in this list are 6-digit hexadecimal addresses followed by the corresponding 8-character program label. Each label represents a program label and its program address.
TASK ENTRY POINTS	Entries in this list are 6-digit hexadecimal addresses and their associated symbolic names of all symbols processed within the established program.
LOCAL COMMON BLOCKS	This section lists the components of local common area of the user task. An entry is a 6-digit hexadecimal address field followed by the local common symbol name.
UNDEF-SYM FULLWORD	A list of all fullword external references for which no definitions have been encountered.
UNDEF-SYM HALFWORD	A list of all halfword external references for which no definitions have been encountered.
LIBRARY ENTRIES	A list of all resolved references. The address fields in this list reflect referenced reentrant library segments.
TASK COMMON BLOCKS	A list of all task common blocks and their addresses.
OVERLAY	Each overlay of a task is listed on a separate page after the root has been mapped. The name of the overlay is followed by all entry points and undefined symbols contained in the overlay.

D.4.13 MAXLU Command

The optional MAXLU command specifies the maximum number of LUs that a task can assign.

Format:

```
MAXLU  lu
```

Parameter:

lu	is a decimal number between 0 and 254 specifying the number of LUs.

Functional Details:

If this command is omitted, the default value is 15. Note that when taking the default or specifying 14 LUs, only 0 through 14 are available to the task. The command may be entered anywhere in pass one. The value of MAXLU determines the size of the task's LU table, a dynamic system data structure defined at task load time. In memory bound situations, the value of MAXLU should be as small as possible to avoid wasting space. Four bytes of memcry are required for each LU in the table.

D.4.14 MXSPACE Command

The optional MXSPACE command sets a limit on the amount of system space that a run time task can use for dynamic system data structures (file control block, etc.) during execution.

Format:

```
    MXSPACE  xxxxx
```

Parameter:

xxxxx	is a hexadecimal number of up to five digits specifying the number of bytes of system space.

Functional Details:

A default assumption of 12kb is made if the MXSPACE command is omitted. This command can be entered anywhere between ESTABLISH and BUILD commands.

D.4.15 NOLOG Command

The NOLOG command halts the LOG command operation.

Format:

$\underline{\text{NOL}}\text{OG}$

D.4.16 OPTIONS Command

The OPTIONS command specifies one or more of the options associated with a task.

Format:

```
         ┌ ┌ET┐ ┐ ┌  ┌AFCONT ┐ ┐ ┌  ┌ RESIDENT    ┐ ┐ ┌  ┌SVCCONT ┐ ┐
OPTIONS  │ │  │ │ │ ,│       │ │ │ ,│             │ │ │ ,│        │ │
         └ └UT┘ ┘ └  └AFPAUSE┘ ┘ └  └NONRESIDENT  ┘ ┘ └  └SVCPAUSE┘ ┘

         ┌  ┌ ROLL  ┐ ┐ ┌  ┌ COM  ┐ ┐ ┌  ┌ CON  ┐ ┐ ┌  ┌ UNIVERSAL   ┐ ┐
         │ ,│       │ │ │ ,│      │ │ │ ,│      │ │ │ ,│             │ │
         └  └NOROLL ┘ ┘ └  └NOCOM ┘ ┘ └  └NOCON ┘ ┘ └  └NONUNIVERSAL┘ ┘

         ┌  ┌FLOAT  ┐ ┐
         │  │       │ │ ┌  ┌NOACCOUNT┐ ┐ ┌  ┌ ACP  ┐ ┐ ┌  ┌ DISC  ┐ ┐
         │ ,│DFLOAT │ │ │ ,│         │ │ │ ,│      │ │ │ ,│       │ │
         │  │       │ │ └  └ ACCOUNT ┘ ┘ └  └NOACP ┘ ┘ └  └NODISC ┘ ┘
         └  └NOFLOAT┘ ┘
```

Parameters:

UT	Normal user task
ET	An executive task (E-task) that can execute privileged instructions. An E-task must contain only positionally independent code (RX2 instruction for memory reference) and cannot reference reentrant library or task common segments. The ET option conflicts with the PURE and ROLL options.
AFPAUSE	If an arithmetic fault occurs during task execution, the task is to be paused.
AFCONTINUE	If an arithmetic fault occurs during task execution, the task is to continue. If a task is to take arithmetic fault traps, it must be established with the AFCONTINUE option.
NONRESIDENT	At end-of-task (EOT), the task is deleted from memory.
RESIDENT	At EOT, the task remains in memory. A resident task cannot be a candidate for roll.
SVCPAUSE	If an SVC 6 execution is attempted, the task

	should be paused. This option applies only to the background segment.
SVCCONTINUE	If an SVC 6 execution is attempted, the call is ignored and task execution continues. This option applies only to the background segment.
ROLL	The task is a candidate for a roll-out/roll-in operation during its execution. If a task of higher priority requires the memory occupied by this task, it can be written (rolled-out) to a direct-access device and its execution suspended until sufficient memory becomes available. The ROLL option conflicts with the ET and RES options.
NOROLL	The task is not a candidate for a roll-out/ roll-in operation.
COM	This task can issue SVC 6 intertask communication calls (send message, queue parameter).
NOCOM	The task cannot issue SVC 6 intertask communication calls (send message, queue parameter).
CON	The task can issue SVC 6 intertask control calls (all SVC 6 functions except send message, queue parameter).
NOCON	The task cannot issue SVC 6 intertask control calls (all SVC 6 functions except send message, queue parameter).
UNIVERSAL	Specifies that the task has the privilege of communicating with all other tasks in the system. In a system containing the multi-terminal monitor (MTM), intertask communication is not permitted between the foreground and the terminal environment. However, a task that is established (using TET) as a universal task can be loaded into the foreground and can communicate with the terminal environment, using SVC 6 queue parameter and send message requests.
NONUNIVERSAL	Communications options are not universally allowed.
FLOAT	Specifies that a task can execute single-precision floating point instructions.
DFLOAT	Specifies that a task can execute double-precision floating point instructions.

NOFLOAT	Specifies that a task cannot execute any floating point instructions. If the FLOAT, DFLOAT, and NOFLOAT parameters are omitted, NOFLOAT is the default.
ACCOUNTING	Specifies that the accounting function is enabled for a task.
NACCOUNTING	Specifies that the accounting function is disabled for a task.
ACP	Specifies that a user task has extended file access privileges and can specify an account number instead of a file class for all SVC 7 functions.
NOACP	Specifies that a user task has no extended file access privileges. If both access privilege parameters are omited, NOACP is the default.
DISC	Specifies that a user task has an extended disc privilege and can assign to a bare disc file. If the disc is on-line, assignments for SRO are allowed. All other assignments are rejected and a message is displayed. If the disc is marked off-line, all access privileges are allowed. See the OS/32 Programmer Reference Manual for a description of the access privileges.
NODISC	Specifies that a user task has no extended disc privileges. If both disc privilege parameters are omitted, NODISC is the default.

Functional Details:

This command is optional, but if entered, it must follow the ESTABLISH command and precede the INCLUDE command. The OPTIONS command is not valid when establishing a reentrant library or task common. The option information is placed in the task LIB, and eventually, into the task control block (TCB) at run time.

Refer to the OS/32 Programmer Reference Manual for a more detailed description of each option. If two conflicting options are specified in one OPTIONS command, e.g., OPTIONS ROLL,ET, the entire OPTIONS command is rejected. If a successive OPTIONS command is in conflict with a preceding OPTIONS command, one of the following occurs:

- The second command takes precedence. This occurs if the second command is the direct opposite of the previous command. For example:

```
OPT    AFP
OPT    AFC
```

No error is generated. The arithmetic fault CONTINUE command takes precedence.

- The second command results in an error. This occurs if the second command is inconsistent with the previous command. For example:

```
OPT    RES
OPT    ROLL
```

The second command is rejected with an error. The task is not rollable.

The OS/32 operator OPTIONS command may be used to change certain of these options. Refer to Chapter 3 for details.

D.4.17 OVERLAY Command

The OVERLAY command is used to indicate that an overlay is to be included in a task being established.

Format:

```
    OVERLAY  name
```

Parameter:

name	is the name of the overlay. This name must be from one to eight alphanumeric characters, with the first character alphabetic.

Functional Details:

TET interprets this statement as ending the definition of a main segment (or previous overlay), and starting the definition of an overlay. Each overlay must be completely defined (with INCLUDE and EDIT statements) before another OVERLAY statement is presented in the command stream. After all overlays are defined, the overlay area is set to the size of the largest area requested, starting at the end of the main segment.

Only one overlay area is reserved in the task's impure segment, no matter how many OVERLAY commands are entered. The OVERLAY command must precede the INCLUDE and EDIT commands that define its contents, and these must precede any other OVERLAY statement or the BUILD command.

D.4.18 PAUSE Command

The PAUSE command temporarily suspends TET operations and returns control to the operating system. The operating system CONTINUE command is used to return control to TET.

Format:

```
    PAUSE
```

D.4.19 PRIORITY Command

The optional PRIORITY command sets the initial and maximum pricrities for the task at run time. This command may be entered during pass one, after the ESTABLISH command.

Format:

```
PRIORITY  ip,mp
```

Parameters:

ip	is a decimal number between 10 and 249 indicating the initial priority of the task.
mp	is a decimal number between 10 and 249 indicating the maximum priority of the task.

Functional Details:

The number specified as mp must be less than or equal to that specified as ip. If mp is not less than or equal to ip, the command is rejected. If this command is not specified the default value of 128 is assumed for both parameters.

D.4.20 QIO Command

The QIO command allows the user to specify the maximum number of proceed I/O requests that may be enqueued by a task.

Format:

```
    QIO n
```

Parameter:

n	is an integer from 0 through 65,535.

Functional Details:

If this command is omitted, the value of n defaults to 0, and proceed I/O requests are processed as in OS/32 R03-01. Tasks established using R03-02 of TET assume a 0 value for n. When n is set to 0, proceed requests behave exactly as in Release 03-01 of OS/32, e.g., a proceed I/O request for an LU with an outstanding request causes the task to enter a wait state until the first request is complete, unless unconditional proceed is specified, in which case the request is rejected. The number of I/O requests a task may issue before the queue is saturated is equal to n plus the number of LU's with active I/O ongoing.

D.4.21 REMOTE Command

The REMOTE command is used during batch mode (CSS input), to instruct TET to abort processing if an error is detected, as no operator is present to reposition, rewind, etc.

Format:

```
    REMOTE
```

Functional Details:

This command can be issued at any point in the sequence, and takes effect immediately. Once entered, TET executes to completion in this mode. To return to an interactive mode, TET must be terminated and restarted.

In this mode, the use of a temporary file is recommended. Refer to Section D.6.2 for further information.

D.4.22 RESOLVE Command

The RESOLVE command is used to resolve external references to a previously established reentrant library segment (such as FORTRAN run time library).

Format:

```
    RESOLVE [fd]
```

Parameter:

fd	is the optional file descriptor on which the reentrant library load module is to be found. The default is to LU 1.

Functional Details:

This command is used after the program referring to the reentrant library (RL) has been included. The RESOLVE command resolves all references to programs found in the reentrant library load module. The task's LIB specifies the reentrant library that is required for the task to run. The task aborts during loading if its required reentrant library is not present.

D.4.23 REWIND Command

The REWIND command assigns a file to LU 1 and rewinds the file.

Format:

```
    REWIND [fd]
```

Parameter:

fd	is the optional file descriptor of the file to be assigned to LU 1 and rewound. If fd is not specified, LU 1 is rewound.

Functional Details:

The REWIND command is used preceding an EDIT command so that the entire file can be edited. In addition, REWIND is used to ready a file for pass two, if no temporary file is used.

D.4.24 TCCM Command

The TCOM command causes TET to construct references to a task common segment for each named common that appears in a TCOM command, and is referenced in the task.

Format:

$$\underline{\text{TC}}\text{OM}\quad \text{name}_1\text{/segno}_1 \left[/ \begin{Bmatrix} \underline{\text{RW}} \\ \underline{\text{RO}} \end{Bmatrix} \; \left[\text{/size}_1 \right] \right]$$

$$\left[\text{,name}_n\text{/segno}_n \left[/ \begin{Bmatrix} \underline{\text{RW}} \\ \underline{\text{RO}} \end{Bmatrix} \; \left[\text{/size}_n \right] \right] \right]$$

Parameters:

name	specifies a 1- to 8-character name of a named common.
segno	is the segment number of the task common, specified as a decimal number from 1 through 15. This field causes TET to position the task common in the appropriate area of the task's address space.
RW and RO	specify the access privileges read/write and read only respectively. If RW and RO are omitted, the default is RW.
size	is a decimal number specifying the number of 256-byte blocks. This size overrides the actual size as encountered in the common definition during task establishment. This size must be greater than that given by the common definition, otherwise TET fails to establish the task. If this parameter is omitted or specified as 0, TET takes the size from the common definition.

Functional Details:

Any named common definition in a task is potentially a task common. Any named common which does not appear in a TCOM or LBLCOM command is considered to be local common, and is included in the impure segment. See Section D.4.10 for a description of the LBLCOM command. A single TCOM command may describe several task commons, or several TCOM commands may be entered, each describing one or more task commons. All TCOM commands must be entered prior to any INCLUDE commands, and after the ESTABLISH command.

If a TCOM and LBLCOM command specifies an override size for the first named common block that is encountered while building the task, the task common segment size is considered to be an override size by the system loader. If no TCOM command is given, but TSKCOM is referenced by the task, TET assumes:

```
    TCOM TSKCOM/14/RW
```

However, if a TCOM command is given, TET does not recognize the name TSKCOM unless it appears in a TCOM command.

NOTE

The name symbol, generated by the FORTRAN V compiler for a common block, consists of the user-specified name with a period (.) appended. To provide a compatible linkage for the CAL user, TET ignores the period generated by FORTRAN. Specifically, TET removes the period from all common block names (both local and task common) that contain one. Therefore, the name specified in the TCOM command should not contain a final period, and a CAL program should not contain two common blocks whose names differ only by a final period.

D.4.25 TSW Command

The TSW command specifies initial setting of the task's run time task status word (TSW), and optionally provides a start address for the task. The TSW defines trap conditions for which the task is responsible.

Format:

```
TSW status [,start address]
```

Parameters:

status	is an 8-digit hexadecimal number indicating the setting for the TSW.
start address	is an optional 6-digit hexadecimal number indicating the starting address for the task.

Functional Details:

If the start address parameter is omitted, the last transfer address found in the included code is used as the starting address. If no transfer address is found, X'100' (the file location immediately following the UDL) is assumed. If a transfer address is found within the overlay, it is ignored.

For a detailed description of the TSW see the OS/32 Programmer Reference Manual.

D.4.26 VOLUME Command

The VOLUME command assigns an override of the system default volume used by TET during its execution.

Format:

```
    VOLUME voln
```

Parameter:

voln	is any legitimate volume name of up to four characters. Whenever a file descriptor that does not specify a volume name is encountered, voln is used.

Functional Details:

The VOLUME command can be entered more than once, if desired, and takes effect immediately. It can appear anywhere in the sequence. Refer to Chapter 3 for a description of the operator command VOLUME.

D.4.27 WFILE Command

The WFILE command assigns a file descriptor to LU 2 and writes a filemark to the file. This command is generally used to separate the main segment from the overlays when they are output on magnetic tape or the same contiguous file.

Format:

```
WFILE [fd]
```

Parameter:

fd	is the file descriptor to be assigned to LU 2. If fd is not specified, a filemark is written to LU 2, which is left positioned past the filemark.

D.5 OPERATING PROCEDURES

TET is run as an established task under OS/32 and is executed by use of LOAD and START commands.

D.5.1 Logical Unit Assignments

Six LUs are used by TET. Table D-2 contains information relating to each unit. LUs 1, 2, and 3 can be assigned by TET command parameters or by the operating system ASSIGN command prior to starting TET. LUs 5 and 7 must be assigned before starting TET. If a temporary file is used, it also must be assigned to LU 4 before starting. A TEMP file can also be used as a temporary file. TEMP files are automatically deleted when they are closed at EOT.

TABLE D-2 TET LOGICAL UNIT ASSIGNMENTS

LOGICAL UNIT	DATA TYPE	USE	DEVICE EXAMPLES	LOGICAL RECORD LENGTH
1	Binary	Object-code input, Image RL input	Paper tape, Mag tape, Disc	Object 126 Image 256
2	Binary	Image load module output	Paper tape, Mag tape, Disc	256
3	ASCII	Memory map output, Command logging	Console, line printer, VDU	Variable, up to 120
4	Binary	Temporary file (optional)	Rewindable device, i.e., mag tape, disc, etc.	126
5	ASCII	Command input	Card reader, Console, VDU	80
7	ASCII	Error messages, warnings, prompts to operator	Console, line printer, VDU	Variable, up to 80

D.5.2 Temporary File Operation

The use of a temporary file is a recommended option as it minimizes the possibility of errors and allows rapid establishment. Because TET uses two passes of the command stream, the temporary file contains the programs input during pass one for use during pass two. If a temporary file is not used and the input file(s) is not a direct-access file, the operator must reload each of the input files for the BUILD process during pass two. If a temporary file is not used, the following message is output to prompt the operator to load programs:

```
LOAD PROGRAMS fd
```

and the program pauses. Programs included from a magnetic tape, paper tape, or cassette must be repositioned. TET should then be resumed by the operator command CONTINUE.

If a temporary file is used, it is rewound at the beginning of pass one. At the end of pass one, a filemark is written to the temporary file and it is rewound for pass two. If a nonrewindable device is used, the operator is prompted via a message to reposition the temporary file. At EOT, the temporary file is rewound, which allows one temporary device to be used for successive TET runs without operator intervention.

NOTE

> If TET is paused during pass two, a DISPLAY LU command entered at the system console reflects the TET reassignment of LUs.

D.5.3 Command Input Sequence

Certain LU considerations constrain the TET command sequence. The commands for establishing a task are input from LU D. If LU 5 is assigned to an interactive device, TET executes in the interactive mode. Otherwise, TET executes in a batch mode. The input mode determines the response to error conditions. TET error messages are summarized in Appendix 4.

Table D-3 lists the TET commands in recommended order and the commands permitted when performing a particular operation. Task establishment is begun by the ESTABLISH command with a parameter specifying TASK, RL, or TCOM. This command must precede all other commands except REMOTE, JOB, LOG, NOLOG, VOLUME, and MAP, which can appear at any point in the stream. The REMOTE command prevents TET from pausing in batch mode. This facilitates batch processing with minimal operator intervention.

Particular operations are:

TA	Building a task
RL	Building a reentrant library
TC	Building a task common
OV	Building an overlay

All commands are optional except ESTABLISH, INCLUDE, BUILD, and END.

TABLE D-3 LOGICAL TET COMMAND SEQUENCE

COMMAND VERB	SEQUENCE CONSIDERATIONS	TA	RL	TC	OV
REMOTE	Anywhere	*	*	*	*
JOB	Anywhere, normally before MAP and AMAP	*	*	*	*
LOG	Anywhere	*	*	*	*
NCLOG	Anywhere	*	*	*	*
VOLUME	Anywhere	*	*	*	*
ESTABLISH	Must precede all commands, except those which may be specified anywhere	*	*	*	*
QIO	Between ESTABLISH and BUILD	*			
OPTIONS	Between ESTABLISH and first INCLUDE	*			
ABSOLUTE	Must precede INCLUDE	*			
TCOM	Between ESTABLISH and first INCLUDE	*			
LBLCOM	Between the associated TCOM and BUILD	*			
INCLUDE	Between ESTABLISH and BUILD or between OVERLAY and BUILD	*	*	*	*
REWIND	Anywhere	*	*	*	*
EDIT	Must follow at least one INCLUDE	*	*		*
GET	Between ESTABLISH and BUILD	*			
OVERLAY	After main segment definition; must be followed by INCLUDE or EDIT	*			*
RESOLVE	After INCLUDE or EDIT	*			*
EXPAND	Between ESTABLISH and BUILD	*			
MXSPACE	Between ESTABLISH and first BUILD/OVERLAY	*			

TABLE D-3 LOGICAL TET COMMAND SEQUENCE (Continued)

COMMAND VERB	SEQUENCE CONSIDERATIONS	TA	RL	TC	OV
MAXLU	Between ESTABLISH and first BUILD/OVERLAY	*			
PRIORITY	Between ESTABLISH and BUILD	*			
TSW	Between ESTABLISH and BUILD	*			
BUILD	Must follow all INCLUDE, EDIT and OVERLAY	*	*	*	*
PAUSE	Anywhere	*	*	*	*
WFILE	Anywhere, normally after BUILD	*	*	*	*
AMAP	Anywhere, normally after BUILD	*	*	*	*
MAP	Anywhere, normally after BUILD	*	*	*	*
END	Anywhere, normally last	*	*	*	*

NOTES

1. The BUILD command marks the beginning of pass two for building the load module. This command is input only once for each load module. Tasks with overlays should contain one BUILD command for each corresponding overlay.

2. The ESTABLISH command marks the beginning of pass one. This command can be input only once during a TET session.

3. The OVERLAY command terminates the definition of the task's segments (impure and pure) during pass one and marks the beginning of an overlay definition during pass one. This command is input once for each overlay.

TET defines the task on the first pass and constructs the task load module on the second pass. Therefore, before including any task code, TET must know if the task is an E-task or if absolute address space is to be reserved. As a result, the OPTIONS and ABSOLUTE commands must be entered before any INCLUDE command.

The distinction between named common and task common is made by the TCOM command. References to named common blocks, designated as task common by a TCOM command, are relocated to the proper program address. The TCOM command must be entered before any INCLUDE command. The task's program contents must then be processed. INCLUDE and EDIT commands are used to select the input object code programs that are part of the task, task common or library. One INCLUDE statement must occur first, to bring in a single relocatable program or an entire file. Any number of INCLUDE statements may follow. If a single program is included, the input read operation stops at the end of that program allowing the next inclusion from the same file or another file.

The EDIT statement reads the entire edit file, and marks for inclusion any programs that have program labels referenced by the already included code. If a temporary file is present, the entire edit file is copied to it, but only the required programs are used during the build operation. EDIT statements can be repeated, allowing an EDIT to bring in programs referenced in code included by a previous EDIT. The user can select programs from multiple files by varying the fd parameters of successive EDIT and INCLUDE statements. See examples of this in Section D.7.

TET resolves all references from one included program to another in a task's included code, but references to a reentrant library are unresolved until the input of a RESOLVE command. The RL referenced must be previously established. The RL being resolved need not be the one currently loaded in the system under which TET is running. TET reads the LIB of the RL load module, and finds the ENTRY symbols that are the same as the referencing EXTRNs in the task. A message indicating that the unresolved labels exist is output at the beginning of pass two, in addition to a table (via the MAP or AMAP commands) which lists these symbols.

Overlays are indicated and named by OVERLAY statements, and defined by INCLUDE and EDIT statements, after their main segments have been completely defined (by INCLUDE and EDIT statements). Space required beyond the area where the task is loaded, for an expanding operation such as a symbol table or SVC get storage calls, can be reserved with a GET or EXPAND command. GET or EXPAND can be entered at any point. The GET or EXPAND area is located above the overlay area within the task's memory space. After an ESTABLISH TA command within TET's first pass, the MXSPACE, MAXLU, PRIORITY, and TSW commands can also be entered.

The first pass ends and the second pass starts when a BUILD command is encountered. Any unresolved references remaining are reported by a message to the operator. A BUILD command creates an image load module for a task, overlay, task common block, or reentrant library using the resolutions and specifications of the first pass. Each overlay module of a task requires a separate BUILD command. The RESOLVE command satisfies all references from included programs to previously established reentrant library segments. A message indicating that unresolved labels exist is output at the beginning of pass two. A RESOLVE command may be issued for as many library segments as are referenced by the included program.

D.6 AUTOMATIC ASSIGNMENT

Specifying a file descriptor in certain TET commands automatically causes the file to be assigned to one of the TET LUs. Table D-4 summarizes the action of the specific commands.

TABLE D-4 AUTOMATIC ASSIGNMENT OF FILE

COMMAND	FD ASSIGNED TO	DEFAULT EXTENSION
INCLUDE	LU 1	OBJ
EDIT	LU 1	OBJ
RESOLVE	LU 1	RTL
BUILD TASK	LU 2	TSK
BUILD RL	LU 2	RTL
BUILD OVLY	LU 2	OVY
BUILD TCOM	LU 2	TCM
WFILE	LU 2	NONE
MAP	LU 3	NONE
AMAP	LU 3	NONE
LOG	LU 3	NONE
REWIND	LU 1	NONE

Various parts of the fd, or the entire fd, can be omitted by the user. Typically, if fd is omitted, TET uses the default LU that should be assigned for that function until a reassignment occurs. For example, assume the user has assigned LU 1 to the paper tape reader/punch (PTRP:) before starting TET:

```
    ES TASK
    IN
    IN
    IN
    IN MAG1:
    IN
    IN
    IN PTRP:
    IN FILEB
    IN FILEA, PROGA
    BU TA, TETOUT
```

When TET finds no file descriptor in the first INCLUDE command, it uses the device already assigned to LU 1. If no device is assigned, TET outputs the message LU 1 NOT ASSIGNED. In this case, the paper tape unit is assigned to LU 1, and TET reads data from LU 1. The paper tape unit is used for each IN command until the IN MAG1: command, where TET closes LU 1 and then assigns the mag tape unit to LU 1. Data is read from the mag tape for each INCLUDE command (the IN PTRP:), where TET switches LU 1 to the paper tape unit again.

If the user does not specify the volume name, and if the VOLUME command has been issued, TET substitutes the volume name specified in the VOLUME command. If the VOLUME command has not been issued, as in this example, the default system volume is used. If the user omits the extension field, TET supplies a default extension. The default is selected according to the command being executed and the data being processed. Table D-4 lists the default extensions. If the fd specified in the BUILD command is a disc file, TET allocates a contiguous file of proper size to build the load module. If a file with the same name already exists, that file is not deleted.

D.7 EXAMPLES OF TET OPERATION

The following examples show the command sequence used to establish tasks under a variety of conditions. In all these examples, use of a temporary file is assumed (LU 4 is assigned prior to starting TET).

D.7.1 Establishing a Simple Task

This example illustrates how to establish a simple task. FILEA contains the object program(s).

```
    ESTABLISH TASK
    INCLUDE FILEA
    BUILD TASK, FILEA.TSK
    MAP                        Produces a map of the task
    END                        Returns control to the system
```

D.7.2 Establishing a Task with Pure and Impure Segments

This example illustrates how to establish a task from a file that contains programs (segmented in pure and impure form), some of which are not required by this task. Programs A, C, and D are to be included in the task. Figure D-1 is a description of the input and output files.

```
 -----------                                    -----------
| PROGRAM H |                                  | PROGRAM H |
|-----------|                                  |-----------|
| PROGRAM G |                                  | PROGRAM F |
|-----------|                                  |-----------|
| PROGRAM F |                                  | PROGRAM D |
|-----------|                                  |-----------|
| PROGRAM E |                                  | PROGRAM C |
|-----------|                                  |-----------|
| PROGRAM D |                                  | PROGRAM A |
|-----------|  CONTAINS EXTERNAL                 -----------
| PROGRAM C |  REFERENCES TO
|-----------|  PROGRAMS A,D, & H
| PROGRAM B |
|-----------|  CONTAINS EXTERNAL
| PROGRAM A |  REFERENCES TO
 -----------   PROGRAM F
                                                LOAD MODULE OF
                                                ESTABLISHED TASK
  FILE A                                        (OUTPUT FILE IMAGE
  (INPUT FILE OBJECT CODE)                      CODE)
```

Figure D-1 Establishing a Task with Pure and Impure Segments

The required command sequence is as follows:

```
    ESTABLISH TASK, PURE
    INCLUDE FILEA,PROGRAMA
    INCLUDE, PROGRAMC
    INCLUDE,PROGRAMD
```

EDIT	Includes PROGRAM F referenced by PROGRAM A, and PROGRAM H referenced by PROGRAM C. Programs A and D are already included.
BUILD TASK	As no fd is specified, and assuming that LU 2 is not assigned to a contiguous disc file, this task is built in memory and output to LU 2.
MAP	Produces a map of the task
END	Returns control to the system

D.7.3 Establishing a Reentrant Library

This example describes the establishment of the FORTRAN run time library. The example assumes that the run time library is input to TET on five paper tapes.

ESTABLISH RL	Specifies that a reentrant library load module be established
INCLUDE PTRP: INCLUDE PTRP: INCLUDE PTRP: INCLUDE PTRP: INCLUDE PTRP:	Includes five paper tapes
BUILD RL,FORT.RTL,RELIBRY,10	Builds a reentrant library load module with the name RELIBRY, using segment 10.
MAP	Produces a map of the completed module
END	Returns control to the system

D.7.4 Establishing a Complex Task with Overlays

The task comprises all three programs from File A and, initially, one program from File C (Program D). However, Program D has references to Programs A, B, and F in the same file. Therefore, the main (impure) segment contains seven programs. Two overlays are necessary for the task. The first overlay is from File C (Program E with the two references, Programs B and C in File B). The second overlay is File B Program A. Figure D-2 is a graphic representation of the task.

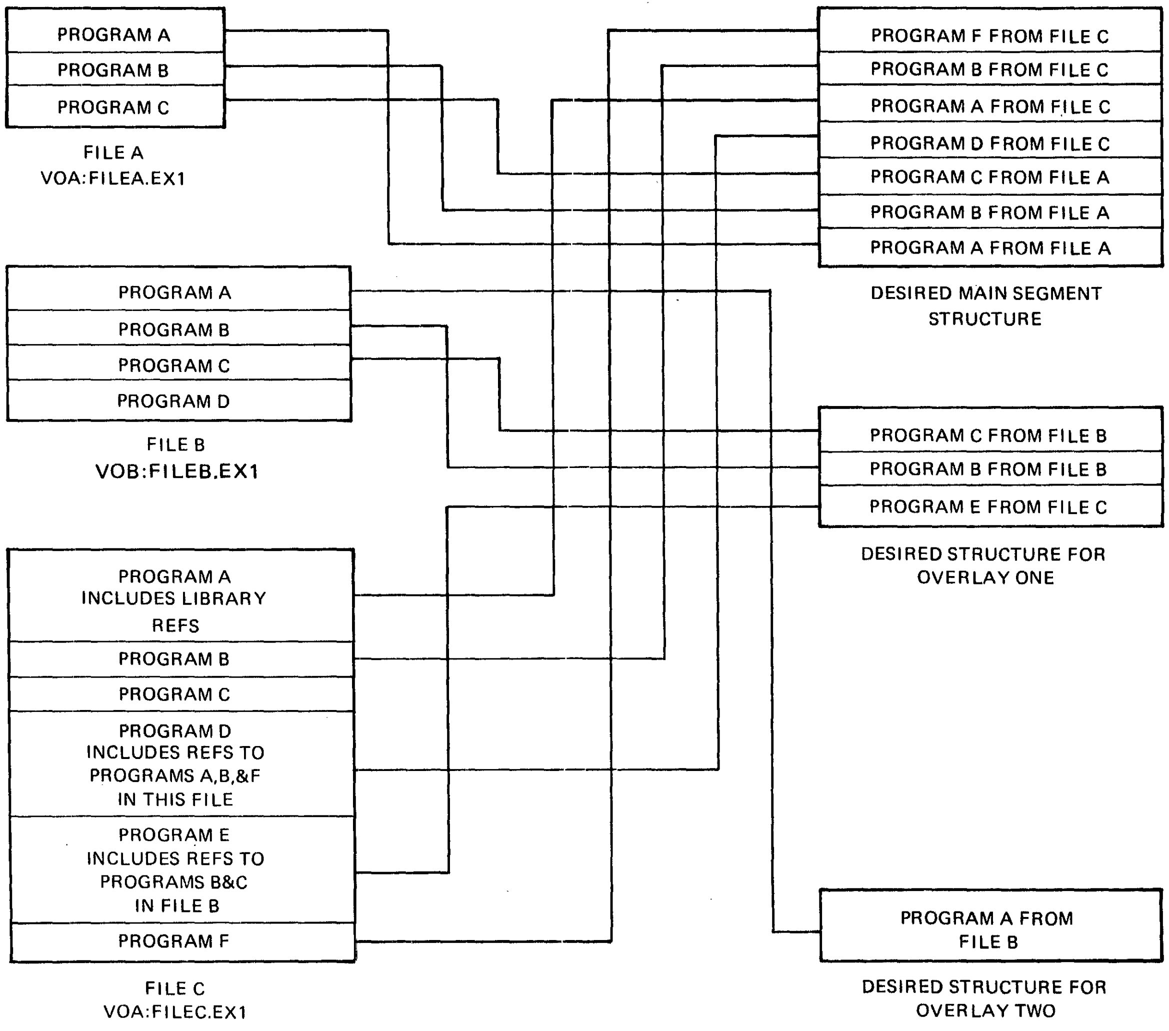

Figure D-2 Graphic Description of a Task with Two Overlays

This command sequence shows the establishment of a task with overlays, references to reentrant libraries, and task common segments. The main segment and each overlay are built on a separate file. Automatic file allocation is used.

The command sequence to establish the task is as follows:

REMOTE	Specifies that TET should not pause; command input is batch.
VOLUME VOA	VOA is the default volume descriptor.
ESTABLISH TASK ABS 200	Provides X'200' bytes of absolute address space before beginning the impure relocatable code.
TCOM GLOBAL/13/RW	Designated common block named GLOBAL as task common; positions GLOBAL in segment 13, program address D0000. Segment has read/write access privileges.
TCOM COMALL/12//16	Designates common block named COMALL as task common, positions COMALL in segment 12, program address C0000. Segment has read/write access privileges as default override segment size=4kb.
INCLUDE FILEA.EX1	Includes all of FILEA.
INCLUDE FILEC.EX1, PROGRAMD	Includes PROGRAM D of FILEC.
REWIND FILEC.EX1	Rewinds FILEC for editing.
EDIT FILEC.EX1	Edits FILEC to include programs A, B, and F in the temporary file.
RESOLVE MAG1: RESOLVE LIB2	The RESOLVE command reads the RL LIBs and satisfies the references. The libraries have been previously established.
MXSPACE 2400	Allows the task to use up to X'2400' bytes of system space for file control blocks.
MAXLU 20	Allows the task to use LUs numbered between 0 and 20.
PRIORITY 20,14	Specifies that the task runs at priority 20. The priority may be

	changed during run time, but may not be set higher than 14.
EXPAND 4	Allows 1024 bytes for get storage calls in the main segment.
OVERLAY OVONE	Terminates definition of the main segment; names and initiates definition of first overlay.
INCLUDE FILEC.EX1, PROGRAME	First overlay contains PROGRAME from FILEC.EX1/
EDIT VOB:FILEB.EX1	Edits FILEB for programs referenced by PROGRAME. Includes PROGRAMB and PROGRAMC from FILEB.
OVERLAY OVTWO	Terminates first overlay definition and initiates second.
INCLUDE VOB:FILEB.EX1, PROGRAMA	Second overlay contains PROGRAMA from VOB:FILE.EX1.
BUILD TASK,FILED.TSK	Initiates pass two processing and starts building the load module of the task's main segment, using temporary files. The fd of the load module is VOA:FILED.TSK. The extension field can be any three characters such as TSK.
BUILD OVLY,FILEG.OVY	Builds first overlay load module whose fd is VOA:FILEG.OVY.
BUILD OVLY,FILEH.OVY	Builds second overlay load module whose fd is VOA:FILEH.OVY.
MAP	Produces a map of entire task
END	Returns control to the system

The impure main segment is built with the absolute address area first, followed by the main and local common areas, an overlay area large enough for the longest overlay, and the EXPAND requested area. At run time the system places the CTOP indicator at the top of the EXPAND area, rounded up to a 256-byte boundary.

The following illustration describes how the task is organized when loaded into memory. Note that the two libraries were previously established. Segment 15 was allocated for the library on MAG1:, and segment 14 was allocated for the library on file LIB2.RTL.

```
                                    --------------------------------
  RL SEGMENTS (15)                  | LIBRARY FROM MAG1:           |
                       F0000        --------------------------------

                                    --------------------------------
              (14)                  | LIBRARY FROM LIB2            |
                       E0000        --------------------------------

                                    --------------------------------
TCOM SEGMENTS (13)                  | TASK COMMON NAMED 'GLOBAL'   |
                       D0000        --------------------------------

                                    --------------------------------
              (12)                  | TASK COMMON NAMED 'COMALL'   |
                       C0000        --------------------------------
                       CTOP
                                    --------------------------------
                                    | EXPAND AREA (1024 BYTES)     |
                       UTOP         |------------------------------|
                                    |                              |
                                    | OVERLAY AREA                 |
                                    |                              |
                                    | LOCAL COMMON                 |
                                    |                              |
IMPURE SEGMENT (0)                  | IMPURE CODE/DATA AREA TASK'S |
                                    |                       MAIN   |
                                    |                       (ROUT) |
                                    |                       SEGMENT|
                                    |                              |
                        200         |- - - - - - - - - - - - - - - |
                                    | USER DEDICATED LOCATIONS     |
                                    --------------------------------
                          0
```

Figure D-3   Memory Map of Overlay Task Establishment

D.7.5 Building an Operating System Image

This command sequence builds an operating system image on a contiguous file. The OS image produced can be loaded by the 32-bit direct address bootstrap loader or the loader storage unit. The operating system should be generated in object format using CUP/MT and the library loader, as outlined in the OS/32 System Planning and Configuration Guide. TET processes the object module produced by the library loader by using the following command stream:

ESTABLISH TASK

ABSOLUTE xxxxx

xxxxx is the address of UBOT from the library loader map.

INCLUDE MTSYSTEM.OBJ

Includes the output of the library loader (assumes the object is contained on file MTSYSTEM.OBJ).

BUILD TASK,OS32MT.301

TET allocates a contiguous file of the proper size and builds an OS image.

END

Returns control to the system

NOTE

To be loadable by either the boot loader or the LSU, the fd of the file containing the OS image must be of the form:

OS32xxxx.hhh

xxxx can be any four ASCII characters; hhh must be a hexadecimal number between X'000' and X'FFF'.

D.7.6 Establishing Compound Overlay Files

The user can build overlays sequentially on a single file. When building overlays to single disc files, default allocation of the files by TET is not sufficient. The user must allocate the file before starting TET. To determine the exact length of the required file, lengths of the individual images, expressed as the number of 256-byte blocks, must be summed to arrive at the approximate total image length. In addition, a single LIB sector for each image to be contained in the file, plus four sectors for work space, are required by TET. The following numerical example is provided:

Overlay 1	500 hex	round up to the nearest X'100' byte boundary
Overlay 2	2200 hex	
Overlay 3	1500 hex	

The segment length expressed in sectors is:

```
Overlay 1         500/100 = 5 sectors
Overlay 2        2200/100 = 22 sectors
Overlay 3        1500/100 = 15 sectors

                 TOTAL    = 3C sectors (hex)
                          +  3 sectors (LIBs)
                          +  4 sectors (workspace)
                 TOTAL    = 43 sectors (hex)
                                or
                             67 sectors (decimal)
```

When generating compound disc files, it is not always permissible to change output units during the process. Consider a task with four overlays, where overlays 1, 2, and 4 are directed to one fd, and overlay 3 is directed to another. Overlays 1 and 2 are built sequentially. LU 2 assignment is changed for overlay 3, and when reassigned for overlay 4, the disc file is rewound. Overlay 4 overwrites those previously built on that file. It is recommended that all overlays for one task be built on the same file, or each overlay be built on a separate file.

D.7.7 Establishing a Block Data Task Common Segment

The following example illustrates how to establish a block data task common segment. Block data is used to initialize the task common area.

```
ESTABLISH TCOM              second parameter defaults to BDATA
INCLUDE FILEA.EX1,PROGB
INCLUDE FILEC.EX9
BUILD TCOM,COMMONBK.TCM
END
```

PROGB in FILEA.EX1 is a block data subprogram containing a common block definition named COMMONBK that initializes several items. FILEC.EX9 contains several block data subprograms, where each program declares common block COMMONBK and initializes or reinitializes items in that block.

D.7.8 Establishing a Sharable Segment

Assume file EDIT.OBJ is a text editor whose code body and absolute data are assembled as pure and whose buffers and save areas (any variable data) are assembled as impure. The following command sequence establishes a pure and impure segment for the editor task:

```
    ESTABLISH TASK,PURE
    INCLUDE EDIT.OBJ
    BUILD TASK,EDIT.TSK
    MAP
    END
```

The first time EDIT.TSK is loaded by the operating system loader, both the pure and impure segments are loaded into memory. Subsequent loads of EDIT.TSK result in only the impure segment being loaded, assuming the pure segment is still in memory. All editing tasks share the same pure segment, saving a considerable amount of memory.

D.7.9 Establishing Preinitialized Task Common

A load module of a task common can be generated by including nonblock data (absolute and variable data) programs as in this example:

```
    ESTABLISH TCOM,NOBDATE
    INCLUDE MAG1:             Reads all programs on mag tape
    INCLUDE PTRP:             Reads all programs on paper tape
    BUILD TCOM,PTRP:,TSKCCM   Load module is output on paper tape
                              punch.    Name of task common is
                              TSKCOM.
    MAP CON:                  Outputs a map on console device
    END
```

D.8 TET ERROR MESSAGES

TET001 symbol NOT FOUND

Search for a program label has failed

TET001 ABS LESS THAN 100

Warning that ABS request overlaps UDL

TET001 ACCESS PRIVILEGE CONFLICT

Warning that access privileges in a LBLCOM command do not match the corresponding access privileges in a TCOM Command.

TET001 ADRS OFLO AT xxxxxx

Halfword address computation greater than FFFF. xxxxxx is address in program where overflow has occurred.

TET002 ADRS OFLO AT xxxxxx

Fullword address computation greater than FFFFFF. xxxxxx is address in program where overflow has occurred.

TET001 CKSM ERR

Checksum error on input object file

TET001 DEV END

End-of-file or end-of-medium encountered on an LU

TET002 DEV END

Device unavailable (hardware error)

TET001 EOM

End-of-medium on LU 1

TET001 EXP TOO BIG

Expand request makes task greater than X'100000'

TET001 FCB AREA FULL

Attempt to assign a file has failed

TET001 FD SYNTAX

File descriptor syntax is incorrect

TET001 FILE ERR: xxxx fd

Error while closing file. xxxx=SVC 7 status, lu (fd)=filename

TET002 FILE ERR: xxxx fd

Error while opening file. xxxx=SVC 7 status, lu (fd)=filename

TET003 FILE ERR: xxxx fd

Error while allocating file. xxxx=SVC 7 status, and lu (fd)= filename

TET001 HW EXTRN ABOVE LIMIT

Halfword reference corresponding definition lies above 64kb address

TET001 ILLEGAL SEGMENT NO.

Segment number in a LBLCOM command is not used in a previous TCOM command

TET001 ILG ABS ADRS

ABS code higher than specified in ABSOLUTE command found in this task

TET002 ILG ABS ADRS

ABS code found in reentrant library segment

TET003 ILG ABS ADRS

ABS code in E-task

TET001 ILG CMD

Command verb is not valid

TET001 ILG CMD PARM

Improper command argument

TET002 ILG CMD PARM

Illegal second parameter in a command

TET003 ILG CMD PARM

Illegal third parameter in a command

TET004 ILG CMD PARM

Illegal fourth parameter in a command

TET001 ILG CMD SEQ

Command not legal at this point in TET operation

TET001 ILG COMN DEFN

TET has encountered common loader items while building a reentrant library segment

TET002 ILG COMN DEFN

TET has encountered common loader items while building an E-task

TET003 ILG COMN DEFN

TET has encountered common loader items for block data in task common

TET004 ILG COMN DEFN

TET has encountered common loader items for block data in an overlay

TET005 ILG COMN DEFN

TET has encountered common loader items; blank common definition found when building BDATA-TCOM

TET006 ILG COMN DEFN

TET has encountered common loader items; more than one common definition in block data subprogram

TET001 ILG DELIMITER x

Command syntax error; a delimiter other than x is required

TET001 ILG OBJ ITEM xx

Loader item from an input object program is illegal (xx=item, see Table A4-1)

TET002 ILG OBJ ITEM xx

Loader item from an input object program is illegal when building RTL (xx=item, see Table D-5)

TET003 ILG OBJ ITEM xx

Loader item from an input object program is illegal when building BDATA-TCOM (xx=item, see Table A4-1)

TET004 ILG OBJ ITEM xx

Loader item from an input object program is illegal when building NOBDATA-TCOM (xx=item, see Table A4-1)

TET001 ILG OPTION

Roll with E-task, roll with resident

TET001 IMP IN RTL

Impure code encountered while processing an RTL segment

TET001 I/O DEV ERROR xxdd (fd)

An I/O error encountered while reading commands; fd=device file descriptor or LU 5 not assigned

TET002 I/O DEV ERROR xxdd (fd)

An I/O error encountered while reading input records; fd=device file descriptor or LU 5 not assigned

TET003 I/O DEV ERROR xxdd (fd)

An I/O error encountered while reading an LIB; fd=device file descriptor or LU 5 not assigned

TET004 I/O DEV ERROR xxdd (fd)

An I/O error encountered while writing to a temporary device; fd=device descriptor or LU 5 not assigned

TET005 I/O DEV ERROR xxdd (fd)

An I/O error encountered while outputing a load module; fd=device descriptor or LU 5 not assigned

TET006 I/O DEV ERROR xxdd (fd)

An I/O error encountered while printing a map; fd=device descriptor or LU 5 not assigned

TET007 I/O DEV ERROR xxdd (fd)

An I/O error encountered; SVC 7 fails during preparation for pass two; fd=device descriptor or LU 5 not assigned

TET001 LCOM IN OVLY

Common definition in overlay segment is not previously defined in main segment

TET001 LCOM TOO BIG

Labeled common too large

TET002 LCOM TOO BIG

Blank common too large in overlay

TET001 LCOM IN RTL

Encountered common loader item while building reentrant LIB segment

TET001 LOAD PROGRAMS fd

Prompts operator to load program if temporary device is not being used, or to rewind input fd

TET001 LU (lu) NOT ASSIGNED

No device assigned to this LU

TET001 MEM FULL

TET work area has been exhausted. No room to build first LIB

TET002 MEM FULL

TET work area has been exhausted. No room to build subsequent LIBs

TET001 MULT DEFD symbol

Multiple definition of the same symbol

TET001 NO DCHN END xxxxxx

xxxxxx is the def address. Address dechaining has exceeded 200,000 links

TET001 NO ENTRIES IN RTL

No entry definition found when building RTL

TET001 OUTSIDE AVAIL MEM

Internal TET error. Attempt to store image code above available user space

TET002 OUTSIDE AVAIL MEM

Internal TET error. Attempt to store image code below user space

TET001 OV NAME MULT DEFD

More than one overlay with same name

TET001 OV MULT DEFD symbol

Two or more overlays define a symbol referenced by the main segment

TET001 TCOM ACCESS PRIV ERR xxxx

Access privilege in TCOM command not RO or RW (xxxx=task common name)

TET001 PROG TOO BIG

Task greater than X'100000'

TET002 PROG TOO BIG

RL greater than 64kb

TET003 PROG TOO BIG

TCOM size greater than X'100000'

TET001 REPOSITION SCRATCH

Message to operator when using nonrewindable temporary device

TET001 RESOLVE ERR

Input file not an RL file

TET002 RESOLVE ERR

Input file not the same name as in a previous RESOLVE command

TET001 SEGMENT ADDRESSING ERROR

Error while mapping logical segment into task address space; segment start address plus size greater than Y'100000'

TET002 SEGMENT ADDRESSING ERROR

Error while mapping logical segment into task address space; segment addresses overlap

TET003 SEGMENT ADDRESSING ERROR

Error while mapping logical segment into task address space; not enough contiguous space for a segment

TET001 SEQ ERR

Sequence number error on input object file

TET001 SHORT RECORD

Record length of build file is less than 256

TET002 SHORT RECORD

Record length of temporary file is less than 126

TET001 TCOM ADRS OUT OF RANGE xxxx

Segment number in TCOM command is less than 1 or greater than 15. (xxxx=task common name)

TET001 TCOM MULT DEFD xxxx

Name xxxx in TCOM command has been used previously in TCOM command

TET001 TCOM NAME SYNTAX ERROR xxxxx

Name xxxx in TCOM command longer than eight characters or not terminated by /.

TET001 TCOM TOO BIG

Second task common definition larger than first

TET002 TCOM TOO BIG

Override size in TCOM command smaller than in common definition

TET001 TET 32 ABORTED

An unrecoverable condition has been detected in batch mode; operator action is required

TET002 TET 32 ABORTED

An unrecoverable condition has been detected when building; no impure or pure code

TET003 TET 32 ABORTED

An unrecoverable condition has been detected; pass two command not found in dictionary

TET004 TET 32 ABORTED

An unrecoverable condition has been detected, edit table overflow; more than 8192 programs processed by EDIT

TET005 TET 32 ABORTED

An unrecoverable condition has been detected, common reference; no corresponding definition

TET006 TET 32 ABORTED

An unrecoverable condition has been detected, block data; no corresponding definition

TET007 TET 32 ABORTED

An unrecoverable condition has been detected, overlays included not found in dictionary

TET008 TET 32 ABORTED

An unrecoverable condition has been detected, building overlay; overlay not found in dictionary

TET009 TET 32 ABORTED

An unrecoverable condition has been detected, object record read routine failed

TET010 TET 32 ABORTED

An unrecoverable condition has been detected; segment table validation error

TET011 TET 32 ABORTED

An unrecoverable condition has been detected; invalid command in error format table

TET012 TET 32 ABORTED

An unrecoverable condition has been detected; a common block defined in a TCOM command contains block data.

TET001 TOO MANY SEGMENTS

More than 16 program segments

TET001 TOO MANY TCOMS xxxx

Only 15 task commons can be named in TCOM commands (xxxx=task common name)

TET001 UNDEFINED SYMBOLS

Operator warning that undefined symbols exist

TET001 WRONG PROG

Program encountered on pass two is in wrong order; symbol not in dictionary

TET002 WRONG PROG

Program encountered on pass two is in wrong order; symbol has wrong address

TET003 WRONG PROG

Program encountered on pass two is in wrong order; no such symbol in common DEF's

TET004 WRONG PROG

Program encountered on pass two is in wrong order; block data subprogram not included when building BDATA-TCOM

TET005 WRONG PROG

Program encountered on pass two is in wrong order; relocatable address in pass two beyond limit

TET006 WRONG PROG

Program encountered on pass two is in wrong order; module size not same as pass one size

TABLE D-5 OBJECT ITEM SIGNIFICANCE

OBJECT ITEMxx	MEANING
0	End-of-record
1	End-of-program
2	Reset sequence number
3	Block data indicator
4	Absolute program address
5	Pure relocatable program address
6	Impure relocatable program address
7	Two bytes of pure relocatable data
8	Two bytes of impure relocatable data
9	Four bytes of pure relocatable data
A	Four bytes of impure relocatable data
B	Common reference
C	EXTRN
D	ENTRY
E	Common definition
F	Program label
10	Three bytes ABS and three bytes pure relocatable data
11	Three bytes ABS and three bytes impure relocatable data
12	Load program transfer address

TABLE D-5 OBJECT ITEM SIGNIFICANCE (Continued)

OBJECT ITEMxx	MEANING
13	Define start of chain
14	Load chain definition address
15	Two bytes ABS and two bytes pure relocatable data
16	Two bytes ABS and two bytes impure relocatable data
17	Short form EXTRN
18	Length of impure and pure segments
19	Perform fullword chain
1A	Perform halfword chain
1B	No operation
1C	2-byte pure translation table address
1D	2-byte impure translation table address
1E	Illegal
1F	One byte ABS data
20	Two bytes ABS data
21	Four bytes ABS data
22	Six bytes ABS data
23	Eight bytes ABS data
.	. . . .
.	. . . .
.	. . . .
5B	120 bytes ABS data
5C to FF	Illegal

INDEX

PUBLICATION COMMENT FORM

Please use this postage-paid form to make any comments, suggestions, criticisms, etc. concerning this publication.

From ______	Date ______
Title ______	Publication Title ______
Company ______	Publication Number ______
Address ______	
______	
______	

FOLD FOLD

Check the appropriate item.

☐ Error	Page No. ______	Drawing No. ______
☐ Addition	Page No. ______	Drawing No. ______
☐ Other	Page No. ______	Drawing No. ______

Explanation:

FOLD FOLD

CUT ALONG LINE

Fold and Staple
No postage necessary if mailed in U.S.A.

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
Computer Systems Division
2 Crescent Place
Oceanport, NJ 07757

TECH PUBLICATIONS DEPT. MS 322A

FOLD FOLD

STAPLE STAPLE