PERKIN-ELMER

OS/32
LINK
Reference Manual

48-005 F00 R02

The information in this document is subject to change without notice and should not be construed as a commitment by the Perkin-Elmer Corporation. The Perkin-Elmer Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license, and it can be used or copied only in a manner permitted by that license. Any copy of the described software must include the Perkin-Elmer copyright notice. Title to and ownership of the described software and any copies thereof shall remain in The Perkin-Elmer Corporation.

The Perkin-Elmer Corporation assumes no responsibility for the use or reliability of its software on equipment that is not supplied by Perkin-Elmer.

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

Printed in the United States of America

TABLE OF CONTENTS

CHAPTERS (Continued)

CHAPTERS (Continued)

APPENDIXES

FIGURES

TABLES

PREFACE

This manual describes the Perkin-Elmer linkage editor, OS/32 Link, which provides the user with the ability to link one or more object modules to produce an executable image. An image can be a task, a partial image, or operating system. This manual is intended for all users who are developing programs for execution on Perkin-Elmer 32-bit computers. The user should be familiar with the Perkin-Elmer OS/32 Multi-Terminal Monitor (MTM) if Link is to be used in an MTM environment. See the OS/32 Multi-Terminal Monitor (MTM) Reference Manual.

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 active, passive, and environment Link commands. Chapter 4 provides examples of Link command sequences. Chapter 5 introduces and explains virtual task management (VTM). Appendix A is the Link command summary. Appendix B is the Link message summary. Appendix C is the VTM message summary. Appendix D explains the format of an object module that is compatible with Link.

Revision 02 of this manual adds support for DEBUG/32 and VTM details changes to the DCMD and the OPTIONS commands to support the Perkin-Elmer Multiprocessor System (Model 3200MPS).

This manual is intended for use with the R01 version of OS/32 Link and the OS/32 R06.2 software release. Additional material specifically related to the Model 3200MPS System has also been included. These Model 3200MPS features are supported by the OS/32 R07.1 software release. Throughout the text, these features are identified as applicable to the Model 3200MPS System only.

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

CHAPTER 1
OS/32 LINK

1.1 INTRODUCTION

Perkin-Elmer OS/32 Link provides the user with the ability to link one or more object modules to produce a task image or partial image that can be loaded via the OS/32 LOAD command.

Link can also build an operating system image from the object module produced by the Perkin-Elmer OS/32 Library Loader or SYSGEN/32. The resulting image can be loaded into memory using the Perkin-Elmer OS/32 Bootstrap Loader or Loader Storage Unit (LSU).

This release of Link includes the DEBUG/32 tables (DTABLES) task option and supports the virtual task manager (VTM). This option allows Link to separate symbolic debug data from the object code and build this data into the tables required by DEBUG/32. VTM provides a user-transparent virtual memory capability that allows some user tasks (u-tasks) consisting of up to 16MB of code and data to execute in as little as 128k bytes of memory.

OS/32 Link can be used with both the Perkin-Elmer Uniprocessor System and the Perkin-Elmer Multiprocessor System (Model 3200MPS). The multiprocessor system consists of one central processing unit (CPU) and up to nine auxiliary processing units (APUs). In a multiprocessor system, the operating system defines a set of logical processing units (LPUs) that are used to direct tasks to physical processors. An LPU is mapped to the CPU or an APU, and each task is assigned an LPU. Link assigns the initial LPU for each task. Link also sets APU control or mapping privileges when building a task, and can optionally list comments embedded in the object file. See the Link DCMD and OPTIONS commands. Also see the Perkin-Elmer Model 3200MPS Overview Manual for more information on using the Model 3200MPS.

1.2 IMAGE FILE FORMAT

Link allocates an image file on disk and builds an image into this file or builds the image into an already existing file. The format of the image file for a task is shown in Figure 1-1.

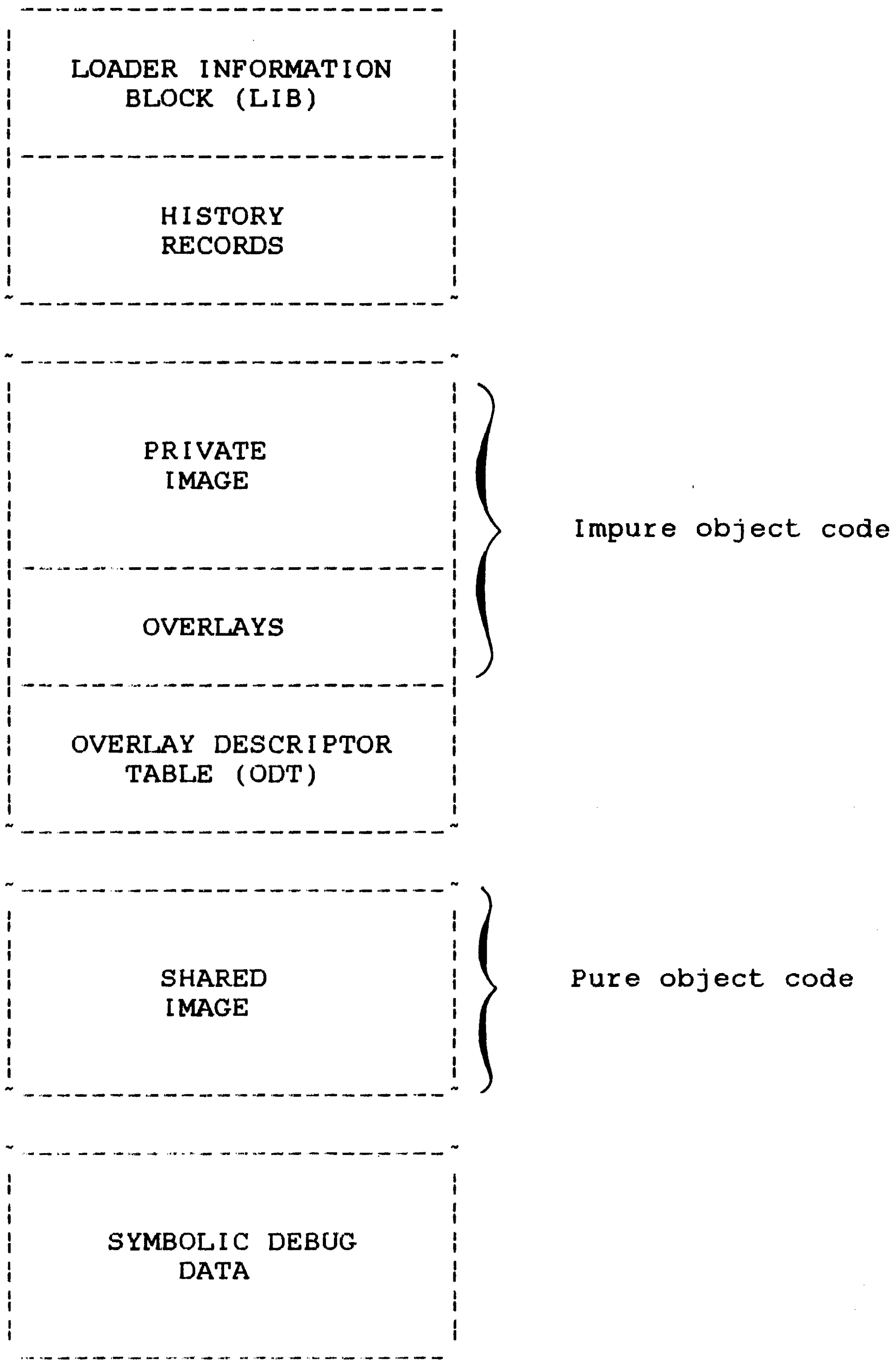


Figure 1-1 Task Image File Format

The first segment in the task image file is the loader information block (LIB). The LIB tells the loader how to load the image into memory. For example, the first byte of the LIB indicates the type of image which is to be loaded. When the task is loaded by the LOAD command, the LIB is kept in the loader's private memory area, not in task memory, until the loader no longer requires it.

Following the LIB is the history records area. This area is defined by OS/32 PATCH; that is, any changes made to the task or to its LIB via PATCH are stored in this area of the file.

The task image that is actually loaded into memory consists of at least one private image segment. Link creates this segment with read, write, and execute privileges below the LIB. The private image segment contains the impure code and, if the NSEGMENTED option in the Link OPTION command was specified, the pure code from the included object modules. Each user who loads the task is provided with a copy of the private image. The first segment of the private image is known as the root segment. The root contains the user-dedicated location (UDL), the primary task workspace, and the task overlay areas defined by the OVERLAY command, along with other user-selected items. In addition, any absolute code found in the object modules is included in the root.

The overlay descriptor table (ODT) following the overlay areas contains instructions for loading the overlays. The ODT is located in the task control block after the image is loaded.

If the SEGMENTED option was specified to Link, the pure code belonging to the root node from the object modules is placed in the shared image segment of the image file. This area has only read and execute access privileges. When the task is first loaded, the shared image is also loaded into memory. Users can read and execute this segment but cannot write to it. Only one copy of the shared image remains in memory during multiple simultaneous executions of the task.

If the task is to be debugged using DEBUG/32, Link formats the task data required by the symbolic debugger and places it in the shared image.

External segments referenced by the task are known as partial images. Partial images can contain any combination of one or more shared segments and private segments that can be used by many different programs; e.g., the Perkin-Elmer FORTRAN VII Run-Time Library (RTL). A partial image is not formatted as a complete task. Its image file consists only of the LIB followed by a single shared segment. Instructions for resolving a partial image referenced by a task are given by the RESOLVE command. See Section 3.21.

The virtual address map of the link establishment summary defines where the root, shared, and partial images will be loaded into memory for the task. See the MAP command (Section 3.14) for more information on the establishment summary.

1.3 LINK SYMBOL TABLE

Before Link actually builds the image into a file, Link builds a symbol table of all of the information required to build the image. This table is used in the image building and map production steps.

As commands are entered, this table grows in memory. When Link runs out of available real memory, it allocates a temporary disk file and copies this table out to the file. Parts of the table are swapped between memory and the file, as required. The less real memory available, the more swapping Link is required to perform and the longer it takes Link to build an image.

To allocate more memory for the Link symbol table, load Link using the workspace parameter of the LOAD command explained in Section 2.2.

1.4 OVERLAYING A PROGRAM USING LINK

During its lifetime, a program may become very large. Link provides a means to execute a program in an area of main storage that is not actually large enough to contain the entire task at one time. Link divides such a program into nodes, collections of modules and common blocks, that are loaded as needed. Only one private node, the root, must remain in main memory throughout the execution of the program; the other nodes reside on disk, from where they are fetched, when needed.

To ensure the integrity of the overlayed program, an overlay structure must be carefully designed. This structure is a tree that shows which nodes of a program occupy the same main memory at different times. Figure 1-2 is a graphic example of an overlay tree structure.

Sample Program

```
|--------
| Call B
| Call C
| Call X
| END; MAIN
|--------
| Subroutine B
|
|
|
| Call X
|
| END; B
|--------
| Subroutine C
|
| Call D
|
| Call X
|
| END; C
|--------
| Subroutine D
| Call X
| Call E
| Call F
|
| END; D
|--------
| Subroutine E
| Global E_AND_F
| Call X
|
| END; E
|--------
| Subroutine F
| Global E_AND_F
| Call X
|
| END; F
|--------
| Subroutine X
|
| END; X
|--------
```

Overlay Tree Structure

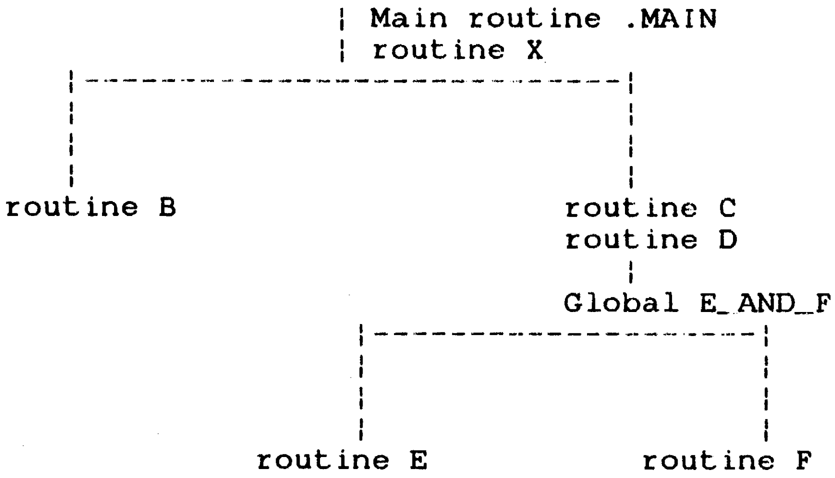


Figure 1-2 Sample Program with Overlay Tree Structure

The sample program is composed of one main routine and six subprograms; B, C, D, E, F, and X. The main routine calls B and C. C, in turn, calls D which calls E and F. All routines call X, and E and F share the global variable E_AND_F.

The main routine must reside in the root node throughout the execution of the task. Also, X should be placed in the root because all other routines call X in this sample program.

The execution of B and C are mutually exclusive; that is, they never call each other directly or indirectly. Therefore, these two subprograms can occupy the same address space. C must remain in storage while D, E, and F are executing. However, there is nothing to be gained by separating routines C and D since they must be present simultaneously, so C and D can be placed in the same node.

The following Link command sequence can be used to implement the overlay structure in Figure 1-2.

```
INCLUDE MYPROG.OBJ,.MAIN
INCLUDE ,X
OVERLAY B,1
   INCLUDE ,B
OVERLAY CD,1
   INCLUDE ,C
   INCLUDE ,D
   OVERLAY E,2
       INCLUDE ,E
   OVERLAY F,2
       INCLUDE ,F
LIBRARY MYLIB.OBJ
LIBRARY F7RTL.OBJ
BUILD MYPROG
```

The OVERLAY command specifies the start of a node and the node's relative position within the tree structure. The two RTL files, MYLIB and the standard Perkin-Elmer RTL, are searched by Link (MYLIB first, then F7RTL.OBJ) for any routines containing entry points matching the unresolved external references of the program. Link places a copy of a library routine in the referencing node unless an ancestor node already contains a copy.

Each node has a fixed length in bytes. The total size of a task depends upon both the routine composition of each node and the structure of the overlay tree. An overlay structure can be represented by a set of parallel paths. A path can be defined as a particular set of nodes (one at each level), each of which is a descendent from the previous level. Therefore, the total size of a task is determined by the path in which the node sizes add up to the greatest number of bytes. By using the cross-reference map from Link, one can manually build a call-tree representation of a program (similar to the one shown in Figure 1-2) as an aid in determining the smallest possible task size.

Normally, the placement of a common block or global block within an overlayed task is determined by the locations that refer to the block. Named common and global blocks, however, are initially positioned by Link no closer to the root than any particular reference to the block. In the sample program in Figure 1-2, subprograms E and F both refer to the global variables E_AND_F. Link will place E_AND_F in the node containing subprograms C and D.

The first consequence is that named common and global entities are initialized every time the overlay is fetched from disk. The second consequence is that more than one copy of a common or global entity can exist on separate paths in the program. That is, two or more overlays can have their own separate and private copies of a common or global entity. These copies could then contain different values.

Link provides the POSITION command to reposition common or global entities into an overlay closer to the root than they normally would be positioned. Global E_AND_F, in the sample program, can be forced into the root node by inserting into the sample Link command sequence:

```
    POSITION  Common=E_AND_F,To=.ROOT
```

1.5 USING LINK-DEFINED SYMBOLS

Link defines seven symbols for general use:

- @TIME1 - HH:M (hour and first digit of minute)
- @TIME2 - M:SS (second digit of minute and second)
- @DATE1 - MM/D (month and first digit of day, assuming default of DATE option is specified at system generation)
- @DATE2 - D/YY (second digit of day and year)
- @UBOT - Address of the lowest byte in the image being built. For tasks, this is always zero. For partial images, this is the first byte of the segment named in the ESTABLISH command.
- @UTOP - Address of the first byte following the included object code. It is rounded according to the ALIGN option specified in the OPTION command.
- @CTOP - Address of the last addressable halfword of the image.

The following program shows how the time and date of a linkedit session can be included in the task image by referencing the symbols @TIME1, @TIME2, @DATE1, and @DATE2.

Example:

```
LINKDAY  PROG  Demonstration progr
         EXTRN @TIME1,@TIME2,@DATE
         EXTRN @UTOP,@CTOP


         PURE
START    SVC   2,LOGLINK
         LA    0,@UTOP
         LA    1,@CTOP
         SVC   2,PAUSE
         SVC   3,0


         ALIGN ADC
PAUSE    DB    0,1


         IMPUR
         ALIGN ADC
LOGLINK  DB    0,7,0,80
         DB    C'Linkedited at - '

         DCF   @TIME1,@TIME2

         DB    C' on '
         DCF   @DATE1,@DATE2

         DB    X'OD',O
         END
```

```
load linkdemo
start
Linkedited at -17:26:10 on 05/26/82
TASK PAUSED
d r
 PSW 000077FO  00000148
 0-3 00000150  000001FE  00000000  00000000
 4-7 00000000  00000000  00000000  00000000
 8-B 00000000  00000000  00000000  00000000
 C-F 00000000  00000000  00000000  00000000
continue
ROD     -END OF TASK CODE=   0    CPUTIME=0.003/0.002
```

1.6 SYSTEM REQUIREMENTS

System requirements for Link R01 are:

- OS/32 5.2 or higher (if DEBUG/32 is used, Link requires OS/32 R06.1 or higher)
- 1 disk device
- 128kb of main storage for Link

1.7 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 from left to right. If a syntax error is detected in a command, that command plus any subsequent commands on the same line are ignored.

In interactive mode, if the specified parameters of a command exceed one line, entering a comma as the last character and a carriage return (CR) causes the following 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 are specified by entering an asterisk (*) before the comment string and placing a CR or semicolon at the end of the string. A comment can be the only data on a line or can precede or follow a command on the same line.

Examples:

```
    *THIS IS THE LINK ROUTINE

    ESTABLISH TASK;*A TASK IS TO BE ESTABLISHED

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

Unless otherwise noted, if the syntax of a Link command includes "number" as a parameter, the number specified is a positive whole number.

1.7.1 File Descriptors

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

Format:

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

Parameters:

voln: is a 1- to 4-character alphanumeric string specifying the name of a disk 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, or
- 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 period (.) followed by 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 65,535 specifying the account number associated with the file. Account numbers 1 through 65,535 (excluding 255) are supported by MTM. Account number 255 is reserved for the authorized user utility. Account number 0 is for system files and is the default account number for all operator commands.

NOTE

Account numbers can only be specified as part of the fd when Link is run from the system console or when Link is run under MTM from an account that has file account privileges.

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 Application Level Programmer Reference Manual for more information on file descriptors.

CHAPTER 2
BUILDING AND STARTING LINK

2.1 BUILDING LINK

If the Perkin-Elmer supplied ready-to-execute version of Link 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 segmented task using the Perkin-Elmer supplied version of Link:

```
ES TASK
OPTION ACPRIVILEGE,SYSSPACE=XFFFF
OPTION SEGMENTED,WORK=(X8000,XC000)
INCLUDE LINK
MAP CON:ALPHABETIC,ADDRESS,XREF
BUILD LINK
END
```

The reserved workspace must be a minimum of 8kb. The more workspace allocated, the less paging to and from disk occurs. The less workspace allocated, the more paging to and from disk occurs. The amount of workspace specified can be overridden when Link is loaded.

2.2 LOADING LINK

Before Link can be loaded into main storage, it must be built as a task image.

2.2.1 Loading Link from the System Console

The following system command loads Link from the system console:

Format:

```
LOAD taskid [,fd] [,workspace]
```

Parameters:

taskid — is a 1- to 8-character alphanumeric string specifying the name of the task after it is loaded into main memory.

fd — is the file descriptor of the device containing the linkage editor image to be loaded into main memory. If this parameter is omitted, the default is taskid.TSK.

workspace — is a decimal number in kb specifying the additional area to be added to the root node. This value overrides the WORK= option if specified when the image was built.

2.2.2 Loading Link from a Multi-Terminal Monitor (MTM) Terminal

The following MTM command loads Link from an MTM terminal:

Format:

```
    LOAD fd[,workspace]
```

Parameters:

fd — is the file descriptor of the device containing the linkage editor image to be loaded into main memory.

workspace — is a decimal number in kb specifying the additional area to be added to the root node. This value overrides the WORK= option if specified when the image was built.

2.2.3 Assigning Workspace for Link

The size of the workspace increment value given when Link is loaded will control the maximum symbol table size generated by Link as shown in the following table:

WORKSPACE INCREMENT	SYMBOL TABLE MAXIMUM
0 - 7	LINK will not run
8 - 15	32 kilobytes
16 - 31	64 kilobytes
32 - 63	96 kilobytes
64 - 95	128 kilobytes
96 - 127	256 kilobytes
128 - 255	1 megabyte
256 - or greater	4 megabytes

2.3 LINK INPUT/OUTPUT (I/O) FILES

Link requires the following I/O files:

- Object files containing the compiled source code.
- Task image file to which Link outputs the task image.
- Map file to which Link sends a listing of the establishment summary and, optionally, all external programs and their addresses.
- Log file which lists all Link commands issued and any Link generated diagnostic messages.
- Command file containing commands to Link.

The Link command file can be built by a command substitution system (CSS) procedure or built as a separate file that can be specified in the START command. If no Link command file is specified in the START command, Link accepts commands interactively from the terminal or console. The BUILD command for Link automatically allocates a file, if the file does not already exist, for the task image using the filename entered, followed by the extension corresponding to the type of image (TASK, OS, partial image) being built. The log file must be preallocated by the user. The user can optionally preallocate a map file. However, LINK will allocate the map file if it does not exist.

Table 2-1 lists the logical unit (lu) assignments that are made automatically by the Link commands.

TABLE 2-1 LOGICAL UNITS ASSIGNED BY LINK

LINK COMMAND	LOGICAL UNITS ASSIGNED	I/O FILE
INCLUDE/LIBRARY	1	Object
BUILD	2	Task Image
MAP	3	Link Map
START		
,COMMAND=	5	Link Command Input
	7	Link Command Output
,LOG=	6	Log
HELP	10	Link Help File

Link also assigns lu9 as needed for the temporary paging of its symbol table.

2.4 STARTING LINK

After Link is loaded into main memory, the system START command starts execution of the Link program and assigns the command and log devices.

Format:

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

Parameters:

COMMAND= fdl specifies the input device on which Link commands are entered. If this parameter is omitted, the default is the command input device (CON:). If CON: is interactive, all messages generated by Link are sent to CON:. If the command input device is batch, all Link messages are sent to the device specified by the LOG parameter.

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 disk file, it must have been previously allocated.

Functional Details:

After the linkage editor is started, the following message is displayed:

```
    PERKIN-ELMER OS/32 LINKAGE EDITOR 03-242 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 that 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 they are entered and have an immediate effect on how the image is to be built. Passive commands are executed during the build process, at which time Link processes them, making symbol table entries, etc. Although passive commands are not executed when entered, the order in which passive commands are encountered might have an effect on the image produced by Link. This is due to the order in which items are entered into Link's internal symbol table. Environment commands affect the link session instead of the image being built. Environment commands have no affect on the image being built, but do establish the environment.

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
DCMD	*			Enables execution of Link commands embedded in object modules. Enables the listing of embedded auxiliary processing unit (APU) comments to the log device in the Model 3200MPS System.
END	*			Terminates the linkage editor
ESTABLISH	*			Specifies the type of image to be built
EXTERNAL		*		Specifies the names of common block(s) to be externally visible from the partial image being built.
FFILE			*	Forward spaces a magnetic tape or contiguous file
HELP			*	Lists and describes all Link commands accepted by the current revision of Link.
INCLUDE	*			Specifies the object modules to be included in the image
LIBRARY		*		Specifies the object libraries to be searched for unresolved external references
LOCAL		*		Specifies entry points that are not to be visible from outside of the partial image being built
LOG			*	Enables logging all commands, messages, and maps to the log device
MAP		*		Generates a map when the image is built

TABLE 3-1 LINK COMMANDS (Continued)

COMMAND	TYPE			MEANING
	ACT	PAS	ENV	
NDCMD	*			Disables execution of Link commands embedded in object modules. Disables listing of embedded comments to the log device in the Model 3200MPS System.
NLOG			*	Disables logging of commands, messages, and maps to the log device
OPTION		*		Sets task and Link options
OVERLAY	*			Defines an overlay and a level for that overlay
PAUSE			*	Pauses the linkage editor
POSITION		*		Moves a common block into a specific overlay node
RESOLVE		*		Specifies a partial image that can be referred to by the task or image being built.
REWIND			*	Rewinds a magnetic tape or contiguous file
SEGMENT				Reserved for future definition
TITLE			*	Specifies a title for the Link map
VOLUME			*	Specifies the default volume to be used for all subsequent file descriptors (fds)
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:

$$\underline{\text{BF}}\text{ILE fd}\left[\left\{ ,\begin{matrix} \text{n} \\ \mathbf{1} \end{matrix} \right\}\right]$$

Parameters:

fd is the file descriptor of the device or file 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.

Example:

```
BF MAG1:,2
```

3.3 BUILD COMMAND

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

Format:

```
    BUILD fd
```

Parameters:

fd — is the file descriptor that is to receive the image. If the extension is omitted, the default extensions are:

.TSK for tasks

.IMG for partial images

.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, the following message is displayed:

```
    ENTER FILE DESCRIPTOR FOR IMAGE>
```

Enter the fd of the device to receive the image.

If a file with the filename specified already exists, Link will overwrite it automatically, without issuing any prompts.

By default, Link allocates a contiguous file for the image. Saving an image to a contiguous file is significantly faster than saving an image to an indexed file.

After the task is built, the Link maps are generated if the MAP command was entered. If the MAP command was not entered, the following message is displayed:

```
    MAP?>
```

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

- ENTER FILE DESCRIPTOR FOR MAP>

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

- SORTED BY ADDRESS?>

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

- CROSS REFERENCE?>

  If YES is entered, a cross-reference map is generated. This map lists all symbols in alphabetical order and the names of all object modules that reference each symbol.

- SORTED ALPHABETICALLY?>

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

  If NO is entered for all of these messages, only an establishment summary is generated. See Section 3.14.

After the BUILD command is executed, the linkage editor builds the image. To only generate a Link map without saving the task image to a file, specify NULL: as the fd to the BUILD command.

Examples:

```
    BUILD COM.IMG

    BUILD TASK

    BUILD TASK.TSK

    BUILD NULL:
```

NOTE

If Link is running in batch mode and cannot allocate the file, the build process is terminated.

3.4 DCMD COMMAND

The define command (DCMD) command is an active command that, when entered without parameters, enables execution of passive Link commands in common assembly language (CAL) object modules included in the image. This command, at the same time, enables listing of embedded comments to the input or log device. In a Model 3200MPS System, this command entered with parameters enables or suppresses listing of APU comments to the log device.

Format:

$$\underline{\text{DC}}\text{MD} \left[\begin{Bmatrix} \underline{\text{APUC}}\text{OMMENT} \\ \\ \underline{\text{NAPUC}}\text{OMMENT} \end{Bmatrix} \right]$$

Parameters:

APUCOMMENT	enables listing of APU comments to the log device.
NAPUCOMMENT	disables listing of APU comments to the log device. This is the default.

The DCMD command enables CAL and FORTRAN programs to contain passive Link commands that will be executed when the image is built. To embed passive Link commands in a CAL program, use the CAL DCMD as follows:

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

NOTE

This DCMD pseudo-op is not the same as the DCMD command described under format.

Example of CAL code containing embedded passive Link commands:

```
    MOD      PROG
             ENTRY  ENTRY
             EXTRN  EXTRNA
             EXTRN  EXTRNB
             EXTRN  EXENTRY
             DCMD   C'OPTION FLOAT'
             DCMD   C'MAP PR:,ALPHA'
             DCMD   C'*PATCH FOR SCR 1183, 1/24/83'
             DCMD   C'*APU MODULE MOD INVOKES SVC CALLS'
             PURE
    ENTRY    L      0,EXTRNA
             ST     0,EXTRNB
             BAL    13,EXENTRY
             SVC    3,0
             END
```

Embedded passive Link commands are treated as if they were part of the Link command sequence. Embedded LIBRARY commands are treated as if they were entered immediately before the BUILD command; all other embedded commands are treated as if they were entered after the INCLUDE command.

If a log device is specified in the START command, all embedded passive Link commands are output to the log device with a plus sign (+) in column 1.

The DCMD command entered without any parameters also enables listing of embedded general comments to the log device. These general comments can refer to patches applied to a particular compiler or other general comments the user does not want suppressed.

In a Model 3200MPS System, some language processors, such as CAL/32 and FORTRAN VII, generate APU information comments embedded in the object files of APU tasks. These APU comment lines always begin with an asterisk (*) and the letters APU. Listing or suppression of the APU comment lines is enabled by entering the DCMD command with the APUCOMMENT or NAPUCOMMENT parameter. If the APUCOMMENT parameter is entered, all comments, including the general comments, are displayed. If the NAPUCOMMENT parameter is entered, APU comments are suppressed, but the general comments are still displayed.

When the program above is linked, the log listing will be:

```
ES TA
INCLUDE MOD
BUILD MOD
```

If the DCMD command with no parameters is entered, the log listing will be:

```
DCMD
ES TA
INCLUDE MOD
+OPTION FLOAT
+MAP PR:, ALPHA
+*PATCH FOR SCR 1183, 1/24/83
BUILD MOD
```

If the DCMD command is entered with the APUCOMMENT parameter, the log listing will be:

```
ES TA
DCMD APUCOMMENT
INCLUDE MOD
+OPTION FLOAT
+MAP PR:, ALPHA
+*'PATCH FOR SCR 1183, 1/24/83'
+*APU 'MODULE MOD INVOKES SVC CALLS'
BUILD MOD
```

Only passive Link commands can be embedded in CAL object modules. If active or environment commands are embedded in CAL object modules, they will be ignored and this message will be output:

```
COMMAND NOT PERMITTED
```

Application users in a uniprocessor system can use the DCMD command with its parameters for developing a Model 3200MPS System.

If this command is not entered, all embedded passive Link commands are executed. To turn this feature off, use the NDCMD command explained in Section 3.15.

3.5 END COMMAND

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

Format:

```
END
```

Functional Details:

If a Link command sequence contains at least one INCLUDE command and an END command sequence before the BUILD command is entered, the following message is displayed:

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

Enter YES if the image is to be built. Enter NO if no image is to be built and the task is to be terminated. See Table 3-2 for the meaning of 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 that did not affect the building of the image.
2	An error occurred that affected the building of the image.
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 to be built and provides a package name to a multiple segment image. The three types of images that can be built are:

- task,
- operating system, and
- partial image

Format:

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

Parameters:

TASK specifies that a task image is to be built. If the ESTABLISH command or the parameters specifying the type of image are omitted, TASK is the default.

OS specifies that an operating system image is to be built.

IMAGE specifies that a partial image is to be built. A partial image is a collection of task segments that can be used by one or more separate tasks. A partial image has no user-dedicated location (UDL).

ACCESS= specifies the access privileges of the partial image, as follows:

- R - specifies that all tasks can read data within the partial image. Execution or modification of data is not allowed. If the ACCESS parameter is omitted, RE is the default.
- E - specifies that all tasks can execute code within the partial image.
- RE - specifies that all tasks can read data and execute code within the partial image. Modification of data is not allowed.
- RW - specifies that all tasks can read and modify data within the partial image. Execution of the data is not allowed.
- RWE - specifies that all tasks can read, modify, and execute data within the partial image.

ADDRESS= m0000 is the starting address of the partial image segment. This address is the bias address used to adjust relocatable addresses to create the real addresses in the partial image segment. The variable m is a hexadecimal number from 0 through BF. If this parameter is omitted, or ADDRESS=* is specified, the partial image segment becomes address-independent and can be assigned a different starting address by each task that refers to it. If relocatable addresses are located in an address-independent partial image segment, they are relocated as though ADDRESS=00000 was specified, and a warning message is issued.

NAME= specifies a package name for a multi-segmented task, partial image or operating system. If this parameter is not specified, the file descriptor in the BUILD command is used as the package name. Package names assigned by this parameter are independent of the names of the individual segments within a multi-segmented image.

package name is a filename.ext that identifies the partial image after it is loaded into main memory. This name is matched against the name specified by the tasks that will refer the partial image.

Functional Details:

If the ESTABLISH command is entered after active commands have been entered and before BUILD is entered, the following message is displayed:

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

Enter YES(Y) or NO(N). If YES is entered, the following message is displayed:

```
    ENTER FILE DESCRIPTOR FOR IMAGE>
```

After fd is entered, the image is built.

If NO is entered, no build is performed, and the following message is displayed:

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

Examples:

```
    ES OS
```

Establish an operating system image.

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

Establish a partial image with RE access privileges and a package name of SEG1.

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

Establish a reentrant library image with RE access privileges.

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

Establish a task common image with RW access privileges.

3.7 EXTERNAL COMMAND

The EXTERNAL command is a passive command that specifies the name of one or more common blocks in a partial image that can be referred to by tasks outside the partial image segment.

Format:

EXTERNAL common block name$_1$ $\left[,\ldots,\text{common block name}_n\right]$

Parameters:

common block name	is the name of a common block outside the partial image segment to which reference will be made.

Functional Details:

Common blocks are local to a partial image that is shared by other tasks 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:

$$\underline{\text{FF}}\text{ILE fd}\left[,\begin{Bmatrix}\text{n}\\ \text{1}\end{Bmatrix}\right]$$

Parameters:

fd is the file descriptor of the device or file 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.

Example:

```
FF MAG1:,2
```

3.9 HELP COMMAND

The HELP command provides a list of all Link commands accepted by the latest revision of Link. HELP also describes the syntax and function of each command.

Format:

```
     [mnemonic]
HELP [        ]
     [   *    ]
```

Parameters:

mnemonic — is the mnemonic for a Link command that is to be described by HELP.

* — lists all Link commands accepted by the latest revision of Link. If no parameter is specified, * is the default.

Functional Details:

If a log device has been specified in the START command for Link, HELP outputs all lists and descriptions of the Link commands to the log device.

Examples:

```
help
BF(ILE)          BU(ILD)          DC(MD)           EN(D)
ES(TABLISH)      EX(TERNAL)       FF(ILE)          H(ELP)
IN(CLUDE)        LI(BRARY)        LOC(AL)          LOG
MA(P)            ND(CMD)          NL(OG)           OP(TION)
OV(ERLAY)        PA(USE)          PO(SITION)       REW(IND)
RES(OLVE)        SEG(MENT)        TI(TLE)          VO(LUME)
WF(ILE)
FOR HELP ON ANY OF THE ABOVE COMMAND MNEMONICS, TYPE HELP
MNEMONIC
```

```
help map
MA(P) : This command is a passive command that displays a map
   containing the names and addresses of symbols.

SYNTAX:  MA(P) [<FD>] [,AL(PHABETIC)] [,AD(DRESS)] [,XR(EF)]

WHERE: <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.  If no log device has been specified, the maps are
   output to the command device, in interactive mode, and to
   device PR:  in batch mode.

The 'ALPHABETIC' parameter specifies that the map is to contain
    all symbols in alphabetic order.

The 'ADDRESS' parameter specifies that the map is to contain all
    symbols in address order.

The 'XREF' parameter specifies that the map is to contain all the
    names of the modules that reference each symbol, and the name
    of the module in which the symbol is defined.
```

3.10 INCLUDE COMMAND

The INCLUDE command is an active command that specifies a file containing object modules and the specific names of object modules that are to be included in the image. The INCLUDE command can be entered any number of times to include object modules from many different files.

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 lul 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. The first character of this string must be alphabetic if "*" or "-" is not specified. 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. The first character of this string must be alphabetic if "*" or "-" is not specified. If this parameter is omitted, module_1 is included. If an asterisk (*) or hyphen (-) with no module name is specified, all modules starting with module_1 to the end of the file are included.

Functional Details:

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

Object code modules specified in this command can consist only of the object code defined in Appendix D. Appendix D lists each loader item accepted by Link and describes what data may follow it.

Examples:

```
INCLUDE LIBRARY.OBJ
```

Include all modules in fd LIBRARY.OBJ.

```
INCLUDE LIBRARY.OBJ, 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, then module SIXTH, and module TENTH through the end of LIBRARY.OBJ

3.11 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 in which they are specified.

Format:

LIBRARY fd_1 $\left[,\ldots,fd_n\right]$

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

When writing programs in high level languages such as FORTRAN or PASCAL, be sure to specify all user libraries before specifying a standard Perkin-Elmer Run-Time Library (RTL). This ensures that each user library routine gets resolved against the standard RTL.

Also, remember that the domain of a LIBRARY command is the entire Link command sequence; i.e., its domain is not restricted to any overlay in which it might be placed. Only the order in which the libraries are specified is significant to Link.

When a program is linked, external references that were not resolved by the INCLUDE and RESOLVE commands are matched against the library(ies) entry points. All external references generated from modules included from the library cause the library modules that resolve those external references to be included, regardless of the order of the modules within the library.

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

Nonlinking 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 either of 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 matched against an unresolved 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 are resolved against the list of unresolved, standard, and weak externals.

Example:

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

Specifies the user RTL and FORTRAN RTLs to be searched.

3.12 LOCAL COMMAND

The LOCAL command is a passive command that specifies one or more entry points in a partial image that can be referred to only by external references within that partial image. This command is valid only when establishing a partial image.

Format:

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

Parameters:

entry point	is a 1- to 8-character alphanumeric string specifying the entry point name. The first character of the string must be alphabetic.

Functional Details:

When a partial image is built, all entry points within that image can be referred to by tasks external to the partial image, unless the entry points are made local to that partial image by the LOCAL command.

Example:

```
LOC ENTRY1
```

3.13 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:

```
    LOG PR:
```

Commands, messages, and maps are to be sent to PR:

```
    LOG M300:LOGFILE
```

Commands, messages, and maps are to be sent to the file LOGFILE on volume M300:.

3.14 MAP COMMAND

The MAP command is a passive command that generates an establishment summary and a map or maps containing the names and addresses of program symbols.

Format:

```
MAP [fd] [,ALPHABETIC] [,ADDRESS] [,XREF]
```

Parameters:

fd — is the file descriptor of the file or 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. If the specified file discriptor is not preallocated, Link will allocate an indexed file (logical record length 120) by that name for the map.

ALPHABETIC — specifies that the map is to contain all symbols in alphabetical order.

ADDRESS — specifies that the map is to contain all symbols in ascending address order.

XREF — specifies that the map is to contain all the names of the object modules that reference each symbol, and the name of the module to which the symbol is defined.

If none of these parameters are specified, only the establishment summary is generated.

Functional Details:

The Link maps generated by the MAP command tell the user how the image is structured and where each subprogram and RTL routine is referenced by the program. These maps can be used to determine whether a user-defined or Perkin-Elmer standard library routine has been referred to or redefined by the program.

Three types of Link maps can be generated: alphabetic, address, and cross-reference. The Link establishment summary precedes the Link maps. Figure 3-1 shows an example of the Link establishment summary. Numbered items contained in this summary are identified as follows:

NUMBER LIST ITEM

1 File descriptor of image file

2 Number of records in image file

3 Image file and address space

4 Task options set by the Link OPTION command or by Link default

5 Node map listing node characteristics as follows:

- LEVEL - indicates the overlay level for the node. (0 indicates that the node is not located in an overlay area.)
- NAME - indicates the name of each segment within the node.
- LENGTH - is a hexadecimal number indicating the length of each segment in bytes.
- PURE - is a hexadecimal number indicating the number of bytes comprising a sharable task segment.
- IMPURE - is a hexadecimal number indicating the number of bytes comprising a nonsharable task segment.
- COMMON - is a hexadecimal number indicating the number of bytes comprising a common data area.
- TABLES - is a hexadecimal number indicating the number of bytes of executable code set aside for Link overlay tables.

6 Virtual address map listing the name, size, address boundaries, and access privileges of each segment. Size is expressed as a decimal number in 1kb (1,204-byte) units.

Following the establishment summary are the symbol maps specified by the MAP command. If no map options are specified, the MAP command outputs an establishment summary only. Symbol maps list data areas and all subprograms and RTL routines called by the program. If the ALPHABETIC option is chosen, symbols and their corresponding nodes are arranged alphabetically as shown in Figure 3-2. If the ADDRESS option is chosen, symbols are arranged according to their addresses within each node as shown in Figure 3-3. The address map also lists each overlay area separately in the order each is defined. As shown in Figure 3-4, if the XREF option is chosen, a cross-reference map is produced. This map arranges symbols according to how they are referred to by the program. For example, in Figure 3-4 the symbol ENTRY is defined by the module INCLUDE while INCLUDE refers to GRABBED and SPACE, which are, in turn, defined by GRABIT.

All of the symbol maps precede each symbol name with a single letter indicating the type of subprogram, routine, or data area named by the symbol. C indicates a common data area. D indicates the name of a data entry point; E is a standard entry point name. P indicates the name of a program.

Following the address of each symbol name in the alphabetic address map are the letters P, I, or A. P indicates that the symbol is located in a pure segment. I indicates the symbol is located in an impure segment, while A indicates an absolute data area.

Examples:

```
MAP PR:
```

An establishment summary is to be output to the printer.

```
MAP MAPFILE,ADDR
```

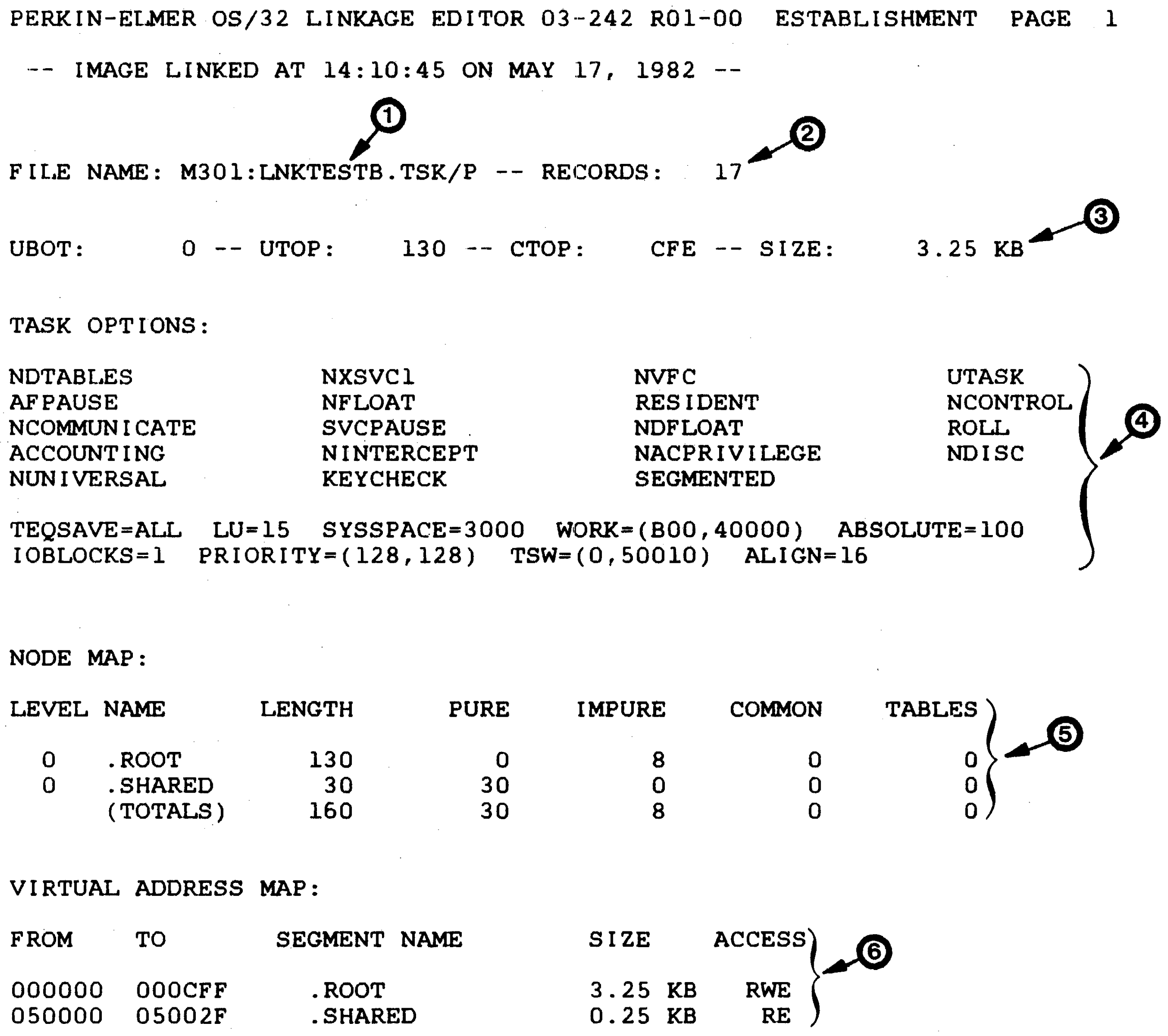
An establishment summary and address map are to be output to the file named MAPFILE.

```
MAP ,ALPHA
```

An establishment summary and alphabetic map are to be output to the log device.

```
MAP PR:,XREF,ALPHA
```

An establishment summary and alphabetic and cross-reference maps are to be output to the printer.



```
PERKIN-ELMER OS/32 LINKAGE EDITOR 03-242 R01-00  ESTABLISHMENT  PAGE  1

 -- IMAGE LINKED AT 14:10:45 ON MAY 17, 1982 --


FILE NAME: M301:LNKTESTB.TSK/P -- RECORDS:   17


UBOT:       0 -- UTOP:    130 -- CTOP:    CFE -- SIZE:     3.25 KB


TASK OPTIONS:

NDTABLES             NXSVC1               NVFC                 UTASK
AFPAUSE              NFLOAT               RESIDENT             NCONTROL
NCOMMUNICATE         SVCPAUSE             NDFLOAT              ROLL
ACCOUNTING           NINTERCEPT           NACPRIVILEGE         NDISC
NUNIVERSAL           KEYCHECK             SEGMENTED

TEQSAVE=ALL  LU=15  SYSSPACE=3000  WORK=(B00,40000)  ABSOLUTE=100
IOBLOCKS=1  PRIORITY=(128,128)  TSW=(0,50010)  ALIGN=16


NODE MAP:

LEVEL NAME        LENGTH        PURE      IMPURE      COMMON      TABLES

  0   .ROOT          130           0           8           0           0
  0   .SHARED         30          30           0           0           0
      (TOTALS)       160          30           8           0           0

VIRTUAL ADDRESS MAP:

FROM     TO        SEGMENT NAME          SIZE      ACCESS

000000   000CFF     .ROOT                3.25 KB    RWE
050000   05002F     .SHARED              0.25 KB     RE
```

Figure 3-1 Example of Link Establishment Summary

```
PERKIN-ELMER OS/32 LINKAGE EDITOR 03-242 R01-00  ALPHABETIC MAP  PAGE  1

 -- IMAGE LINKED AT 14:10:45 ON MAY 17, 1982 --


SYMBOL     -- NODE      -- ADDRESS        SYMBOL     -- NODE      -- ADDRESS

E-ENTRY      .SHARED     050010-P        E-GRABBED    .SHARED      050000-P
P-GRABIT     .SHARED     050000-P        P-INCLUDE    .SHARED      050010-P
E-SPACE      .ROOT       000110-I
```

Figure 3-2 Example of Link Alphabetic Map

```
PERKIN-ELMER OS/32 LINKAGE EDITOR 03-242 R01-00   ADDRESS MAP     PAGE     1

 -- IMAGE LINKED AT 14:10:45 ON MAY 17, 1982 --


NODE: .ROOT      - LEVEL:  0 - ADDRESS:      0 - SIZE:   130 - PARENT:

SYMBOL    --    ADDRESS        SYMBOL    --    ADDRESS        SYMBOL    --    ADDRESS

P-GRABIT        000110-I       E-SPACE         000110-I       P-INCLUDE       000120-I

NODE: .SHARED    - LEVEL:  0 - ADDRESS:  50000 - SIZE:    30 - PARENT:

SYMBOL    --    ADDRESS        SYMBOL    --    ADDRESS        SYMBOL    --    ADDRESS

P-GRABIT        050000-P       E-GRABBED       050000-P       P-INCLUDE       050010-P
E-ENTRY         050010-P
```

Figure 3-3 Example of Link Address Map

```
PERKIN-ELMER OS/32 LINKAGE EDITOR 03-242 R01-00   CROSS-REFERENCE PAGE     1

 -- IMAGE LINKED AT 14:10:45 ON MAY 17, 1982 --


SYMBOL       DEFINED        REFERENCED BY

E-ENTRY      INCLUDE
E-GRABBED    GRABIT         INCLUDE
E-SPACE      GRABIT         INCLUDE
```

Figure 3-4 Example of Link Cross-Reference Map

3.15 NDCMD COMMAND

NDCMD is an active command that disables execution of passive Link commands embedded in object modules included in the image. This command also suppresses listing of general comments to the log device.

Format:

```
    NDCMD
```

Functional Details:

The DCMD command reenables execution of passive Link commands embedded in object modules and reenables listing of embedded general comments (see Section 3.4).

3.16 NLOG COMMAND

The no log (NLOG) command is an environment command that terminates the logging process.

Format:

```
    NLOG
```

Functional Details:

Logging can be restarted by the LOG command explained in Section 3.13.

3.17 OPTION COMMAND

The OPTION command is a passive command that sets the task options that will be in effect during task execution.

CAUTION

WHEN THE TASK IS LOADED UNDER MTM, CERTAIN MTM CONFIGURATIONS CAN OVERRIDE THE TASK OPTIONS SET BY THE OPTION COMMAND. SEE THE OS/32 MULTI-TERMINAL MONITOR (MTM) REFERENCE MANUAL FOR MORE INFORMATION.

Format:

```
OPTION [ABSOLUTE={ a    }] [,{NACCOUNTING}] [,{ACPRIVILEGE }]
                 { X100 }     {ACCOUNTING }     {NACPRIVILEGE}

       [,ALIGN={value}] [,{APCONTROL }] [,{APMAPPING }]
               { 16  }     {NAPCONTROL}     {NAPMAPPING}

       [,{APUONLY }] [,{COMMUNICATE }] [,{CONTROL }] [,{DFLOAT }]
         {NAPUONLY}    {NCOMMUNICATE}    {NCONTROL}    {NDFLOAT}

       [,{DISC }] [,{DTABLES }] [,ENTRY=(main entry,debug entry)]
         {NDISC}    {NDTABLES}

       [,{DTASK}] [,{FLOAT }] [,{INTERCEPT }] [,IOBLOCKS={b}]
         {ETASK}    {NFLOAT}    {NINTERCEPT}            {1}
         {UTASK}

       [,{NKEYCHECK}] [,LU={lu}] [,LPU={lproc}] [,{NAFPAUSE}]
         {KEYCHECK }       {15}        {  0  }     {AFPAUSE }

       [,PRIORITY=([{ipri}] [,{mpri}])] [,{RESIDENT }] [,{NROLL}]
                    {128 }     {128 }      {NRESIDENT}    {ROLL }

       [,{SEGMENTED }] [,SYSSPACE={   decimal value   }]
         {NSEGMENTED}             {Xhexadecimal value}
                                  {      X3000        }

       [,{NSVCPAUSE}] [,TSW=([{status}] [,{st adr}])]
         {SVCPAUSE }           {  *   }     {  0   }
                               {  0   }

       [,TEQSAVE={NONE   }] [,{UNIVERSAL }] [,{VFC }] [,VFD=fd]
                 {PARTIAL}     {NUNIVERSAL}     {NVFC}
                 {ALL    }

       [,VTM={n}] [,WORK=({nominal workspace},{maximum workspace})]
             {4}           {        *        } {                 }
                           {       X50       } {      X40000     }

       [,{XSVC1 }]
         {NXSVC1}
```

Parameters:

ABSOLUTE reserves a specified number of bytes of main storage for absolute data. If this parameter is not specified, Link reserves 256 (X'100') bytes of main storage for absolute data.

a is a 1- to 6-digit hexadecimal number specifying the number of bytes of main storage that are to be reserved by Link for absolute data. X100 is the default.

NACCOUNTING turns off the accounting facility for the task if accounting was enabled at system generation (sysgen). If this parameter is not specified, ACCOUNTING is the default.

ACCOUNTING turns on the accounting facility for the task if accounting was enabled at sysgen. The accounting facility collects task related data including the task's roll-time, wait-time, I/O transfer count, and the end of task code. If the accounting facility was not specified at sysgen and ACCOUNTING is specified, no accounting data will be collected. If this parameter is specified and the accounting facility was specified at sysgen, accounting data is collected.

ACPRIVILEGE provides a user task (u-task) with extended file access privileges as follows:

- a u-task can specify an account number instead of a file class for all file management functions

- a u-task can turn off the KEYCHECK option, if set.

  If this parameter is not specified, NACPRIVILEGE is the default. This option has no affect on executive tasks (e-tasks) or diagnostic tasks (d-tasks).

WARNING

IF A TASK LOADED FROM THE SYSTEM CONSOLE IS TO ACCESS FILES UNDER AN ACCOUNT NUMBER OTHER THAN 0, ACPRIVILEGE MUST BE SPECIFIED FOR THAT TASK.

NACPRIVILEGE specifies that a u-task has no extended file access privileges. If the extended file access privilege option is not specified, NACPRIVILEGE is the default. This option has no affect on e-tasks or d-tasks.

ALIGN specifies the byte boundary for aligning object modules within segments. Unused bytes between aligned modules are filled with zeros. If this parameter is omitted, all object modules begin on the next highest quadword boundary (value=16), unless already on such a boundary.

value is a decimal number expressed as an even power of two in the range from 4 to 2,048. If this parameter is not specified, 16 bytes (one quadword) is the default boundary alignment value for all object modules.

APCONTROL specifies that the task can obtain APU control privileges. This option is valid for a Model 3200MPS System only. Control of an APU by a task is accomplished through the supervisor call (SVC) 13 parameter block. See the OS/32 Supervisor Call (SVC) Reference Manual. If this option is omitted, NAPCONTROL is the default.

NAPCONTROL specifies that the task cannot obtain APU control privileges. This option is valid for a Model 3200MPS System only and is the default.

APMAPPING specifies that the task can obtain APU mapping privileges. This option is valid for a Model 3200MPS System only. If this option is omitted, NAPMAPPING is the default.

NAPMAPPING specifies that the task cannot obtain APU mapping privileges. This option is valid for a Model 3200MPS System only and is the default.

APUONLY specifies that the task can execute on an APU only. Any transfer of control from the APU to the CPU causes the task to pause. This option is valid on a Model 3200MPS System only. If this option is omitted, NAPUONLY is the default.

NAPUONLY specifies that the task can execute on an APU or a CPU. This option is valid for a Model 3200MPS System only and is the default.

COMMUNICATE specifies that the task can perform the SVC 6 intertask communication functions. If this parameter is not specified, the task cannot communicate with other tasks.

NCOMMUNICATE prevents the task from issuing an SVC 6 for intertask communication. If the intertask communication option is not specified, NCOMMUNICATE is the default.

CONTROL specifies that the task can perform the SVC 6 intertask control functions. If this parameter is not specified, the task cannot issue an SVC 6 to control the execution of another task.

NCONTROL prevents the task from issuing an SVC 6 for intertask control. If the intertask control option is not specified, NCONTROL is the default.

DFLOAT specifies that a task can execute double precision floating point instructions. If this parameter is not specified, the task cannot execute double precision floating point instructions.

NDFLOAT prevents the task from executing double precision floating point instructions. If the double precision option is not specified, NDFLOAT is the default.

DISC is the bare disk I/O privilege option. This option allows a u-task or diagnostic task (d-task) to bypass the file manager and directly assign I/O requests to a disk device. If the disk is marked online, only assignments for shared read only (SRO) are allowed. Any other assignment is rejected, and a privilege error message is output. If the disk is marked offline, all access privileges are allowed. See the OS/32 Supervisor Call (SVC) Reference Manual for a description of the access privileges. This option has no affect on e-tasks, since they have bare disk privileges by definition.

NDISC

prevents u- and d-tasks from directly assigning I/O requests to a disk device. If the bare disk I/O privilege is not specified, NDISC is the default. This option has no affect on e-tasks.

NOTE

> If a task is loaded under MTM and DISC is not specified, or DISC is specified but the task loader has the ETASK option disabled, the image is loaded without the bare disk I/O privilege.

DTABLES

causes the task loader to build the appropriate debug tables for the symbolic debug data contained in the image. This option also increases the number of logical units used by the task, by one. However, LU=15 still appears on the Link map. If DTABLES is not specified, debug tables are not built.

NDTABLES

prevents the task loader from building debug tables so that all debug data contained in the image is discarded. If this option is not specified, debug tables are built.

ENTRY

specifies the name of an entry point in the root node or the debug task where execution of the task image is to begin. If this option is omitted, the entry point is the starting address specified when the task was assembled or compiled.

main entry

is a standard entry point known to Link while the image is being built. Standard entry points include those for partial images but exclude data entry (DNTRY) points. If only the main entry is specified, omit the parentheses.

debug entry

is the name of the entry point to the debug task. ENTRY places the debug entry point into the loader information block (LIB) for the task. If the main entry is specified with the debug entry, the main entry is moved to the symbolic debug data table.

DTASK

specifies that a d-task image is to be built. A d-task has its own virtual address space but can execute privileged instructions. If no task type parameter is specified, UTASK is the default.

ETASK specifies that an e-task image is to be built. An e-task contains only positional-independent pure and impure code and cannot reference partial images. An e-task can execute privileged instructions. If no task type parameter is specified, UTASK is the default.

UTASK specifies that a u-task image is to be built. A u-task cannot execute privileged instructions unless the task is linked with option ACPRIVILEGE or is running under MTM with specified privileges. MTM allows a user to specify privileges for an account. Once specified, all users on that account are allowed to use those privileges. See the OS/32 Multi-Terminal Monitor (MTM) System Planning and Operator Reference Manual. If no task type parameter is specified, UTASK is the default.

FLOAT specifies that the task can execute single precision floating point instructions. If FLOAT is not specified, the task cannot execute single precision floating point instructions.

NFLOAT prevents the task from executing single precision floating point instructions. If the single precision option is not specified, NFLOAT is the default.

INTERCEPT specifies that the task can intercept an SVC issued by another task before the SVC is processed by the operating system. If this option is not specified, the task cannot intercept an SVC issued by another task. For more information on SVC interception, see the OS/32 System Level Programmer Reference Manual.

NINTERCEPT prevents the task from intercepting an SVC issued by another task. If the SVC interception option is not specified, NINTERCEPT is the default.

IOBLOCKS specifies the maximum number of I/O blocks assigned to the task. Each I/O control block can contain one queued I/O request. If this option is not specified, Link automatically assigns one I/O control block to the task.

b is a decimal number from 1 through 65,535 indicating the number of I/O blocks assigned to the task.

NKEYCHECK prevents the operating system from checking the file protection keys of a u- or d-task having accounting or bare disk I/O privileges. If this option is not specified, the operating system will check the file protection keys for all privileged u-tasks. NKEYCHECK has no affect on e-tasks.

KEYCHECK causes the operating system to check the file protection keys of a u- or d-task having accounting or bare disk I/O privileges. If the file protection option is not specified, KEYCHECK is the default. KEYCHECK has no affect on e-tasks.

LU specifies the maximum number of logical units that can be assigned to the task. If this option is not specified, the maximum number of logical units is 15.

lu is a decimal number from 0 through 255.

LPU specifies the logical processing unit (LPU) used to direct tasks to processors. Each task is assigned an LPU. Each LPU is logically mapped to a processor. Assignment of a particular LPU number results in the assignment of that task to the associated processor. The default assignment is zero, which specifies execution on the CPU. This option is valid on a Model 3200MPS System only.

lproc specifies the LPU that the task is to be assigned to. Legal values can range from decimal zero to the maximum number of LPUs present in the system (MAXLPU) up to maximum of 255. MAXLPU is a sysgen parameter. See the System Generation/32 (SYSGEN/32) Reference Manual. This option is valid on a Model 3200MPS System only.

NAFPAUSE allows task execution to continue after an arithmetic fault occurs. If NAFPAUSE is not specified, task execution is suspended after an arithmetic fault.

AFPAUSE suspends task execution after an arithmetic fault occurs. If the NAFPAUSE fault option is not specified, AFPAUSE is the default.

PRIORITY specifies the initial and maximum priorities of the task. If this option is not specified, both the initial and maximum task priorities are 128. See the OS/32 Operator Reference Manual for an explanation of priority.

ipri is a decimal number from 11 through 254 indicating the initial task priority. The initial priority must be less than or equal to the specified maximum priority (mpri). If ipri is not specified, the default is 128.

mpri is a decimal number from 11 through 254 indicating the maximum priority of the task. If mpri is not specified, the maximum priority is 128 (the value specified for the initial priority).

RESIDENT specifies that the task is to remain in main memory after task execution is terminated. The task can then be restarted by the operator without issuing an OS/32 LOAD command. If this option is not specified, the task will be removed from memory after task termination.

NRESIDENT specifies that the task is to be removed from main memory after task execution is terminated. If the RESIDENT option is not specified, NRESIDENT is the default.

NROLL prevents the task from being rolled in and out of main memory during task execution. If this option is not specified, the task can be rolled during execution.

ROLL specifies that the task can be rolled in and out of memory during task execution. If the NROLL option is not specified, ROLL is the default.

SEGMENTED specifies that the pure segment of a u- or d-task can be shared when more than one copy of the u-task is loaded. If this option is not specified, the pure segment cannot be shared. SEGMENTED is incompatible with OPTION ETASK.

NSEGMENTED specifies that the pure segment of a u- or d-task cannot be shared when more than one copy of the u- or d-task is loaded. If the SEGMENTED option is not specified, NSEGMENTED is the default. NSEGMENTED is incompatible with OPTION ETASK.

SYSSPACE specifies the maximum amount of system space that a task can use during execution. If this option is not specified, the maximum system space that can be used is 12,288 (X3000) bytes.

decimal value is a 1- to 7-digit decimal number specifying the maximum amount of system space.

hexadecimal value is a 1- to 6-digit hexadecimal number preceded by an X specifying the maximum amount of system space.

NSVCPAUSE specifies that SVC 6 is treated as a no-operation (NOP) (applies to .BG tasks only). If a background task issues an SVC 6, the operating system ignores that call and continues execution of the task. If this option is not specified, the operating system pauses the execution of a background task that issues an SVC 6.

SVCPAUSE specifies that SVC 6 is treated as an illegal SVC (applies to .BG tasks only). If an SVC 6 is issued by a background task, the operating system pauses execution of that task. If the SVC 6 PAUSE option for background tasks is not specified, SVCPAUSE is the default.

TSW sets the task status and starting address fields of the task status word (TSW) in the LIB. An OR operation is performed on the status field in the LIB before the TSW is loaded into the final TSW for the task image. This option overrides any starting address specified by ENTRY.

status is a 1- to 8-digit hexadecimal number indicating the initial setting of the status field of the TSW in the LIB. If the asterisk (*) is specified, the current TSW is reset to zero. If status is not specified, the initial setting of the status field is zero.

st adr is a 1- to 6-digit hexadecimal number indicating the starting address for the task. This address overrides the starting address specified when the task was assembled or compiled as well as any starting address specified by the ENTRY option.

TEQSAVE informs the operating system whether or not the register contents should be saved and restored when the task enters or exits a task event service routine. The parameters of this option are:

- NONE - specifies that no register contents are saved and restored by OS/32 when the task enters or exits a task event service routine.
- PARTIAL - specifies that only the register contents that are used by the task event service routine are saved and restored when the task enters or exits the routine.
- ALL - specifies that all register contents are saved by OS/32 when the task enters or exits a task event service routine.

If this option is not specified, ALL is the default.

UNIVERSAL allows a task to communicate with all the other tasks in the system. If this option is not specified, a task can only communicate with other tasks having the same group ID as the task.

NUNIVERSAL specifies that a task can communicate with only those tasks in the system having the same group ID as the task. If the universal communication option is not specified, NUNIVERSAL is the default.

VFC turns on vertical forms control (VFC) for all task I/O operations. If this option is not specified, VFC is turned off for all task I/O operations.

NVFC turns off VFC for all I/O operations. If the VFC option is not specified, NVFC is the default.

NOTE

A task can override the NVFC and VFC options for specific devices or I/O operations by issuing the appropriate SVC 1 or SVC 7. See the OS/32 Supervisor Call (SVC) Reference Manual for more information on using SVC 1 and SVC 7 for VFC.

VFD specifies the secondary storage file for a virtual task. If this option is not specified, VTM will allocate a temporary file at run-time.

fd

is a file descriptor for a contiguous file that must occupy a minimum of CTOP/256 minus 255 sectors (plus 256 sectors if fd is the task image file). If the fd is the task image file itself, the task image is destroyed at run-time.

WARNING

IF OPTION VFD=fd IS SPECIFIED, MULTIPLE COPIES OF THE SAME TASK IMAGE CANNOT BE RUN.

VTM

specifies that a virtual task image is to be built.

n

is a decimal number from 2 through 127 specifying the number of resident 64kb working pages available for task memory management. If n is not specified, the default is 4.

WORK

specifies the number of bytes of main memory that can be added to the root node by the LOAD command for task workspace.

NOTE

Hexadecimal numbers specified by the WORK option must be preceded by an X; e.g., X40000.

nominal workspace

is a 1- to 6-digit hexadecimal or 1- to 7-digit decimal number indicating the workspace to be added if the workspace parameter in the LOAD command is not specified. If nominal workspace is not specified by the WORK option, 80 bytes (X50) will be added by LOAD.

The nominal workspace value is added to any nominal workspace values specified by previous OPTION WORK= commands to obtain the total nominal workspace.

If an asterisk (*) is specified, the nominal workspace is reset to zero. If only nominal workspace is specified, the parentheses are not required.

maximum workspace

is a 1- to 6-digit hexadecimal or 1- to 7-digit decimal number indicating the maximum amount of workspace that can be added by the LOAD command. If the maximum workspace is not specified, 256K (X40000) is the maximum number of bytes that can be added. The maximum workspace value is added to the maximum workspace values specified by previous OPTION WORK= commands to obtain the total maximum workspace.

XSVC1

indicates that if the task issues an SVC 1 with bit 7 of the function code set, the options specified by the SVC 1 extended option field are to be executed for all drivers which use this field. If XSVC1 is not specified, an SVC 1 with bit 7 set performs an image I/O transfer. See the OS/32 Supervisor Call (SVC) Reference Manual for more information on the SVC 1 function code and extended options.

NXSVC1

indicates that if the task issues an SVC 1 with bit 7 of the function code set, an image I/O transfer is performed. If the XSVC1 option is not specified, NXSVC1 is the default. See the OS/32 Supervisor Call (SVC) Reference Manual for more information on the SVC 1 function code and extended options.

Examples:

```
OPTION ACPRIVILEGE,NKEYCHECK,ALIGN=4,
       DFLOAT,LU=10,PRIORITY=(,200),
       SYSSPACE=X4000,VFC,XSVC1,
       WORK=(X100,X1000)
```

In this example, the task is to be linked as a u-task with extended file access privileges and without key checking. All object modules will be aligned to the nearest fullword boundary. The task can execute double precision floating point instructions and assign up to ten logical units. Maximum task priority is 200; initial task priority is 128. VFC is in effect for all I/O operations. The options specified by the SVC 1 extended option field are to be executed for all drivers that use this field. The task can be loaded with a maximum workspace of 4,096 bytes. If workspace is not specified in the LOAD command, the task will be loaded with 256 bytes. Note that X precedes the hexadecimal numbers in the WORK option. Maximum system space that can be used by this task is 16,384 bytes.

```
    OPTION DTABLES,ENTRY=(,DEBUG32)
```

In this example, the u-task is to be debugged using DEBUG/32. DTABLES builds the required debug tables needed to run DEBUG/32 while ENTRY specifies the name of the entry point to the debug task.

```
    OPTION INTERCEPT,TEQSAVE=PARTIAL
```

This example shows the task options that apply to a u-task that is to be linked with the SVC interception software. INTERCEPT allows the u-task to intercept an SVC of another task. TEQSAVE=PARTIAL indicates that all register contents used by the task event service routine are to be saved and restored. See the OS/32 System Level Programmer Reference Manual for more information on SVC interception and the task event service routine.

```
    OPTION VTM=5,VFD=PROG1.VTM
```

This example shows the task options that apply when a u-task is to run under the virtual memory manager. See Chapter 5. VTM specifies that a virtual image is to be built; VFD specifies that PROG1.VTM is to be used as a secondary storage file by the virtual task.

```
    OPTION FL,RES,LU=10,WORK=X3000,TSW=(,B020),APC,APM
```

This example shows the task options that can apply when the task is to run on the APU of a Model 3200MPS System. The task can execute single precision floating point instructions; is resident; has a maximum of 10 logical units that can be assigned to it; has a maximum workspace of X3000 bytes; has a starting address field of XB020 in the LIB; can obtain APU control privileges, and APU mapping privileges in a multiprocessor system. The APC and APM options are valid on a Model 3200MPS System only.

3.18 OVERLAY COMMAND

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

Format:

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

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 no more than 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. Overlayed 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 Supervisor Call (SVC) Reference Manual.

Example:

```
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
```

3.19 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 OS/32 CONTINUE command.

3.20 POSITION COMMAND

The POSITION command is a passive command that repositions common blocks into a node closer to the root segment than Link would normally position them.

Format:

$$\text{\underline{PO}SITION \underline{CO}MMON=}\left\{\begin{array}{c}\text{name}\\ (\text{name}_1,\ldots,\text{name}_n)\\ *\end{array}\right\}\left[\text{,}\underline{\text{TO}}\text{=}\left\{\begin{array}{c}\text{nodename}\\ \text{.ROOT}\end{array}\right\}\right]$$

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:

Normally, the placement of a common block within an overlayed task is determined by placement of the locations that refer to the block. A blank common is always positioned in .ROOT. A named common block, however, is initially positioned by Link no closer to the root than any particular reference to the block.

There are two consequences to this positioning policy. The first is that named common blocks are initialized each time an overlay is fetched from disk. The second consequence is that more than one copy of a common entity can exist on separate paths in the program; i.e., two or more overlays can have their own separate and private copies of a common entity. These copies could then contain different values.

Example:

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

3.21 RESOLVE COMMAND

The RESOLVE command is an active command that specifies the name of a partial image to be referred to by the task image. The partial image can be a global entity generated at the console by the OS/32 TCOM command, a sharable segment created by Link R00, or a partial image created by Link R01.

Format:

```
RESOLVE     [fd] [,NAME=package name]

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

Parameters:

fd — is the file descriptor of the partial image. If fd is not specified, the default partial image is the global task common defined by the TCOM command. If the file extension for a partial image created by Link R01 is not specified, the default extension is .IMG. Because the default extension for sharable segments created by Link R00 is .SEG, the file extension should be specified when these segments are resolved.

NOTE

> Link cannot get the size of a task common segment defined by TCOM from an image file; therefore, when the partial image is a global task common, the size of the partial image must be specified by the SIZE or STRUCTURE parameter in the RESOLVE command.

NAME= specifies the package name of the partial image. If this parameter is omitted, fd must be specified, and the default package name is the package name assigned to the partial image when it was established. When the task is loaded, the package name is matched against the names of any partial images already in main memory. If a partial image with the specified package name is not found in memory when the task is loaded, the package name is converted into an fd which is then used to locate and load a partial image.

package name is a filename.ext that identifies the partial image after it is loaded into memory. This name is matched against either the name of the global entity specified by TCOM or the package names of sharable segments or partial images.

ACCESS= specifies the access privilege of the partial image as follows:

R specifies that the task can read data within the partial image. Execution or modification of data is not allowed.

E specifies that the task can execute code within the partial image but cannot read or modify data within the image.

RE specifies that the task can read data and execute code within the partial image. Modification of data is not allowed. If the ACCESS= parameter is omitted, the default is RE.

RW specifies that the task can read and modify data within the partial image. Code execution is not allowed.

RWE specifies that the task can read and modify data and execute code within the partial image.

ADDRESS= m0000 is the starting address of the partial image. If the RESOLVE command specifies an fd for a partial image that is not address-independent, the specified address must match the address specified in the LIB of the partial image. If ADDRESS= is not specified, and the address was not specified when the partial image was established, Link automatically assigns an address to the partial image. The variable m is a hexadecimal number in the range from 0 through BF.

STRUCTURE= structures task common blocks within the partial image specified by fd. If fd is not specified, this parameter is used to structure global task common defined by the TCOM command.

$\text{name}_1 \ldots \text{name}_n$ is an 8-character alphanumeric string specifying the name of the task common block to be structured.

$\text{size}_1 \ldots \text{size}_n$ is a 1- to 6-digit hexadecimal number or a 1- to 7-digit decimal number specifying the length in bytes of the task common block. (Hexadecimal numbers must be preceded by an X; e.g., XF0.) This number must be greater than or equal to the size of the task common block specified by the program. If this number is smaller than the size specified by the program, Link outputs a warning message and uses the size specified by the program. The program size is also used if this parameter is omitted.

SIZE= specifies the minimum and maximum number of bytes of main memory that the partial image can occupy. If SIZE= and fd are not specified, the default size of the partial image is that specified by the STRUCTURE parameter. If SIZE is not specified but fd is, the default size of the partial image is the size obtained form the LIB of the partial image specified by fd.

min is a 1- to 6-digit hexadecimal number or a 1- to 7-digit decimal number specifying the minimum number of bytes of main memory that the partial image can occupy. (A hexadecimal number must be preceded by an X; e.g., XF0.)

max — is a 1- to 6-digit hexadecimal number or a 1- to 7-digit decimal number specifying the maximum number of bytes that the partial image can occupy. If the max is less than the min, Link will replace max with min and continue without displaying an error message. If a hexadecimal number is specified, it must be prefixed with an X.

Functional Details:

When Link resolves an external reference against a partial image, all of the segments within that partial image are involved. At least one segmentation register is reserved in the image being built for each segment in the partial image. It is assumed that a partial image requires all of its segments, even though the image making the references does not call entry points in each segment of the partial image.

Each entry point to the partial image is entered into the symbol table which Link creates as it processes the commands and builds the image. All entry points are entered into the symbol table whether or not the entry symbol is ever referred to by the image being built. If a partial image is never referred to, Link may delete it from the table before the map is produced.

When the task making references to the partial image is loaded, the user-specified minimum and maximum size values are compared with the actual size of the partial image. 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 partial image refers to other partial images, these references are automatically included in the image's LIB. These secondary references need not be specified again by the RESOLVE command.

Examples:

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

    ESTABLISH TASK
    RESOLVE COMX,STRUCTURE=(COMX/X0A)
    INCLUDE PROG1
    BUILD PROG1
    END

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

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

3.22 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 or file to be rewound.

Example:

```
    REWIND MAG1:
```

3.23 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 1- to 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, the title must be enclosed within single quotation marks ('). If this command and this parameter are not specified, no title is printed at the top of the maps.

Functional Details:

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

Examples:

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

3.24 VOLUME COMMAND

The VOLUME command is an environment command that specifies the volume to be used by the linkage editor when no volume is specified in an fd.

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.

Example:

```
    VO M300
```

3.25 WFILE COMMAND

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

Format:

$$\underline{\text{WF}}\text{ILE fd}\left[,\begin{Bmatrix} n \\ 1 \end{Bmatrix}\right]$$

Parameters:

fd is the file descriptor of the device or file 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.

Example:

```
WF MAG1:,2
```

CHAPTER 4
USING LINK

4.1 INTRODUCTION

This chapter provides examples of Link command sequences used to build task and operating system images. See Chapter 3 for detailed information on the Link commands.

4.2 BUILDING A TASK IMAGE

The following example builds a task image from an object module called MOD1.OBJ produced by the common assembly language (CAL) assembler. MOD1.OBJ has no external references.

Example:

```
    ES TASK
    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 image. The file extension .OBJ is the default extension for the INCLUDE command. Because INCLUDE is an active command, it is executed immediately.

The MAP command specifies that an establishment summary is to be output to PR1:. The MAP command is a passive command that is executed only when the BUILD command is entered.

The BUILD command builds the image 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 that is executed immediately.

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

4.3 BUILDING FORTRAN, COBOL, AND COMMON ASSEMBLY LANGUAGE (CAL) TASK IMAGES

This section provides examples for building COBOL, FORTRAN and CAL task images, linking subroutine libraries, outputting Link maps, using the OPTION command, and imbedding Link commands in object modules.

4.3.1 Building a COBOL Task Image

The following example builds a task image from the COBOL object module MOD2.OBJ containing external references. The task image 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.

Example:

```
ES TASK
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 image.

The LIBRARY command specifies that the COBOL run-time library (RTL) file COBOL.LIB is to be searched, and any routines that contain entry points matching external references are to be included in the task image. The LIBRARY command is a passive command that causes the specified library to be searched when the image is built.

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

The MAP command specifies that an establishment 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 and stores it in file MOD2.TSK.

The END command terminates the linkage editor.

4.3.2 Building a FORTRAN Task Image

The following example builds a task image from the FORTRAN object module MOD3.OBJ containing external references. The image is to include both single and double precision floating point capabilities and additional workspace for the user and Perkin-Elmer standard RTLs.

Both cross-reference and alphabetic Link maps are to be output to the printer.

Example:

```
INCLUDE MOD3
LIBRARY USERLIB,F7RTL
OPTION DFLOAT,FLOAT,WORK=XA00
MAP PR1:,ALPHABETIC,XREF
BUILD MOD3
END
```

The INCLUDE command specifies that the object modules in the input file MOD3.OBJ are to be included in the image.

The LIBRARY command specifies that the user library file USERLIB.OBJ and Perkin-Elmer FORTRAN RTL file F7RTL.OBJ are to be searched in the order that they are named and that any routines containing entry points matching external references are to be included in the task image.

The OPTION command specifies that the single and double precision floating point capabilities and additional workspace for the RTLs are to be included as part of the task image.

The MAP command generates an establishment summary, an alphabetic map listing the names and locations of all modules and entry points and a cross-reference map of all entry points and modules referencing them.

The BUILD command builds the task image and stores it in file MOD3.TSK.

The END command terminates the linkage editor.

4.3.3 Building a Common Assembly Language (CAL) Task Image Using Embedded Link Commands

The following example builds a task image from the CAL object module, MOD4.OBJ, containing external references and imbedded Link commands. The image will include single and double precision floating point capabilities. An establishment summary and cross-reference and alphabetic maps are to be output to the printer.

Execution of all imbedded Link commands in MOD4 is disabled by the NDCMD command; Link commands imbedded in the user library are enabled by the DCMD command. Two commands are entered on one line separated by a semicolon. Comment lines are specified by preceding each comment with an asterisk.

Example:

```
    NDCMD;*IGNORE IMBEDDED COMMANDS IN MOD4
    INCLUDE MOD4; LIBRARY USERLIB
    OPTION DFLOAT,FLOAT,WORK=XA00
    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 specified by an INCLUDE command are treated as if they were entered immediately after the INCLUDE command. Commands imbedded in object modules specified 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 enables this feature.

4.4 BUILDING OVERLAYED TASK IMAGES

This section discusses building overlayed task images using subroutines, root segments, overlay areas, root nodes, and overlay nodes. The 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 Overlayed Task Image

The following example builds a task image from the object file MOD5.OBJ which consists of a main program 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 program occupies 10kb of memory, while the largest overlay occupies 10kb of memory which is a total of 20kb for the whole task. This task would occupy 40kb of memory without using the overlay feature. The MAP command specifies that an establishment summary and address map are to be generated. All the routines are contained in file MOD5.OBJ.

Example:

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

The first INCLUDE command specifies that the object module MSP in the input file MOD5.OBJ is to be included in the image. Because MSP is not specified by an OVERLAY command, it 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. It is contained in the object file most recently specified in an INCLUDE command (MOD5.OBJ), and it will be automatically loaded into memory when MOD5 calls SUBA if it is not already in memory.

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 memory 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 include the object module called SUBC as part of overlay C.

The MAP command specifies that an establishment summary and a listing of the names and locations for each overlay are to be produced in address order.

The BUILD command builds the image called MOD5.TSK which consists of a root segment and an overlay area large enough to contain the largest overlay (A, B, or C).

The END command terminates the linkage editor.

4.4.2 Building a More Complex Overlayed Task Image

The following example builds an overlayed task image from the object file MOD6.OBJ which consists of a main program 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 overlayed when SUBA is in memory. SUBB calls SUBB1 and SUBB2, and SUBB1 and SUBB2 are to be overlayed when SUBB is in memory. This overlay process can be accomplished by using another level of overlay areas. Figure 4-1 illustrates the overlay structure for this example.

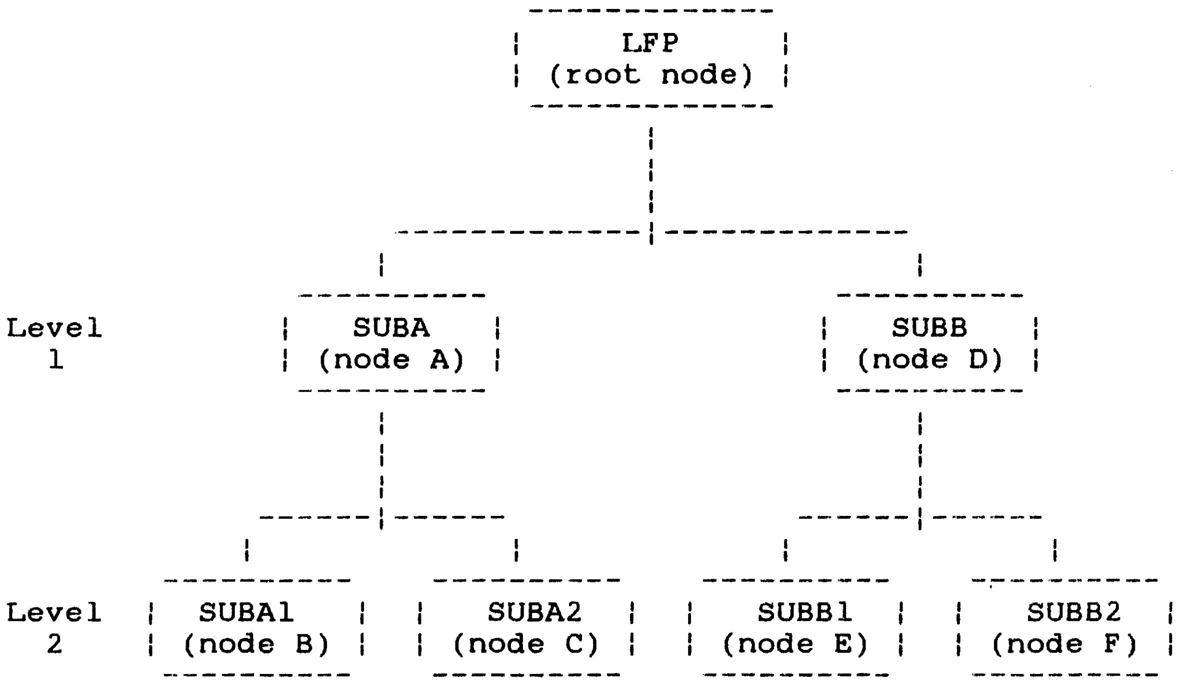


Figure 4-1 Sample Overlay Structure

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, node D and node E form a path. Only nodes in the same path can be in memory at the same time and, therefore, a routine can only call routines in nodes that are in the same path as the node containing the calling routine.

The overlay nodes can 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 requiring the largest overlay area.

In the following example all subroutines are contained in file MOD6.OBJ. Utility routines called in the task are in USERLIB.OBJ.

Example:

```
INCLUDE M300:MOD6.OBJ,LFP
  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 LFP in the input file MOD6.OBJ is to be included in the image. LFP resides in the root node.

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, indicating that these overlays are descendants of overlay D.

The LIBRARY command specifies that the user library file USERLIB.OBJ is to be searched for any routines containing entry points matching unresolved external references. These entry points 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 that overlay area is a descendant an overlay area already containing a copy of the requested routine.

If modules SUBA1 and SUBA2 refer to 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 program LFP MOD6 references routine TAG, a copy of the routine is only included in the root segment regardless of any other overlay areas referring to it. However, if two copies of a routine are to be included in two overlay areas (one being a descendant of the other), each routine must be explicitly included by the INCLUDE command.

The MAP command specifies that an establishment summary and a listing of the names and locations for each overlay in address order are to be generated.

The BUILD command builds the image 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 the locations of references to them. For example, if ALPHA is a common block referred to by routines in a particular node, ALPHA is included in that node.

If ALPHA is referred to by routines in more than one overlay node, ALPHA is included in the numerically lowest level node of the path in which each node refers to ALPHA. This is subject to the restriction that reference to ALPHA is not made in a numerically higher level node than the one in which ALPHA is placed.

If SUBA1 and SUBA2 both reference ALPHA, ALPHA 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 one that makes reference to it. For example, placing a common block in the root node prevents the data in it from being reinitialized each time the node which makes reference to it is loaded.

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

Example:

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

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

4.5 BUILDING PARTIAL IMAGES

Partial images, such as blockdata modules and RTLs, must be separately built by Link to be used or referenced by established tasks. The following 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 partial image that includes local and external entry points.

Example:

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

The first ESTABLISH command specifies that the partial image to be built is called COMMONS.IMG with read/write access privileges. The ACCESS and NAME parameters provide information that is verified against the parameters specified in a RESOLVE command for a task making reference to the partial image. For example, if a RESOLVE command in a task referring to the partial image specifies read-only access, the access is allowed because it is a subset of the maximum access privileges specified in the previous example. A request for execute access is 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 can be referred to by tasks outside the partial image.

Normally, common blocks are considered local. Note that either the STRUCTURE parameter in a subsequent RESOLVE command in the task making reference or the EXTERNAL command, not both, are required to match external references to the common with the initialized common blocks in COMMONS. The EXTERNAL command is passive.

The first BUILD command builds the partial image and stores it in file COMMONS.IMG.

The second ESTABLISH command specifies that a partial image, F7RTL.IMG, is to be built with read-execute access privileges only. The ADDRESS parameter specifies that this segment is to start at XF0000 in the address space of any task which references it. If the ADDRESS parameter is not specified, or the task making reference does not specify an address in the RESOLVE command, Link automatically locates the partial image within the address space of the task making reference.

The third INCLUDE command includes all the FORTRAN RTL routines in F7RTL.OBJ in the partial image to be built.

The LOCAL command prevents the entry points .DI, .DO, .TGD, .TASKID, .HYDEX, and .HYEXP from being referred to by tasks outside the partial image.

The second BUILD command builds the partial image and stores it in file F7RTL.IMG.

The END command terminates the linkage editor.

The operator TCOM command creates common areas within the system's task space. A task can use this common area instead of the partial image. See the OS/32 Operator Reference Manual for an explanation of the TCOM command.

4.6 BUILDING A TASK IMAGE REFERRING TO PARTIAL IMAGES

OS/32 allows multiple tasks to share a single copy of a partial task. In particular, shared common blocks allow data to be shared or communicated among tasks. Shared copies of RTLs allow more efficient use of main memory.

The following example builds a FORTRAN task image. MOD7.OBJ is a FORTRAN program that refers to two partial images, COMMONB and F7RTL. COMMONB contains two common blocks, DELTA and GAMMA. F7RTL contains the Perkin-Elmer FORTRAN RTL.

Example:

```
    INCLUDE MOD7
    RESOLVE COMMON.IMG,NAME=COMMONB,ACCESS=R,
    CONTINUE>STRUCTURE=(DELTA/X1000,GAMMA/X80)
    RESOLVE F7RTL.IMG
    MAP PR1:,ADDRESS
    BUILD MOD7
    END
```

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

The first RESOLVE command specifies that COMMON.IMG is the file containing a partial image called COMMONB, 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 4,096 bytes of the partial image COMMONB are to be allocated for the common block DELTA. The next 128 bytes after the first 4,096 bytes are to be allocated for the common block GAMMA. The parameters in the RESOLVE command are compared with the information in the file COMMON.IMG. Any information not provided by the parameters is taken from the file or defaulted. At run-time, the preinitialized partial image is loaded from the file.

The second RESOLVE command specifies that another partial image is to be loaded from the file F7RTL.IMG. All of the other parameters default to information contained in the file.

The MAP command specifies that an establishment 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 and stores it in the file MOD7.TSK. The partial images are referenced to resolve external references and to determine the placement of common blocks. The partial images are stored as separate image files and are not included as part of the task image that references them.

The END command terminates the linkage editor.

4.7 BUILDING AN OPERATING SYSTEM IMAGE

The following example builds an operating system image from the object module MTSYSTEM.OBJ produced by the library loader. MTSYSTEM.OBJ contains no external references. A map is to be generated listing the names and locations of all symbols, tasks, and entry points in alphabetical and address order.

Example:

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

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

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

The MAP command specifies that an establishment summary and a listing of the names and locations of all modules and entry points in alphabetical and address order are to be generated and sent to PR1:.

The BUILD command builds the operating system image and stores it in the file OS32R0n.000 which can be loaded into memory by the bootstrap loader or the loader storage unit (LSU).

The END command terminates the linkage editor.

CHAPTER 5
VIRTUAL TASK MANAGEMENT (VTM)

5.1 INTRODUCTION

The VTM provides a user-transparent virtual memory capability for large FORTRAN tasks. User tasks (u-tasks) consisting of up to 16Mb of code and data can execute in as little as 128kb of user task memory. VTM also supports common assembly language (CAL) and PASCAL programs with some code restrictions.

VTM uses the memory address translator (MAT) to optimize run-time performance. It contains run-time algorithms to provide performance for the widest possible scope of u-task characteristics. VTM employs a least recently used working set algorithm. The virtual activity of a VTM task is independent of the operating system and does not impact other tasks in the system. VTM tasks are nonrollable by default but can be made rollable.

5.2 SYSTEM REQUIREMENTS

The minimum requirements for use of this feature are any Perkin-Elmer processor equipped with MAT hardware, and OS/32 6.2 and higher. Perkin-Elmer processor Models 7/32, 8/32, and 3220 are not supported.

5.3 USER INTERFACE TO VIRTUAL TASK MANAGEMENT (VTM)

The following sections describe how to use VTM.

5.3.1 Declaring a Virtual Task Management (VTM) Task

The user declares a virtual task at Link via the Link OPTION command:

```
    OPTION  VTM[=n]
```

where n is the number of 64kb working pages desired for task memory management.

The minimum value of n is 2, the default is 4, and the maximum is 127. The number of working pages needed for reasonable performance varies depending upon the user's applications and needs.

NOTE

> The VTM option and the Link overlay feature are incompatible and must not be used in the same task.

5.3.2 Virtual Task Management (VTM) Secondary Storage

An additional option may also be specified via the Link OPTION command:

```
    OPTION [VFD=fd]
```

where fd is a contiguous file to be used as secondary or external storage for the virtual task.

If the VFD option is not entered, VTM allocates a temporary file at run-time.

The specified file descriptor (fd) may be the task image file itself, in which case the task image file might be destroyed at run-time. When OPTION VFD is specified, multiple copies of the same task image cannot be run concurrently. The fd must be a minimum of CTOP divided by 256 minus 255 sectors (plus 256 sectors if fd is the task image file).

5.3.3 Including the Virtual Task Management (VTM) Module

Prior to including any task modules, the user must include the VTM object module supplied with the operating system package. The VTM module is approximately 8kb in size.

5.3.4 Virtual Task Management (VTM) Task Workspace

All logical workspace required for the execution of a virtual task must be requested at Link time via the WORK option of the Link OPTION command. Additional memory cannot be obtained via the LOAD command.

5.3.5 Example of Virtual Task Management (VTM) Link Procedures

The following Link command sequence demonstrates how to build a VTM task.

Example:

```
    OPTION VTM=5
    OPTION DFLOAT,FLOAT,WORK=X3000
    INCLUDE VTM32
    INCLUDE MAIN
    INCLUDE SUB1
    INCLUDE SUB2
    LIBRARY F7RTL
    MAP PR:
    BUILD FORTTASK
    END
```

FORTTASK executes in five working pages, using a temporary file as secondary storage.

5.3.6 Virtual Task Managment (VTM) Logical Units

For a VTM task, the two highest numbered valid task logical units are reserved for VTM use. For example, if OPTION LU is not specified, logical units 13 and 14 are reserved for VTM.

5.3.7 Rolling of Virtual Task Management (VTM) Tasks

VTM tasks are nonrollable by default. A user can specify roll eligibility after loading and before starting the task by modifying the memory location which specifies roll eligibility.

Example:

```
    MOD 104,1
```

5.3.8 Absolute Code

Absolute-original code or data cannot extend beyond X'400' in a VTM task.

5.4 FORTRAN OPERATIONAL RULES

The following FORTRAN operational rules are for the VTM feature:

- The u-task workspace requested by the WORK option should not exceed 64kb in a virtual task. Input/Output (I/O) transfers are limited to 64kb.
- Nonlanguage I/O calls made through the use of SYSIO fall under the CAL coding restrictions.

5.5 COMMON ASSEMBLY LANGUAGE (CAL) RESTRICTIONS

SVC 1 I/O buffers and SVC parameter blocks should not cross logical 64kb boundaries to ensure proper execution. It is suggested that the buffers be placed in the first 64kb of the task to avoid this possibility.

5.6 PASCAL CODE RESTRICTIONS

To ensure proper execution, file variables should be declared before any other variables in the global variable declarations of the main program. The total size of the file buffers, plus 80 bytes of control data for each file, should not exceed 64kb.

5.7 PERFORMANCE MEASUREMENT

The user can analyze the relative performance of a virtual task with different numbers of working pages using the data available in the OS/32 DISPLAY ACCOUNTING command.

NOTE

> Certain tasks, by their nature, do not perform well in a virtual environment. Tasks with extensive compute bound array access in which a working set cannot be contained in the number of specified working pages might operate poorly as VTM tasks.

5.8 VIRTUAL TASK MANAGEMENT (VTM) ERROR CONDITIONS

All VTM error conditions result in the u-task being cancelled with end of task code 1 or one of the end of task codes explained in Table 5-1. A summary of VTM error messages is presented in Appendix C.

TABLE 5-1 VIRTUAL TASK MANAGEMENT (VTM) END OF TASK CODES

END OF TASK CODE	MEANING
00	SVC address error
01	Execute protect error
02	Write protect error
03	Read protect error
04	Access level error
07	Shared segment table size error
08	Private segment table size error

APPENDIX A
LINK COMMAND SUMMARY

```
          [ (  n  ) ]
BFILE fd  [,{      }]
          [ (  1  ) ]

BUILD fd

      [ ( APUCOMMENT  ) ]
DCMD  [ {             } ]
      [ ( NAPUCOMMENT ) ]

            ( TASK                                                                 )
            (                                                                      )
            ( OS                                                                   )
            (                                                                      )
            (                      ( R   )                                         )
            (                      ( E   )                                         )
ESTABLISH   {  IMAGE   [,ACCESS= { RE  } ] [,ADDRESS= { m0000 } ]                  }
            (                      ( RW  )            {   *   }                    )
            (                      ( RWE )                                         )
            (                                                                      )
            (          [,NAME=package name]                                        )

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

          [  (  n  ) ]
FFILE fd  [ ,{      }]
          [  (  1  ) ]

      [mnemonic]
HELP  [        ]
      [   *    ]
```

```
                   [ [ {module1}   [ {modulen} ]               ]
INCLUDE [fd]       [ [,{   *   }   [-{   *   } ],...,modulex   ]

LIBRARY fd1 [,...,fdn]

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

LOG fd

MAP [fd][,ALPHABETIC][,ADDRESS] [,XREF]

NDCMD

NLOG

                   {  a  }    {NACCOUNTING}    {ACPRIVILEGE }
OPTION [ABSOLUTE=  {     } ] [,{           }] [,{            }]
                   {X100 }    {ACCOUNTING }    {NACPRIVILEGE}

          {value}    {APCONTROL }    {APMAPPING }
       [,ALIGN={     }] [,{          }] [,{          }]
          { 16  }    {NAPCONTROL}    {NAPMAPPING}

        {APUONLY }    {COMMUNICATE }    {CONTROL }    {DFLOAT }
       [,{        }] [,{            }] [,{        }] [,{       }]
        {NAPUONLY}    {NCOMMUNICATE}    {NCONTROL}    {NDFLOAT}

        {DISC }    {DTABLES }
       [,{     }] [,{        }]  [,ENTRY=(main entry,debug entry)]
        {NDISC}    {NDTABLES}

        {DTASK}    {FLOAT }    {INTERCEPT }              {b}
       [,{ETASK}] [,{      }] [,{          }] [,IOBLOCKS={ }]
        {UTASK}    {NFLOAT}    {NINTERCEPT}              {1}

        {NKEYCHECK}       {lu}        {lproc}    {NAFPAUSE}
       [,{         }] [,LU={  }] [,LPU={     }] [,{        }]
        {KEYCHECK }       {15}        {  0  }    {AFPAUSE }

                      {ipri}    {mpri}         {RESIDENT }    {NROLL}
       [,PRIORITY=([  {    }] [,{    }])] [,{         }] [,{     }]
                      {128 }    {128 }         {NRESIDENT}    {ROLL }

        {SEGMENTED }                {   decimal value   }
       [,{          }] [,SYSSPACE={Xhexadecimal value}]
        {NSEGMENTED}                {      X3000        }

        {NSVCPAUSE}            {status}    {st adr}
       [,{         }] [,TSW=([{   *   }] [,{      }])]
        {SVCPAUSE }            {   0  }    {  0   }

                  {NONE   }    {UNIVERSAL }    {VFC }
       [,TEQSAVE={PARTIAL}] [,{          }] [,{    }] [,VFD=fd]
                  {ALL    }    {NUNIVERSAL}    {NVFC}

            {n}             {nominal workspace}  {maximum workspace}
       [,VTM={ }] [,WORK=({        *         },{                 })]
            {4}             {       X50       }  {     X40000      }

        {XSVC1 }
       [,{      }]
        {NXSVC1}
```

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

$$\underline{\text{PA}}\text{USE}$$

$$\underline{\text{PO}}\text{SITION }\underline{\text{CO}}\text{MMON=}\left\{\begin{matrix}\text{name}\\ (\text{name}_1,\ldots,\text{name}_n)\\ *\end{matrix}\right\}\left[,\underline{\text{TO}}\text{=}\left\{\begin{matrix}\text{nodename}\\ \text{.ROOT}\end{matrix}\right\}\right]$$

$$\underline{\text{RES}}\text{OLVE}\quad[\text{fd}]\ [,\underline{\text{NA}}\text{ME=package name}]$$

$$,\underline{\text{AC}}\text{CESS=}\left[\left\{\begin{matrix}\text{R}\\ \text{E}\\ \text{RE}\\ \text{RW}\\ \text{RWE}\end{matrix}\right\}\right]\ [,\underline{\text{AD}}\text{DRESS=m0000}]$$

$$\left[,\underline{\text{ST}}\text{RUCTURE=}\left(\text{name}_1\ [/\text{size}_1]\ [,\ldots,\text{name}_n\ [/\text{size}_n]]\right)\right]$$

$$[,\underline{\text{SI}}\text{ZE= }[\text{min },\text{max}]]$$

$$\underline{\text{REW}}\text{IND fd}$$

$$\underline{\text{TI}}\text{TLE title}$$

$$\underline{\text{VO}}\text{LUME }[\text{voln}]$$

$$\underline{\text{WF}}\text{ILE fd}\left[,\left\{\begin{matrix}\text{n}\\ 1\end{matrix}\right\}\right]$$

APPENDIX B
LINK MESSAGE SUMMARY

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 to a node that is not in the same path as is the node referring to it.

ATTEMPT TO POSITION x IN LOWER LEVEL NODE

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

```
ATTEMPT TO REFERENCE ADDRESS number
ADDRESS OUTSIDE OF ADDRESS SPACE FOR IMAGE
-FILE:  vol:filename.ext/a  -MODULE:module
-RECORD:number - BYTE:number
```

The task image being built refers to an address outside the address space of any of the known segments or partial images of the task. This message identifies the file, module, record number, and byte number of the object code that caused the error.

BUILD NOT SUPPORTED ON THIS DEVICE

A file other than an indexed, nonbuffered indexed, contiguous or extended contiguous file, or the null device 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 is not permitted in a common assembly language (CAL) object module.

COMMON x ENCOUNTERED IN MORE THAN ONE PARTIAL IMAGE

The same common block was specified in more than one of the partial images referred to by the task.

COMMON BLOCK x, UNREFERENCED

The common block named was never referred to.

COMMON BLOCK x SPECIFIED IN POSITION COMMAND IS PART OF PARTIAL IMAGE

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

CONTINUATION NOT PERMITTED

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

ENTRY POINT x SPECIFIED IN ENTRY OPTION NOT FOUND

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

ENTRY POINT x SPECIFIED IN LOCAL COMMAND NOT DEFINED

The entry point named was never defined.

ESTABLISHMENT ABORTED

A serious error occurred that prevented the image from being built. Link is cleared as if an image was built with all options reset to initial load values.

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 was encountered.

fd IS NOT A PARTIAL IMAGE

The file descriptor (fd) specified by the RESOLVE command is not a partial image.

fd NOT FOUND

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

INSUFFICIENT WORK SPACE

Link was not loaded with enough workspace. It will return to command mode, clear itself as if an image had been built with all options reset to initial load values.

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 specified fd.

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 PARAMETER LENGTH

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

```
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, leaves the remainder of the chain unresolved, and prints the error message above.

```
LOCATION COUNTER number WAS DEFINED PREVIOUSLY
THIS ERROR OCCURRED IN
-FILE:  vol:filename.ext/a  -MODULE:module
-RECORD:number  -BYTE:number
```

The specified location counter (LOC) number in the object code was already defined by Link and cannot be redefined within this object module. To correct this error, recompile the module identified by this message.

```
LOCATION COUNTER number WAS NOT DEFINED PREVIOUSLY
THIS ERROR OCCURRED IN
-FILE:  vol:filename.ext/a  -MODULE:module
-RECORD:number  -BYTE:number
```

The object code did not define the specified LOC for Link. To correct this error, recompile the module identified by this message.

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 loader item in an object module.

MODULE xxxxxx ATTEMPTS TO INITIALIZE xxxxxx THAT IS IN A PARTIAL IMAGE

While a task is being linked, the task cannot initialize any common blocks within the partial images that are resolved with the task. Consequently, if the task attempts to perform an initialization; e.g., through a BLOCKDATA statement, Link will build the image but no initialization of that common block is performed. After the task image is built, the task common will contain the data that was present when the partial image was built. The above message indicates which object module tried to perform the initialization of the specified block within the partial image.

MODULE xxxxxxx NOT FOUND

A module specified in an INCLUDE command was not found.

MORE THAN 192 SEGMENTATION REGISTERS REQUIRED

More segmentation registers are required than the maximum 192.

n AMBIGUOUSLY DEFINED SYMBOLS

Entry points were defined in parallel paths and were referred to by a node common to both paths. This message appears in the establishment summary of the Link maps and is followed by a list of the ambiguously defined entry points.

n COMMAND(S) ENCOUNTERED IN OBJECT CODE

The specified number (n) of Link commands imbedded in the CAL object modules included in the image were encountered.

n MULTIPLY DEFINED SYMBOLS

The specified number (n) of entry points were encountered which were defined more than once in the image being built.

n UNDEFINED EXTERNAL SYMBOLS

This message is output if any standard external symbols remain unresolved after the image is built.

n UNDEFINED WEAK EXTERNAL SYMBOL(S)

This message is output if any weak external symbols remain unresolved after the image is built.

name SPECIFIED IN POSITION COMMAND NOT FOUND

The named common block that was specified by a POSITION command could not be found.

NODE is NOT SUITABLE FOR OVERLAYS

This message indicates that the Link command sequence is attempting to overlay the task in a partial image or pure segment.

NUMERIC VALUE OUT OF RANGE

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

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

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

PROGRAM TRANSFER ADDRESS IN PROGRAM module IN AN OVERLAY

A program transfer address (PTA)(starting address) was specified for the task in a module that is in an overlay node. Link ignores the specified PTA and calculates the task's starting address by another method.

OVERLAY DEFINED OUT OF ORDER

An OVERLAY command specified a level inconsistent with the rules for defining overlays.

RECORD LENGTH FOR MAP DEVICE/FILE < 64 BYTES

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

SEGMENT AT x OVERLAPS PREVIOUSLY DEFINED SEGMENT

The end address of an impure, pure, or shared logical segment was greater than the beginning address of another segment. See the establishment summary 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.

SIZE OF SEGMENT TRUNCATED TO PHYSICAL SIZE

The maximum length of the partial image specified by the SIZE parameter in the RESOLVE command is larger than any existing segment for that image. This message indicates that Link is using the size of the existing segment for the maximum partial image size rather than the maximum specified by SIZE.

TOO MANY OPERANDS

More operands than allowed were encountered.

VTM TASK WORKSPACE IS GREATER THAN 64K BYTES

When a FORTRAN task is linked as a virtual task, the user task workspace requested by the WORK option should not exceed 64kb. This message indicates that the WORK option for the FORTRAN task being linked exceeds 64kb.

VIRTUAL SYMBOL TABLE SPACE LIMIT EXCEEDED

More than 256kb of symbol table space required.

WARNING: ABSOLUTE SPACE LESS THAN 100

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

WARNING: ADDRESS OF PARTIAL IMAGE SEGMENT FOR fd DOES NOT MATCH ADDRESS SPECIFIED ON RESOLVE COMMAND

This warning is output if the RESOLVE command specifies an fd and an address for an address-dependent partial image, and that address does not match the address in the loader information block (LIB) for that partial image. Link uses the address specified in the partial image's LIB.

WARNING: COMMON xxxxxx APPEARS MORE THAN ONCE IN STRUCTURE COMMAND

In the STRUCTURE parameter of the RESOLVE command, the user attempted to use the same name to define two separate common blocks. Common block names within a partial image must be unique.

WARNING: ITEM NOT PERMITTED IN AN ADDRESS INDEPENDENT SEGMENT
-FILE: vol:filename.ext/a -MODULE:module
-RECORD:number -BYTE:number

The loader item encountered cannot be properly processed while building an address independent partial image segment. Loader items involving relocatable data or items which set the LOC to an absolute value cause this message to be displayed.

WARNING: ITEM NOT PERMITTED IN E-TASK
-FILE: vol:filename.ext/a -MODULE:module
-RECORD:number -BYTE:number

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

WARNING: LOGICAL UNIT 254 IS RESERVED FOR DEBUG PROGRAM

This message is displayed of lu=255 is entered with the DTABLES option. If the program is to be debugged using DEBUG/32, logical unit (lu) 254 cannot be assigned by the program.

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: NAME OF PARTIAL IMAGE FOR fd DOES NOT MATCH NAME SPECIFIED IN RESOLVE COMMAND

The name given to a partial image when it was linked does not match the name specified in the NAME parameter of the RESOLVE command. The package name specified in the RESOLVE command overrides the name found in the LIB of the partial image file.

WARNING: OPTION "NSEGMENTED" HAS BEEN SELECTED

An invalid segmentation option was selected. Link builds a nonsegmented task.

WARNING: OPTION "VTM" HAS BEEN DISABLED. INCOMPATIBLE OPTIONS SPECIFIED

User selected task options that are incompatible with VTM.

WARNING: OPTION "VTM" HAS BEEN DISABLED. TASK ABSOLUTE AREA GREATER THAN X400.

VTM will not run if the task absolute area is greater than X'400'.

WARNING: OPTION "VTM" HAS BEEN DISABLED. VIRTUAL CTOP EXCEEDS ACTUAL CTOP OF TASK.

Number of allocated VTM pages exceeds the actual size of the task. Increase task workspace or decrease number of VTM pages.

WARNING: OPTION "VTM" HAS BEEN DISABLED. VTM OBJECT MODULE NOT FOUND

User omitted INCLUDE command for VTM32.OBJ.

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

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

WARNING: PREASSIGNMENT FOR LU NOT USED

After Link was loaded, the user assigned an lu that could not be used as an input/output (I/O) file for Link.

WARNING: TASK REQUIRES MORE THAN 1M ADDRESS SPACE

The task being built requires more than 1Mb of memory address space.

WARNING: TASK REQUIRES MORE THAN 12M ADDRESS SPACE

The task being built requires more than 12Mb of memory address space.

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 function, 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
	NAME	4	Filename does not exist on specified volume
	DISC SPACE	5	Insufficient disk space available to allocate or assign a file
	PROTECTION KEY	6	File being assigned had non-zero protection keys
	ACCESS PRIVILEGE	7	Specified access privileges could not be granted
	SYSTEM SPACE	8	Insufficient system space available
	ASSIGNMENT	9	lu is already assigned or device is offline
	DEVICE TYPE	A	Specified volume is not a direct access device
	FILE DESCRIPTOR	B	The fd format is incorrect
	TRAP GENERATING DEVICE	C	Specified trap generating device does not exist in the system, is not a connectable device, or is busy and cannot be connected
	GROUP/ SYSTEM FILE	D	Allocation or deletion was attempted on a system or group file

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 TYPES AND STATUS

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

APPENDIX C
VIRTUAL TASK MANAGEMENT (VTM) MESSAGE SUMMARY

INSUFFICIENT VTM WORKING PAGES

For this task, at least one additional working page is required for VTM execution.

MEM.FAULT AT xxxxxx INSTR AT xxxxxx CODE=xx (task paused)

Task memory access fault. xx is the SVC 7 error status.

TASK FD ASGN-ERR - CODE=xx

Error in assigning task file. xx is the SVC 7 error status.

VIRT FD ALLO-ERR - CODE=xx

Error in allocating temporary file. xx is the SVC 7 error status.

VIRT FD ASGN-ERR - CODE=xx

Error in assigning VFD file. xxx is the SVC 7 error status.

VIRT FD NOT CONTIG

Specified file is not contiguous.

VIRT FD TOO SMALL

Specified file is too small.

VTM RD-ERR STAT=xxxx (task paused)

Unrecoverable read error on a virtual I/O transfer. xxxx is the SVC 1 status halfword; a device independent status of 00 indicates a length of transfer error.

VTM WT-ERR STAT=xxxx (task paused)

Unrecoverable write error on a virtual I/O transfer. xxxx is the SVC 1 status halfword; a device independent status of 00 indicates a length of transfer error.

APPENDIX D
OBJECT MODULE FORMAT

Object modules accepted by Link are stored in indexed files with a record length of 126 bytes. Each record contains a sequence number, a checksum, and at least one loader item.

The sequence number is contained in the first two bytes of the record. The first record of the module has a sequence number of -1 (hexadecimal value FFFF). For each record following, one is subtracted from the sequence number. Record two has the sequence number -2, or FFFE. Record three has -3, or FFFD. This continues until the last record in the object module is reached or until a loader item is encountered which resets the sequence number to -1.

The second two bytes of the record contain the checksum for the record. It is calculated by performing an EXCLUSIVE-OR operation on each halfword of data in the record (excluding the sequence number).

The remainder of the record contains loader items. A loader item is a command byte, followed by zero or more bytes of data. The command byte informs Link how to interpret the data which follows or requests Link to perform some specific action.

For example, loader item 11 is followed by six bytes of data. The first three are to be loaded directly into the image at the current location in the image. The last three are to be used as an address offset from the beginning of the impure area for this object module. The absolute address of the impure area is to be added to this offset. The least-significant three bytes of the resulting sum are to be stored in the image immediately following the first three bytes. The current location is to be incremented by six bytes.

Loader items must end in the record in which they begin. They may not begin in one record and finish in the following record.

This appendix lists the object code loader items accepted by LINK R01. Each loader item is followed by a description of the data to be associated with it.

LOADER ITEM	DATA FORMAT	DESCRIPTION
0	(none)	End of record
1	(none)	End of object module
2	(none)	Reset sequence number
3	8 bytes name 3 bytes displacement any of these loader items: 7, 8, 9, A, 10, 11, 15, 16, 1B, 1C, 1D, 1F-5B, 60, 61, 62, 63, 64	Block data item
4	3 bytes address value	Absolute program address
5	3 bytes address value	Pure relocatable address
6	3 bytes address value	Impure relocatable address
7	2 bytes address data	Pure relocatable address
8	2 bytes address data	Impure relocatable address
9	4 bytes address data	Pure relocatable address
A	4 bytes address data	Impure relocatable address
B	8 bytes common name 3 bytes displacement	Common reference
C	8 bytes external name address loader item (4, 5, 6, or 5F)	External reference EXTRN
D	8 bytes entry name address loader item (4, 5, 6, or 5F)	Entry point definition
E	8 bytes common name 3 bytes length	Common block definition
F	8 bytes program name	Program name
10	3 bytes absolute data 3 bytes address data	Instruction with pure relocatable address
11	3 bytes absolute data 3 bytes address data	Instruction with impure relocatable address

LOADER ITEM	DATA FORMAT	DESCRIPTION
12	address loader item (4, 5, 6, or 5F)	Load program start address
13	address loader item (4, 5, 6, or 5F)	Start of reference chain
14	address loader item (4, 5, 6, or 5F)	Chain definition address
15	2 bytes absolute data 2 bytes address data	Instruction with pure relocatable address
16	2 bytes absolute data 2 bytes address data	Instruction with impure relocatable address
17	8 bytes external name address loader item (4, 5, 6, or 5F)	Short (halfword) external reference
18	3 bytes impure length 3 bytes pure length	Length of pure and impure segments
19	(none)	Perform fullword chain
1A	(none)	Perform halfword chain
1B	(none)	No operation
1C	2 bytes address data	Pure translation table address
1D	2 bytes address data	Impure translation table address
1E		Not used
1F	1 byte absolute data	Absolute data
20	2 bytes absolute data	Absolute data
21	4 bytes absolute data	Absolute data
22	6 bytes absolute data	Absolute data
23	8 bytes absolute data	Absolute data
24	10 bytes absolute data	Absolute data
25	12 bytes absolute data	Absolute data
26	14 bytes absolute data	Absolute data

LOADER ITEM	DATA FORMAT	DESCRIPTION
27	16 bytes absolute data	Absolute data
28	18 bytes absolute data	Absolute data
29	20 bytes absolute data	Absolute data
2A	22 bytes absolute data	Absolute data
2B	24 bytes absolute data	Absolute data
2C	26 bytes absolute data	Absolute data
2D	28 bytes absolute data	Absolute data
2E	30 bytes absolute data	Absolute data
2F	32 bytes absolute data	Absolute data
30	34 bytes absolute data	Absolute data
31	36 bytes absolute data	Absolute data
32	38 bytes absolute data	Absolute data
33	40 bytes absolute data	Absolute data
34	42 bytes absolute data	Absolute data
35	44 bytes absolute data	Absolute data
36	46 bytes absolute data	Absolute data
37	48 bytes absolute data	Absolute data
38	50 bytes absolute data	Absolute data
39	52 bytes absolute data	Absolute data
3A	54 bytes absolute data	Absolute data
3B	56 bytes absolute data	Absolute data
3C	58 bytes absolute data	Absolute data
3D	60 bytes absolute data	Absolute data
3E	62 bytes absolute data	Absolute data
3F	64 bytes absolute data	Absolute data
40	66 bytes absolute data	Absolute data

LOADER ITEM	DATA FORMAT	DESCRIPTION
41	68 bytes absolute data	Absolute data
42	70 bytes absolute data	Absolute data
43	72 bytes absolute data	Absolute data
44	74 bytes absolute data	Absolute data
45	76 bytes absolute data	Absolute data
46	78 bytes absolute data	Absolute data
47	80 bytes absolute data	Absolute data
48	82 bytes absolute data	Absolute data
49	84 bytes absolute data	Absolute data
4A	86 bytes absolute data	Absolute data
4B	88 bytes absolute data	Absolute data
4C	90 bytes absolute data	Absolute data
4D	92 bytes absolute data	Absolute data
4E	94 bytes absolute data	Absolute data
4F	96 bytes absolute data	Absolute data
50	98 bytes absolute data	Absolute data
51	100 bytes absolute data	Absolute data
52	102 bytes absolute data	Absolute data
53	104 bytes absolute data	Absolute data
54	106 bytes absolute data	Absolute data
55	108 bytes absolute data	Absolute data
56	110 bytes absolute data	Absolute data
57	112 bytes absolute data	Absolute data
58	114 bytes absolute data	Absolute data
59	116 bytes absolute data	Absolute data
5A	118 bytes absolute data	Absolute data

LOADER ITEM	DATA FORMAT	DESCRIPTION
5B	120 bytes absolute data	Absolute data
5C	1 byte location counter number 8 bytes section name 8 bytes data pool name	Define PURE location counter
5D	reserved for future use	Reserved
5E	reserved for future use	Reserved
5F	1 byte location counter number 3 bytes address data	Load program address
60	1 byte location counter number 2 bytes address data	Defined counter relocatable address
61	1 byte location counter number 4 bytes address data	Defined counter relocatable address
62	1 byte location counter number 2 bytes absolute data 2 bytes address data	Instruction with address based on a defined location counter
63	1 byte location counter number 3 bytes absolute data 3 bytes address data	Instruction with an address based on a defined location counter
64	reserved for future use	Reserved
65	8 bytes external name 1 byte reference type 00 - Standard 01 - Weak 10 - INCLD 4 bytes address offset address loader item (4, 5, 6, or 5F)	Extended external reference

LOADER ITEM	DATA FORMAT	DESCRIPTION
66	8 bytes entry name 1 byte entry type 00 - Standard 01 - Data 10 - Weak address loader item (4, 5, 6, or 5F)	Extended entry point definition
67	1 byte character count 1-80 bytes of command	Imbedded LINK commands
68-FF	reserved for future use	Reserved

INDEX

PUBLICATION COMMENT FORM

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

1. Publication number____

2. Title of publication____

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

____

____

____

4. Was the publication easy to understand? If not, why?

____

5. Were illustrations adequate? ________________________________

____

____

6. What additions or deletions would you suggest? ____________________

____

____

7. Other comments: ____

____

____

From ______________________________ Date ______________________

Position/Title ________________________

Company ___________________________

Address ___________________________

____

____

APLE STAPLE

ILD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

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

POSTAGE WILL BE PAID BY ADDRESSEE

PERKIN-ELMER

Data Systems Group
106 Apple Street
Tinton Falls, NJ 07724

TTN:
ECHNICAL SYSTEMS PUBLICATIONS DEPT.

LD FOLD

APLE STAPLE

PERKIN-ELMER
Technical Systems Division

DOCUMENTATION CHANGE NOTICE

The purpose of this documentation change notice (DCN) is to provide a quick and efficient way of making technical changes to manuals before they are formally updated or revised.

The manual affected by these changes is:

48-005 F00 R02 OS/32 Link Reference Manual

For conversion purposes, a Compatible Link Utility (R02) is included with the OS/32 Software Package. This utility is designed to allow users who have extensive Link command files built using the Link R01 command syntax to continue to use those Link command sequences and also be able to use all of the new enhancements included in the R02 revision of Link.

The users who elect to use the Compatible Link Utility should note that there are five commands with formats that differ from the formats documented in the R02 release of Link. The formats of these commands are the same as those documented in the R01 Link Manual.

The differences between the R01 and R02 versions of these commands are as follows. Keep in mind that the R01 versions of these commands are those supported by the Compatible Link Utility.

- BUILD Command

  The BUILD command fd has a default extension of .IMG in the Link R01 version, and an R02 default extension of .SEG, documented on Pages 3-5 and A-1 of the R02 version.

- ESTABLISH Command

The ESTABLISH command in the R01 manual has a SHARED option. The R01 version of the ESTABLISH command is:

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

On Pages 3-12 and A-1 of the R02 manual, the ESTABLISH command has an IMAGE option instead of the SHARED option in the R01 version. The R02 version is:

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

- OPTION Command

The OPTION command has different values for the ENTRY, WORK, and SYSSPACE options. The R01 version of the OPTION command is:

```
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}]

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

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

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

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

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

The R02 version of the OPTION command, that appears on Pages 3-33 and A-2, is:

```
OPTION [ABSOLUTE={ a    }] [,{NACCOUNTING}] [,{ACPRIVILEGE }]
       [         {X100  }] [ {ACCOUNTING }] [ {NACPRIVILEGE}]

       [,ALIGN={value}] [,{APCONTROL }] [,{APMAPPING }]
       [       { 16  }] [ {NAPCONTROL}] [ {NAPMAPPING}]

       [,{APUONLY }] [,{COMMUNICATE }] [,{CONTROL }] [,{DFLOAT }]
       [ {NAPUONLY}] [ {NCOMMUNICATE}] [ {NCONTROL}] [ {NDFLOAT}]

       [,{DISC }] [,{DTABLES }] [,ENTRY=(main entry,debug entry)]
       [ {NDISC}] [ {NDTABLES}]

       [,{DTASK}] [,{FLOAT }] [,{INTERCEPT }] [,IOBLOCKS={b}]
       [ {ETASK}] [ {NFLOAT}] [ {NINTERCEPT}] [          {1}]
       [ {UTASK}]

       [,{NKEYCHECK}] [,LU={lu}] [,LPU={lproc}] [,{NAFPAUSE}]
       [ {KEYCHECK }] [    {15}] [     {  0  }] [ {AFPAUSE }]

       [,PRIORITY=([{ipri}] [,{mpri}])] [,{RESIDENT }] [,{NROLL}]
       [          ([{128 }] [ {128 }])] [ {NRESIDENT}] [ {ROLL }]

       [,{SEGMENTED }] [,SYSSPACE={   decimal value   }]
       [ {NSEGMENTED}] [          {Xhexadecimal value }]
                       [          {      X3000        }]

       [,{NSVCPAUSE}] [,TSW=([{status}] [,{st adr}])]
       [ {SVCPAUSE }] [     ([{  *   }] [ {      }])]
                      [     ([{  0   }] [ {  0   }])]

       [,TEQSAVE={NONE   }] [,{UNIVERSAL }] [,{VFC }] [,VFD=fd]
       [         {PARTIAL}] [ {NUNIVERSAL}] [ {NVFC}]
       [         {ALL    }]

       [,VTM={n}] [,WORK=({nominal workspace},{maximum workspace})]
       [     {4}] [      ({        *        } {                 })]
                  [      ({       X50       } {     X40000      })]

       [,{XSVC1 }]
       [ {NXSVC1}]
```

- SHARED Command

The SHARED command in the R01 manual is replaced by the RESOLVE command on Pages 3-52 and A-3 in the R02 manual. The SHARED command syntax is:

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

                   ┌ ⎧ R   ⎫ ┐
                   │ ⎪ E   ⎪ │                ⎧m0000⎫
          ,ACCESS= │ ⎨ RE  ⎬ │ [,ADDRESS=     ⎨     ⎬ ]
                   │ ⎪ RW  ⎪ │                ⎩  *  ⎭
                   └ ⎩ RWE ⎭ ┘

          [,STRUCTURE=(name1  [/size]   [,...,namen [/sizen]])]
          [,SIZE=([min  [,max]])]
```

The R02 RESOLVE command syntax is:

```
RESOLVE     [fd] [,NAME=package name]

                    ┌ ⎧ R   ⎫ ┐
                    │ ⎪ E   ⎪ │
          ,ACCESS=  │ ⎨ RE  ⎬ │  [,ADDRESS=m0000]
                    │ ⎪ RW  ⎪ │
                    └ ⎩ RWE ⎭ ┘

          [,STRUCTURE= (name1 [/size1] [,...,namen [/sizen]])]
          [,SIZE= [min ,max]]
```

The rest of this DCN refers to errors that must be corrected in the R02 version of the Link Manual. This portion of the DCN is not related to the Compatible Link Utility.

- Page iv

  Please delete reference to Table 5-1, and add the following reference after B-2:

  C-1 VIRTUAL TASK MANAGEMENT (VTM) MEMORY FAULT CODES

  with a page reference of C-1.

- Page 5-3

  In the last sentence, please change:

  Absolute-original code... to:

  Absolute-origined code...

- Page 5-4

  In the last paragraph, please change:

  or one of the end of task codes explained in Table 5-1.

  to:

  or one of the memory fault codes explained in Table C-1.

- Page 5-5

  Please delete Table 5-1 from Page 5-5. This table will appear on Page C-1.

- Page C-1

  After the second message, please insert the table from Page 5-5, with the following changes:

  TABLE C-1 VIRTUAL TASK MANAGEMENT (VTM) MEMORY FAULT CODES

Please change the heading for the first column of this table from:

END OF TASK CODES

to:

MEMORY FAULT CODES

- Page C-1

  After the second message (MEM FAULT AT ...), please delete the sentence that reads:

  xx is the SVC 7 error status.

  and replace it with the following sentence:

  xx specifies the code that describes the type of memory error fault that occurred. These codes are defined in Table C-1.

- Page C-1

  In the explanation for the fifth message (VIRT FD ASGN-ERR...), please change:

  xxx is the SVC...

  to:

  xx is the SVC...

- Page IND-3

  Under the alphabetical heading V, in the 6th line, please change:

  end of task codes

  to:

  memory fault codes

  with a page reference of C-1.