OS/32 LINK
Reference Manual

OS/32 Version R08-03 and higher

48-005 F00 R05

The information contained in this document is subject to change without notice. Concurrent Computer Corporation has taken efforts to remove errors from this document, however, Concurrent Computer Corporation's only liability regarding errors that may still exist is to correct said errors upon their being made known to Concurrent Computer Corporation.

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 all copyright notices, trademarks, or other legends or credits of Concurrent Computer Corporation and/or its suppliers. Title to and ownership of the described software and any copies thereof shall remain in Concurrent Computer Corporation and/or its suppliers.

The licensed program described herein may contain certain encryptions or other devices which may prevent or detect unauthorized use of the Licensed Software. Temporary use permitted by the terms of the License Agreement may require assistance from Concurrent Computer Corporation.

Concurrent Computer Corporation assumes no responsibility for the use or reliability of this software if used on equipment that is not supplied by Concurrent Computer Corporation.

Concurrent Computer Corporation, 106 Apple Street

Tinton Falls, New Jersey 07724

Printed in the United States of America

TABLE OF CONTENTS

CHAPTERS (Continued)

CHAPTERS (Continued)

CHAPTERS (Continued)

APPENDIXES

FIGURES

TABLES

SYNTAX CONVENTIONS

GENERAL SYNTAX RULES

These rules clarify the syntax of the commands in this document. Refer to these conventions when interpreting the command syntax.

Multiple commands may appear on one line, but each one must be separated by a semicolon (;). When multiple commands are entered on the same line, they are executed sequentially. If an error occurs, any subsequent commands on the line are ignored.

If the first character of any command input is an asterisk (*), the remainder of that line is considered to be a comment and is not executed. It is copied to the system log device if logging is active.

In a batch environment, continue parameters by entering a comma as the last character and continuing the parameters on the following line.

Statement Syntax Conventions

The following conventions are used in all statement, command, and instruction formats. They point out differences between information that must be entered exactly as shown and that which denotes information provided by the user. However, when entering this information, upper- or lower-case can be used.

Underlining points out the mnemonic of the entry and means that at least the underlined portion must be entered. If an entry is not underlined at all, the entire entry must be entered.

    PAUSE

Commands and parameters are represented in upper-case and must be entered as shown.

    DELETE actno

Variables are represented in lower-case and denote information provided by the user:

    ACCOUNT n

Punctuation must be entered exactly as shown.

Commas separate parameters and substitute for missing positional parameters:

LIST [actno$_1$ [,actno$_2$]]

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

PRIVILEGE [actno$_1$ [-actno$_2$] [,{priv$_1$ [,priv$_2$] / *}]]

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

EOU [NOSAVE][,UNPROTECT]

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

IOTAB (classno,classid[,iocount])$_1$,..., (classno,classid[,iocount])$_n$

Brackets represent optional parameters:

ENCRYPT [fd]

Braces represent required parameters of which one must be chosen:

LOG {ON / OFF}

Shading represents default options:

ERRMODE EQU {0 / 1}

An equals sign associates a parameter with its keyword:

Upper- and Lower-Case Characters

All commands and parameters can be entered in either upper- or lower-case. Parameters that are retained internally (such as task identifiers) are translated to uppercase. Subsequent displays show the uppercase version.

Decimal and Hexadecimal Numbers

The OS/32 commands use decimal rather than hexadecimal numbers for most numeric operands. One exception is addresses, which are expressed in hexadecimal. Numeric operands are always integers except for the SET SYS and TCOM commands, and the segment size increment field of the LOAD command where the decimal point is permissible. Leading zeros can be omitted in numeric operands, whether decimal or hexadecimal.

Task Identifiers

Task identifiers must consist of 1- to 8-alphanumeric characters; the first character must be alphabetic. Valid task identifiers are:

```
TASK3

MAX

X

T997XY25
```

Examples of invalid task identifiers are:

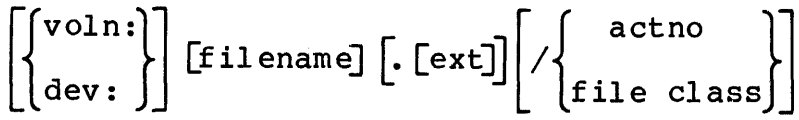
34TASK	First character is not alphabetic
T43.2	Contains a nonalphanumeric character
TASK12345	Contains more than eight characters

The background task has the special identifier .BG[nnnnn].

File Descriptors

Many of the command formats in this manual require the user to enter a file descriptor (fd). File descriptors are entered in the following format:

Format:



Parameters:

voln:	is the name of the disk volume on which the file resides. It may be from 1 to 4 characters long. The first character must be alphabetic and the remaining alphanumeric. This parameter need not be specified. If this parameter is not specified, the default user volume is used. When voln is not specified, the colon separating voln and filename must not be entered.
dev:	is a 1- to 4-character device name. The first character must be alphabetic and the remaining alphanumeric. A colon must follow the device name, and neither the filename or the extension is entered.
filename	is the name of a file and is from 1 to 8 characters long. The first character must be alphabetic and the remaining alphanumeric. If a filename is specified when a device mnemonic is specified as dev:, the filename is ignored.
.ext	is a 1- to 3-character alphanumeric string preceded by a period (.) specifying the extension to a filename. If the period and extension are omitted, a default extension is appended to the filename, if appropriate for that particular command; otherwise, it remains blank. If the period is specified and the extension is omitted, the default is a blank extension.

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

file class — is the class name of the file and consists of one character. The class names are:

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

The file class can be specified by a terminal user or the system operator. If the system operator specifies /P, /G, or /S, the operation is performed to account 0 only.

Examples:

In the following example, PACK: is the volume name, CAL is the filename, .TSK is the extension name and 0 is the account number.

```
PACK:CAL.TSK/0
```

In the following example, CONV is the filename, and .CAL is the extension name with a default account number on the default volume.

```
CONV.CAL
```

In the following example, all filenames beginning with CGG as the first three characters and ending with the extension GG are requested.

```
CGG-.*GG Possible filenames are:

CGG1.AGG          CGG12.AGG          CGG123.AGG
CGG1.BGG          CGG12.BGG          CGG123.BGG
CGG1.CGG          CGG12.CGG          CGG123.CGG
   .                  .                  .
   .                  .                  .
   .                  .                  .
CGG12345.XGG      CGGABCDE.ZGG       CGG.YGG
```

In the following example, CAL is the filename with a default extension, default account number, and default volume.

```
CAL
```

In the following example, M300: is the volume name, and MAR is the filename with a default extension and default account number.

```
M300:MAR
```

In the following example, CARD: is the device mnemonic.

```
CARD:
```

PREFACE

This manual describes the 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 an operating system. This manual is intended for all users who are developing programs for execution on 32-bit computers using the OS/32 operating system. The user should be familiar with Multi-Terminal Monitor (MTM) if Link is to be used in an MTM environment (see the Multi-Terminal Monitor (MTM) Reference Manual).

Chapter 1 provides an 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). Chapter 6 discusses relocation within executive tasks (e-tasks). Chapter 7 provides an explanation of the OBJECT/32 Utility. Appendix A is the Link and OBJECT/32 command summary. Appendix B is the Link and OBJECT/32 message summary. Appendix C is the VTM message summary. Appendix D explains the format of an object module that is compatible with Link.

The following changes were made in revision F00 R05:

- Two optional parameters, ABORT and ERROR, have been added to the BUILD command.
- The INCLUDE command has a new parameter, -BLKDATA which controls the inclusion of block data.
- A new parameter was added to the MAP command. The UNREFERENCED parameter specifies that all symbols, including those that are not used, will appear in the map(s) requested.
- The OBJECT/32 Utility is documented in the R08-03 software release. Chapter 7 has been added to describe the utility and how it improves the performance of Link. Also included is loading and starting information. OBJECT/32 commands are described with their purpose, syntax, and functional details.
- New messages associated with the OBJECT/32 Utility have been added to Appendix B.

This manual is intended for use with OS/32 R08-03 software version and higher.

CHAPTER 1
OS/32 LINK

1.1 INTRODUCTION

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 OS/32 Library Loader or Sysgen/32. The resulting image can be loaded into memory using OS/32 Bootstrap Loader or loader storage unit (LSU).

This manual includes the DEBUG/32 tables (DTABLES) task option. It also allows Link to separate symbolic debug data from the object code and build this data into the tables required by DEBUG/32. Support for the virtual task manager (VTM) has been included also. 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 128kB of memory.

OS/32 Link can be used with both the uniprocessor system and the multiprocessor system (3200MPS Family of Processors). The multiprocessor system consists of one central processing unit (CPU) and any combination up to a maximum of nine auxiliary processing units (APUs) and input/output processors (IOPs). In a multiprocessor system, the operating system defines a set of logical processing units (LPUs) that are used to direct tasks to execution queues. Link assigns the initial LPU for each task. Link also sets APU control or queue 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 3200MPS Family of Processors Overview Manual for more information on using the 3260MPS System.

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.

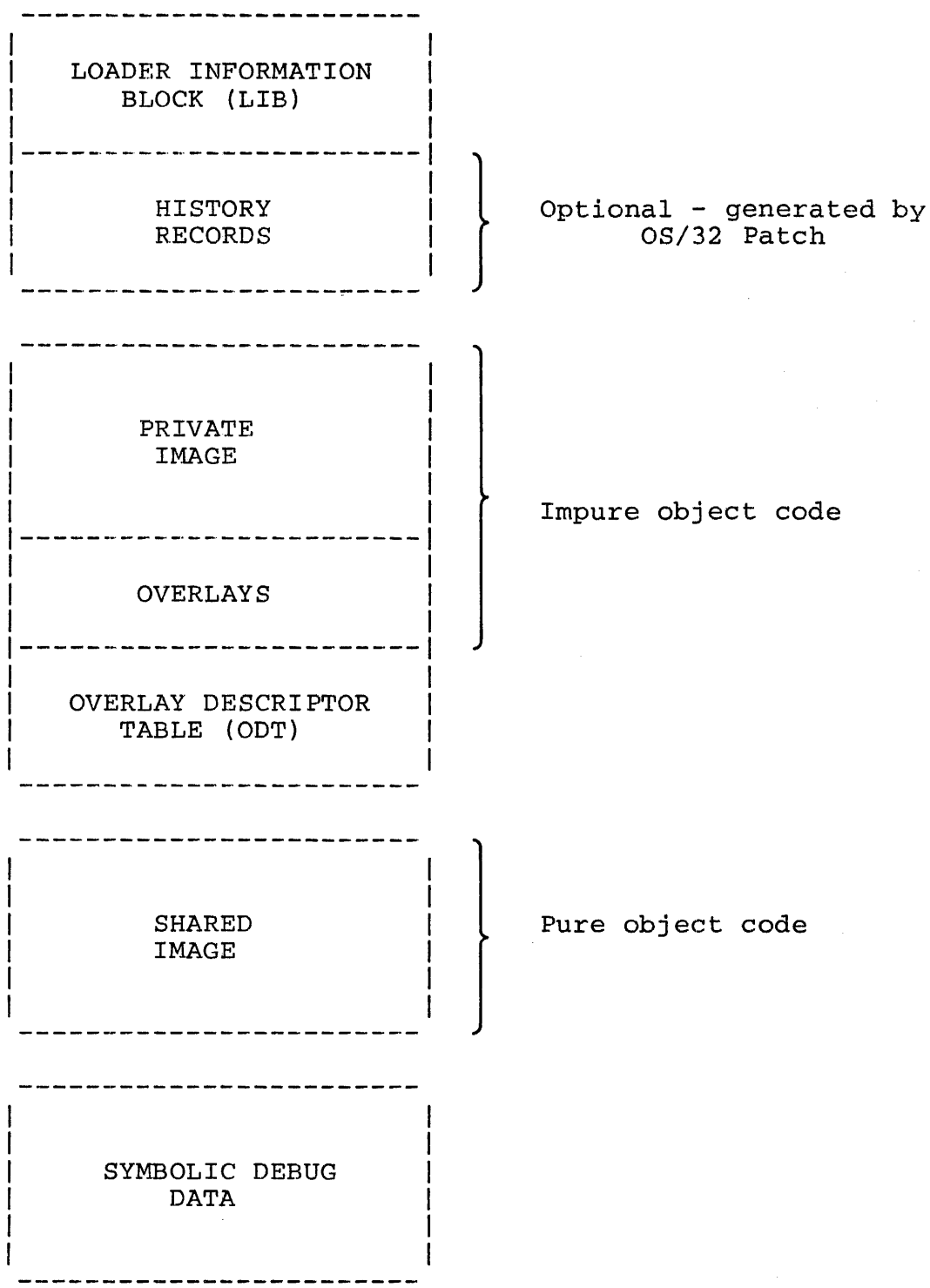


Figure 1-1 Task Image File Format

The first section in the task image file is the loader information block (LIB). The LIB tells the OS/32 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. The history records are created by OS/32 Patch. Patch is a utility that allows the user to update a program by making changes to its image or object file instead of the source. Any changes made to the task or its LIB via Patch are recorded in the history records area. See the OS/32 PATCH Reference Manual for more information.

Following the LIB and the history records (if they exist) is the task image that is actually loaded into memory. This task image consists of at least one private image segment. The linkage editor creates the private image with read, write, and execute privileges. The private image contains the impure code from the included object modules. Impure code is code that cannot be shared by other executing tasks. It can consist of the user program code, data that the user designates as impure, and common data areas such as those used by the FORTRAN COMMON statement to store variables. If NSEGMENTED is specified as a task option in the Link OPTION command when the task is built, the pure code is also included in the private image.

Each user loading 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 primary task workspace, the impure code, the user-dedicated location (UDL), and if the task is nonsegmented, the pure code. In addition, any absolute code found in the object modules is located in the root.

If a task is to use overlays (i.e., after the task is loaded, certain subroutines, or overlays, are to remain in the image file to be fetched into the root as needed), they are formatted in the private image overlay area following the root. Link is instructed to construct overlays through the OVERLAY command (see Chapter 3) in the link sequence.

The overlay descriptor table (ODT) following the overlay area contains instructions that tell the loader when to load the overlays into memory. The ODT is loaded into the task control block (TCB) after the task is loaded. All task status information is stored in the TCB during task execution.

If the SEGMENTED parameter of the OPTION command is specified, all pure code from the object modules is placed in the shared image segment of the the image file. This area has only read and execute access privileges. When the first copy of the segment is loaded into memory, both a private and a shared image segment are created. Each user who loads the task is provided with a copy of the private image. Only one copy of the shared image remains in memory during multiple simultaneous executions of the task.

If the task is debugged using the Symbolic Debugger (DEBUG/32), Link places task data required by the debugger following the shared image segment. This data remains in the image file during task execution so that it is always available for use by the debugger.

A task may require access to subroutines or data areas in addition to those created by the programmer and contained in the task's object modules. OS/32 supports two types of external code and data. One type is an object module such as the FORTRAN or Pascal run-time library (RTL). Routines in object libraries are included in a task's root segment or shared segment using the Link LIBRARY command. The other type of external code or data is called a partial image. A partial image may consist of code (e.g., an RTL routine) or data (e.g., a shared common block) or both. Partial images are built by separate runs of the linkage editor, and each partial image exists in its own image file. A partial image is included in a task's address space by the Link RESOLVE command (see Chapter 3).

The virtual address map of the link establishment summary defines where in the task's logical address space, the root, shared, and partial images are located (see Chapter 3 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 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 (sysgen).)
- @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.

1.4.1 Program Development Using Link-Defined Symbols

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

For more detailed information on program development, see the Multi-Terminal Monitor (MTM) Reference Manual.

Example:

```
*cal linkdemo
*ssysprt con:

     LINKDAY   PROG  Demonstration program
               EXTRN @TIME1,@TIME2,@DATE1,@DATE2
               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'0D',0
               END


*compile linkdemo
*load linkdemo
*start
Linkedited at - 11:29:33 on 02/18/86
TASK PAUSED
*display registers
   PSW  000077F0  00000146
   0-3  00000150  000011FE  00000000  00000000
   4-7  00000000  00000000  00000000  00000000
   8-B  00000000  00000000  00000000  00000000
   C-F  00000000  00000000  00000000  00000000
*continue
DEBBIE  -END OF TASK CODE= 0  PROCESSOR=0.005 TSK-ELAPSED=52
```

1.5 SYSTEM REQUIREMENTS

System requirements for Link are:

- The current OS/32 release (see Preface). (If DEBUG/32 is used, Link requires OS/32 Version R06-01 and higher.)
- One disk device
- 128kB of main storage for Link

1.6 LINK COMMAND SYNTAX

In interactive mode, if the specified parameters of a Link 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 a decimal number as a parameter, the number specified is a positive whole number. Hexadecimal values may be used, if preceded by the letter 'X'. For example, X10 , a hexadecimal 10, is equivalent to a decimal 16.

CHAPTER 2
BUILDING AND STARTING LINK

2.1 BUILDING LINK

If the supplied ready-to-execute version of Link is used, no build is necessary. However, if a new version of Link is built, this sequence of commands builds Link as a segmented task using the supplied version of Link. In the example shown below, xxx is the current version of the OS/32 software. For example, version R08-03 is entered in the format as 083.

```
ESTABLISH TASK
OPTION ACPRIVILEGE,SYSSPACE=XFFFF
OPTION SEGMENTED,WORK=(X8000,XC000)
INCLUDE LINK
MAP CON:,ALPHABETIC,ADDRESS,XREF
RESOLVE HELPRXXX.SEG/S
BUILD LINK
END
```

See the Package Information Document for the current OS/32 revision number.

The reserved workspace must be a minimum of 8kB. The more workspace allocated, the less paging to and from the 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 OS/32 LOAD 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 file containing the linkage editor image to be loaded into main memory. If this parameter is omitted, the default is taskid.TSK.

NOTE

The supplied ready-to-execute version of Link is included in a file named LINK.TSK.

workspace — is a decimal number in kilobytes (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. If this parameter omitted, the default is 32kB.

NOTE

Link requires the OS/32 Help segment (.HELPRXXX.SEG/S- where XXX specifies the current OS/32 release. As an example, Release 08-03 is entered in the format 083) to be preloaded into memory or be available on account 0 of the default system disk volume.

2.2.2 Loading Link From an 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 file containing the linkage editor image to be loaded into main memory.
workspace	is a decimal number in kilobytes (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. If this parameter is omitted, the default is 32kB.

2.2.3 Assigning Workspace for Link

The size of the workspace increment value given when Link is loaded controls 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	32kB
16 - 31	64kB
32 - 63	96kB
64 - 95	128kB
96 - 127	256kB
128 - 255	1MB
256 - or greater	4MB

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 automatically allocates a file if the file does not already exist. The filename entered with the BUILD command is given the extension corresponding to the type of image being built (task, partial image, or operating system). 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	ACCESS
INCLUDE/LIBRARY	1	Object	SRO
BUILD	2	Task image	SRW
MAP	3	Link map	SWO
START			
,COMMAND=	5	Link command input	SRO
	7	Link command output	SWO
,LOG=	6	Log	SWO
HELP	10	Link help File	SRO

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 OS/32 or MTM START command starts execution of the Link program and specifies the command and log files or devices.

Format:

START [,COMMAND=fd] [,LOG=fd]

Parameters:

COMMAND=	fd specifies the input file or device from which Link commands are read. If this parameter is omitted, the default is the command input device (CON:). If the command input device is interactive, all messages generated by Link are sent to it. If the command input file is batch, all Link messages are sent to the file specified by the LOG parameter.
LOG=	fd specifies the output file or device from which all commands are entered and messages generated are written. If the command input file 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:

```
Concurrent Computer Corp OS/32 LINKAGE EDITOR 03-242 Rxx-xx
```

The revision number (Rxx) indicates the revision level of Link and the update number (-xx) 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
- Environmental

Active commands are executed as they are entered and have an immediate affect 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 can affect the image produced by Link. This is due to the order in which items are entered into Link's internal symbol table. Environmental commands affect the link session instead of the image being built. Environmental commands have no affect on the image being built, but do establish the Link environment.

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

TABLE 3-1 LINK COMMANDS

COMMAND	TYPE ACT	TYPE PAS	TYPE ENV	MEANING
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 3200MPS Family of Processors.
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.

TABLE 3-1 LINK COMMANDS (Continued)

COMMAND	TYPE ACT	TYPE PAS	TYPE ENV	MEANING
MAP		*		Generates a map when the image is built.
NDCMD	*			Disables execution of Link commands embedded in object modules. Disables listing of embedded comments to the log device in the 3200MPS Family of Processors.
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

ACT = Active; PAS = Passive; ENV = Environmental

```
-----------
|  BFILE  |
-----------
```

3.2 BACKSPACE (BFILE) COMMAND

The BFILE command is an environmental command that backspaces a magnetic tape or contiguous file a specified number of filemarks.

Format:

BFILE fd [,{n / 1}]

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.

Functional Details:

The 60MB cartridge tape does not support the BFILE command.

Example:

The following example causes the device MAG1: to backspace two filemarks.

```
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 [,ABORT][,ERROR]

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

ABORT — indicates that Link will not build the task image at all if unresolved externs or multiply-defined symbols are encountered. If this occurs, the process aborts with an end of task (EOT) code of 3.

ERROR — indicates that Link will still build the task image if unresolved externs or multiply-defined symbols are encountered. If this occurs, the process will return an EOT code of 2.

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. While in the interactive mode, 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 file or device to receive the image.

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

By default, Link allocates a contiguous file for the task image file. Building an image file to a contiguous file is significantly faster than building an image to an indexed file. However, if not enough contiguous disk space is available for the whole size of the task, an indexed file is allocated for the task image. 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 Y (YES) or N (NO). If YES 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

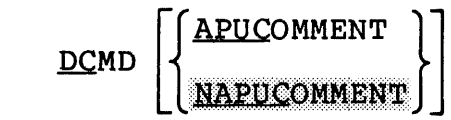
If Link is running in batch mode and cannot allocate the file, the build process is terminated.

| DCMD |

3.4 DEFINE COMMAND (DCMD) COMMAND

The DCMD is an active command that, when entered without parameters, enables execution of passive Link commands in object modules included in the image. This command, at the same time, enables listing of embedded comments to the input or log device. In programs written for the 3200MPS Family of Processors, this command entered with parameters enables or suppresses listing of APU comments to the log device.

Format:



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/32 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/32 program, use the CAL/32 DCMD pseudo-operation as follows:

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

NOTE

This DCMD pseudo-operation is not the same as the Link DCMD command.

Example of CAL/32 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 one.

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

In programs written for any of the 3200MPS Family of Processors, 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 suppressing 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.

Standard warning and error messages produced by a language processor and inserted into the object code are documented in the user guide for that language. Link does not know anything about these messages; neither their contents nor what created them.

When the previous program is linked, the following log listing is displayed:

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

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

```
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 is displayed:

```
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/32 object modules. If active or environmental commands are embedded in CAL/32 object modules, they are ignored and the following message is displayed:

```
COMMAND NOT PERMITTED
```

Application users in a uniprocessor system can use the DCMD command with its parameters for developing any of the 3200MPS Family of Processors.

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:

While Link is in the interactive mode, if a Link command sequence contains at least one INCLUDE command, and the END command is entered before the BUILD command 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 EOT codes.

TABLE 3-2 LINK EOT CODES

EOT 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.

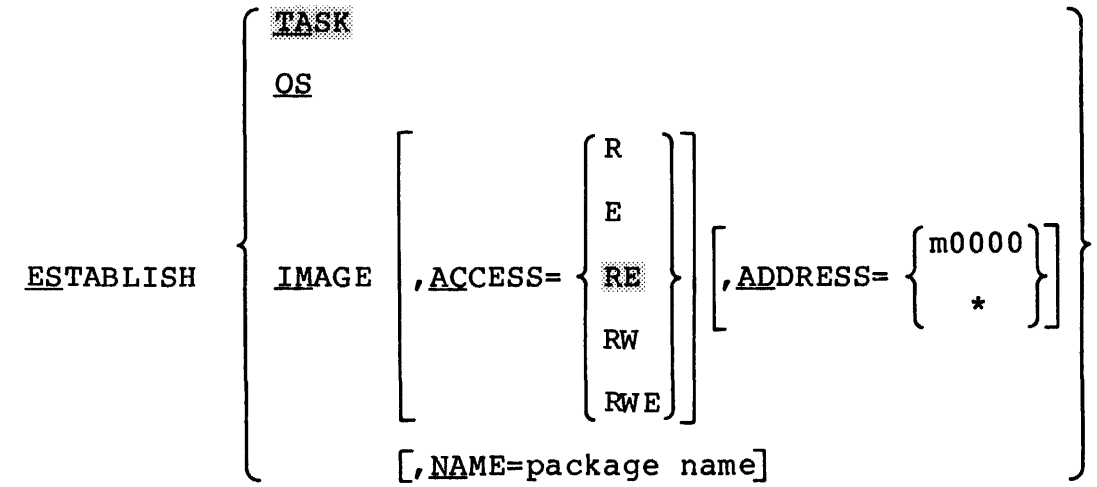
```
 -----------
| ESTABLISH |
 -----------
```

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:



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.
- 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. If the ACCESS parameter is omitted, RE is the default.
- 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 in memory. The variable m is a hexadecimal number from 0 through FF. The address must be greater than CTOP for any task which references this partial image to prevent overlapping of the task impure and the segment to be built. 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 multisegmented task, partial image or operating system. If this parameter is not specified, the fd 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 multisegmented 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 to the partial image.

Functional Details:

If the ESTABLISH command is entered after active commands are entered but before the BUILD command is entered, the following message is displayed:

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

Enter Y (YES) or N (NO). 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 with a relocatable address of F0000.

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

Establish a reentrant library image with RE access privileges and a relocatable address of A0000.

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

Establish a task common image with RW access privileges with a relocatable address of 00000.

```
-----------
| EXTERNAL  |
-----------
```

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

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 FORWARD FILE (FFILE) COMMAND

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

Format:

FFILE fd [,{n / 1}]

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:

The following example causes the device MAG1: to forward space two filemarks.

```
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:

HELP [mnemonic / *]

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.

For some commands (e.g., OPTION), the HELP information will require that more than one screen be displayed. In this case, Link displays a maximum of 23 lines, then prompts for a carriage return (CR) to continue the display. Any character except a CR aborts the remainder of the display.

Example:

```
*LOAD LINK
*START
>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)]
                      [,UN(REFERENCED)]

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.

The 'UNREFERENCED' parameter specifies to pr the names of all
  unreferenced entry points that are part of the map.  By
  default, the printing of these labels is suppressed.

>
```

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:

INCLUDE [fd][-BLKDATA] $\left[\left[,\left\{\begin{matrix}\text{module}_1\\ *\end{matrix}\right\}\right]\ \left[-\left\{\begin{matrix}\text{module}_n\\ *\end{matrix}\right\}\right],\ldots,\text{module}_x\right]$

Parameters:

fd — is the file descriptor of the file or device containing the modules to be included. Either a 126-byte format library or OBJECT/32 format library may be specified. If this parameter is omitted, a preassigned logical unit 1 (1ul) 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 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.

-BLKDATA controls the inclusion of block data. If a partial image is being built, all block data associated with named EXTERNAL common areas will be included in the partial image; all other data will be ignored. If a task is being built, all data will be included except block data associated with common areas residing in partial images.

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.

NOTE

If the NSEGMENTED option is selected, Link writes object modules to the task image in the same order as they are included. However, if SEGMENTED is specified, Link chooses the order of modules in the task image. In this case, modules will normally appear in exactly the opposite order that they were included.

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 specified.

Format:

LIBRARY fd_1 [,...,fd_n]

Parameters:

fd	is the file descriptor of the library to be searched. A 126-byte format or OBJECT/32 format library may be specified. 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, C^{3}Ada, or Pascal, be sure to specify all user libraries before specifying a standard run-time library (RTL). This ensures that each user library routine gets resolved against the standard RTL.

Also, keep in mind that the range of a LIBRARY command is the entire Link command sequence (prior to the next BUILD or ESTABLISH command); 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-operation are not matched against the library. These references are only resolved against entry points to modules that are explicitly included or are included from a library through normal (strong) external references.

Nonlinking external references generated by the INCLD pseudo-operation are matched against module names in the library.

Weak entry points in the library generated by the WNTRY pseudo-operation 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-operation.

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 in that order.

```
-----------
|   LOCAL   |
-----------
```

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 point$_1$ [,...,entry point$_n$]

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] [,UNREFERENCED]

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 descriptor is not preallocated, Link allocates 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.
UNREFERENCED	specifies that all symbols, including those that are not used, will appear in the map(s) requested. Otherwise, the MAP command will show only referenced symbols.

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 a 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 (1024-byte) units.

The symbol maps specified by the MAP command follow the establishment summary. If no map options are specified, the MAP command outputs an establishment summary only. Symbol maps list data areas, 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.

The letters P, I, or A follow the address of each symbol name in the alphabetic address map. 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.

```
Concurrent Computer Corp OS/32 LINKAGE EDITOR 03-242 Rxx-xx ESTABLISHMENT PAGE 1
 -- IMAGE LINKED AT 14:43:11 ON FEBRUARY 26,1989 --

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           NXTENDED

TEQSAVE=ALL  LU=15  SYSSPACE=X3000  WORK=(X50,X40000)  ABSOLUTE=X100
IOBLOCKS=1  PRIORITY=(128,128)  TSW=(X0,X1BC)  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

```
Concurrent Computer Corp OS/32 LINKAGE EDITOR 03-242 Rxx-xx ALPHABETIC MAP PAGE 1
 -- IMAGE LINKED AT 14:43:11 ON FEBRUARY 26, 1989 --

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

```
Concurrent Computer Corp OS/32 LINKAGE EDITOR 03-242 R ADDRESS MAP PAGE 1
 -- IMAGE LINKED AT 14:43:11 ON FEBRUARY 26, 1989 --

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

```
Concurrent Computer Corp OS/32 LINKAGE EDITOR 03-242 Rxx-xx CROSS-REFERENCE PAGE 1
 -- IMAGE LINKED AT 14:43:11 ON FEBRUARY 26, 1989 --

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 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).

```
-----------
|   NLOG    |
-----------
```

3.16 NO LOG (NLOG) COMMAND

The NLOG command is an environmental 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 are in effect during task execution.

NOTE

When a task image created by link is loaded under MTM, certain MTM configurations can override the task options set by the option command. See the Multi-Terminal Monitor (MTM) Reference Manual for more information.

If the syntax of a Link command includes a decimal number as a parameter, the number specified is a positive whole number. Hexadecimal values may be used, if preceeded by the letter "X". For example, X10, a hexadecimal 10, is equivalent to a decimal 16.

Example 1:

```
    ALIGN = 16              LU = 15
```

or

```
    ALIGN = X10             LU = X0F
```

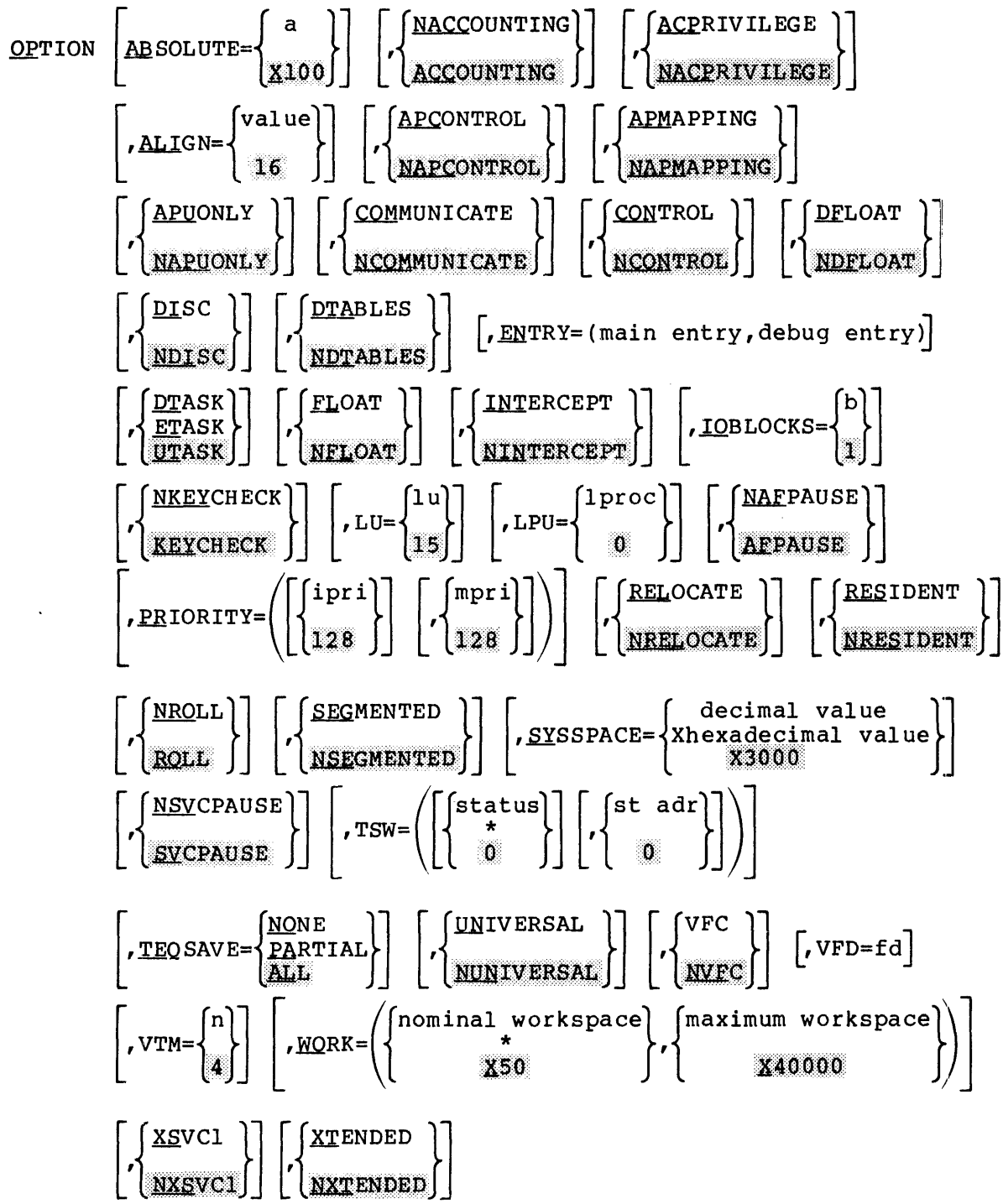
Example 2:

```
    WORK = (80, 262144)     PRIORITY = (128, 90)
```

or

```
    WORK = (X50, X40000)     PRIORITY = (X80, X5A)
```

Format:


OPTION
[ABSOLUTE={a / X100}]
[,{NACCOUNTING / ACCOUNTING}]
[,{ACPRIVILEGE / NACPRIVILEGE}]
[,ALIGN={value / 16}]
[,{APCONTROL / NAPCONTROL}]
[,{APMAPPING / NAPMAPPING}]
[,{APUONLY / NAPUONLY}]
[,{COMMUNICATE / NCOMMUNICATE}]
[,{CONTROL / NCONTROL}]
[,{DFLOAT / NDFLOAT}]
[,{DISC / NDISC}]
[,{DTABLES / NDTABLES}]
[,ENTRY=(main entry,debug entry)]
[,{DTASK / ETASK / UTASK}]
[,{FLOAT / NFLOAT}]
[,{INTERCEPT / NINTERCEPT}]
[,IOBLOCKS={b / 1}]
[,{NKEYCHECK / KEYCHECK}]
[,LU={lu / 15}]
[,LPU={lproc / 0}]
[,{NAFPAUSE / AFPAUSE}]
[,PRIORITY=([{ipri / 128}] [,{mpri / 128}])]
[,{RELOCATE / NRELOCATE}]
[,{RESIDENT / NRESIDENT}]
[,{NROLL / ROLL}]
[,{SEGMENTED / NSEGMENTED}]
[,SYSSPACE={decimal value / Xhexadecimal value / X3000}]
[,{NSVCPAUSE / SVCPAUSE}]
[,TSW=([{status / * / 0}] [,{st adr / 0}])]
[,TEQSAVE={NONE / PARTIAL / ALL}]
[,{UNIVERSAL / NUNIVERSAL}]
[,{VFC / NVFC}]
[,VFD=fd]
[,VTM={n / 4}]
[,WORK=({nominal workspace / * / X50},{maximum workspace / X40000})]
[,{XSVC1 / NXSVC1}]
[,{XTENDED / NXTENDED}]

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. All relocatable code in the impure segment is relocated from this value.

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 EOT code. If the Accounting facility was not specified at sysgen and ACCOUNTING is specified, no accounting data is 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.

NOTE

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.

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).

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 the 3200MPS Family of Processors only. Control of an APU by a task is accomplished through the supervisor call 13 (SVC13) 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 the 3200MPS Family of Processors only and is the default.

APMAPPING specifies that the task can obtain APU mapping privileges. This option is valid for the 3200MPS Family of Processors 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 the 3200MPS Family of Processors 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 central processing unit (CPU), except for supervisor functions, causes the task to pause. This option is valid on the 3200MPS Family of Processors 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 the 3200MPS Family of Processors only and is the default.

COMMUNICATE — specifies that the task can perform the SVC6 intertask communication functions. If this parameter is not specified, the task cannot communicate with other tasks.

NCOMMUNICATE — prevents the task from issuing an SVC6 for intertask communication. If the intertask communication option is not specified, NCOMMUNICATE is the default.

CONTROL — specifies that the task can perform the SVC6 intertask control functions. If this parameter is not specified, the task cannot issue an SVC6 to control the execution of another task.

NCONTROL — prevents the task from issuing an SVC6 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 (DPFP) instructions. If this parameter is not specified, the task cannot execute DPFP instructions.

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

DISC — is the bare disk input/output (I/O) privilege option. This option allows a u-task or d-task to bypass the file manager and directly assign I/O requests to a disk device. If the disk is marked on-line, 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 off-line, 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 in the image for DEBUG/32. This option also increases the number of logical units used by the task, by one. However, LU=15 still appears on the Link map.

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

ENTRY specifies the name of an entry point in the root node or the debug task where execution of the task image begins. 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 for the debug task. The debug entry point specifies the location where execution of the task image begins. In addition, the main entry or default entry is reserved for use by DEBUG/32.

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 can contain only positional-independent pure and impure code and cannot reference partial images. An e-task can execute privileged instructions and reference physical memory addresses.

UTASK specifies that a u-task image is to be built. A u-task cannot execute privileged instructions. If no task type parameter is specified, UTASK is the default.

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

NFLOAT — prevents the task from executing SPFP 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 checks 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 — is a decimal number from 0 through 255 which specifies the maximum number of logical units that can be assigned to a task. The task must be linked with this parameter one greater than the highest lu number quoted in the code. The lu number identifies the highest lu number that can be referenced, not the number of logical units to be assigned. For example, a task linked with 20 logical units may use logical units 0 through 19.

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

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 (LPU) up to maximum of 255. LPU is a sysgen statement. See the System Generation/32 (Sysgen/32) Reference Manual.

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

AFPAUSE — pauses 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 greater or equal numerically 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).

RELOCATE — specifies that a relocation table is to be built within the task image if the ETASK option is also specified.

NRELOCATE — specifies that a relocation table should not be built within the task image if the ETASK option is specified. If the RELOCATE option is not specified, NRELOCATE is the default. See Chapter 6 for more information on the RELOCATE option.

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 is 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 code of a u- or d-task can be shared when more than one copy of the task is loaded. Both a private and a shared-image segment are created when the first copy of the segment is loaded into memory. The pure code is loaded into the pure segment which is shareable by all users. Impure and absolute code are built into the impure segment which is private to the user. If this option is not specified, the pure segment cannot be shared. SEGMENTED is incompatible with the OPTION ETASK parameter.

NSEGMENTED — specifies that the pure code of a u- or d-task is combined with the impure code and cannot be shared when more than one copy of the task is loaded. Pure and impure code are built into the impure segment which is not shareable. NSEGMENTED is the default.

SYSSPACE

(DSS)

specifies the maximum amount of system space that a task can use during execution. System space is used for file control blocks associated with open disk files and other operating system data structures associated with the task. 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 SVC6 is treated as a no-operation (NOP) (applies to .BG tasks only). If a background task issues an SVC6, 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 SVC6.

SVCPAUSE

specifies that SVC6 is treated as an illegal SVC (applies to .BG tasks only). If an SVC6 is issued by a background task, the operating system pauses execution of that task. If the SVCPAUSE 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. If multiple TSW options are specified, an OR operation is performed on the status field 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 loader information block (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.

NOTE

For more information about TSW bit settings, refer to Chapter 3 in the Application Level Programmer Reference Manual.

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 SVC1 or SVC7. See the OS/32 Supervisor Call (SVC) Reference Manual for more information on using SVC1 and SVC7 for VFC.

VFD specifies the secondary storage file for a virtual task. If this option is not specified, VTM allocates 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.

NOTE

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

this can be overridden by h XXX,3Ø4

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) are 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. If SEGMENTED is specified, the shared segment follows the maximum workspace. A task may fail to load on certain systems with limited memory. Care must be taken by the user not to exceed the maximum memory available. Consult the Link map to determine the amount of system space available.

XSVC1 indicates that if the task issues an SVC1 with bit 7 of the function code set, the options specified by the SVC1 extended option field are to be executed for all drivers which use this field. If XSVC1 is not specified, an SVC1 with bit 7 set performs an image I/O transfer. See the OS/32 Supervisor Call (SVC) Reference Manual for more information on the SVC1 function code and extended options.

NXSVC1 indicates that if the task issues an SVC1 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 SVC1 function code and extended options.

NOTE

Data communications devices do not check the extended SVC1 bit in the task options; therefore, the user must assure that the extended options field is set up if bit 7 of the function code is set.

If performing I/O to data communications devices, refer to the OS/32 Basic Data Communications Reference Manual and the OS/32 Supervisor Call (SVC) Reference Manual.

XTENDED enables the task to be loaded into extended task space. Extended task space is memory which resides above the first 16MB boundary. If this parameter is not specified, NXTENDED is the default.

NXTENDED specifies that the task be loaded into non-extended task space. Non-extended task space is memory which resides below the first 16MB boundary.

NOTE

A LOAD command issued with ,XTENDED or ,NXTENDED appended to it overrides the Link OPTION command for loading into extended or nonextended memory. If neither mnemonic is specified, the option specified at Link time is used.

Functional Details:

OS/32 places some restrictions on which tasks can communicate with one another by assigning a group ID to each task. Normally, a task can communicate only with tasks within its assigned group.

Group IDs are assigned according to the operating environment under which a task is loaded. Tasks loaded into an OS/32 real-time environment are divided into two groups: foreground and background. A monitor and its subtasks are assigned to their own group. System tasks (the console monitor, the command processor, MTM, and the spooler) are in a separate group called the systems group.

To communicate with tasks outside its group, a foreground task should be link-edited with the UNIVERSAL task option enabled. OS/32 defines a background task as nonuniversal to prevent it from communicating with tasks outside its group.

Examples:

```
OPTION ACPRIVILEGE,NKEYCHECK,ALIGN=4,
       DFLOAT,LU=10,PRIORITY=(,100),
       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 (DPFP) instructions and assign up to ten logical units. Maximum task priority is 100; initial task priority is 128. VFC is in effect for all I/O operations. The options specified by the SVC1 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 OS/32 or MTM LOAD command, the task is 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 the 3200MPS Family of Processors. The task can execute single precision floating point (SPFP) 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; and can obtain APU control privileges, and APU mapping privileges in a multiprocessor system (MPS). The APC and array paging method (APM) options are valid on the 3200MPS Family of Processors 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:

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

Parameters:

overlay name	is an 8-character alphanumeric string specifying the name of the overlay to be loaded into main storage. The name .ROOT is reserved for the root segment.
level	is a decimal number from 1 through 256 specifying the number of overlays between the overlay being defined and the root (inclusive). The number specified must be 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 are 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).

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 environmental 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:

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

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= nodename 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 a passive 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 or a partial image created by the ESTABLISH command.

Format:

```
RESOLVE    [fd][,NAME=package name]
           [,ACCESS={R|E|RE|RW|RWE}][,ADDRESS=m0000]
           [,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 is not specified, the default extension is .IMG.

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 FF.

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.

$name_1 \ldots name_n$ is an 8-character alphanumeric string specifying the name of the task common block to be structured.

$size_1 ... size_n$ 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.

NOTE

If common blocks in a partial image are declared by using the EXTERNAL command when the partial image is built, STRUCTURE cannot be specified when resolving against that partial image. Doing so results in the following message: COMMON XXXXXXXX ENCOUNTERED IN MORE THAN ONE PARTIAL IMAGE.

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 from 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., XFO.)

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 replaces max with min and continues 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.

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 environmental 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:

This example causes the tape on device MAG1: to be rewound.

```
REWIND MAG1:
```

3.23 TITLE COMMAND

The TITLE command is an environmental 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 'TINTON FALLS DEVELOPMENT GROUP'
TI 'DEPARTMENT 3145'
```

| VOLUME |

3.24 VOLUME COMMAND

The VOLUME command is an environmental 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 environmental command that writes a filemark on a magnetic tape or contiguous file.

Format:

WFILE fd [,{n / 1}]

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 images, operating system images, and partial 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/32 (CAL/32) assembler. MOD1.OBJ has no external references. The task built consists of one impure segment.

Example:

```
ESTABLISH 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. Because the OPTION command is not specified, the defaults are in effect. See Chapter 3 for a description of the OPTION command.

4.3 BUILDING FORTRAN, COBOL, AND COMMON ASSEMBLY LANGUAGE/32 (CAL/32) TASK IMAGES

This section provides examples for building COBOL, FORTRAN, and CAL/32 task images, linking subroutine libraries, outputting Link maps, using the OPTION command, and embedding Link commands in object modules.

There are three types of images that can be built through the use of the ESTABLISH command: task, operating system, and partial images.

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 includes the single precision floating point (SPFP) capability. A map is generated listing the names and locations of all modules and entry points in address order.

Example:

```
ESTABLISH 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 included in the image.

The LIBRARY command specifies that the COBOL run-time library (RTL) file, COBOL.LIB, is searched and any routines that contain entry points matching external references are 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 SPFP capability is included as part of the task image. All other parameters of this command take their default values.

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 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 includes both SPFP and double precision floating point (DPFP) capabilities and additional workspace for the user and standard RTLs.

Both cross-reference and alphabetic Link maps are 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 included in the image.

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

The OPTION command specifies that the SPFP and DPFP capabilities and 4840 bytes of additional workspace for the RTLs are included as part of the task image. All other parameters of the command take their default values.

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/32 (CAL/32) Task Image Using Embedded Link Commands

The following example builds a task image from the CAL object module, MOD4.OBJ, containing external references and embedded Link commands. The image includes SPFP and DPFP capabilities. An establishment summary, cross-reference, and alphabetic maps are output to the printer.

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

Example:

```
NDCMD;*IGNORE EMBEDDED COMMANDS IN MOD4
INCLUDE MOD4; LIBRARY USERLIB
OPTION DFLOAT,FLOAT,WORK=4840
MAP PR1:,ALPHABETIC,XREF
DCMD;*PROCESS EMBEDDED COMMANDS IN LIBRARY MODULES
BUILD MOD4
END
```

Link accepts passive commands that are 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 embedded in object modules specified by an INCLUDE command are treated as if they were entered immediately after the INCLUDE command. Commands embedded 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 embedded commands to be ignored. The DCMD command enables this feature.

4.4 BUILDING OVERLAYED TASK IMAGES

This section discusses building overlayed task images. 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 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, which 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 4-1 is a graphic example of an overlay tree structure.

Sample FORTRAN 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

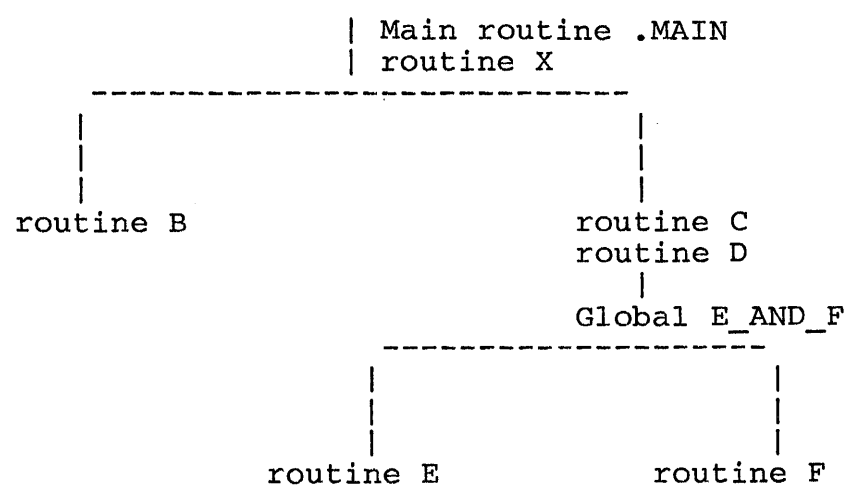


Figure 4-1 Sample FORTRAN Program with Overlay Tree Structure

The sample FORTRAN 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 4-1.

```
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 INCLUDE command specifies that the object module .MAIN in the input file MYPROG.OBJ is to be included in the image. .MAIN resides in the root node. Because the following INCLUDE command is specified before any OVERLAY command, X is placed in the root node. This is done since all other routines call X.

The OVERLAY command specifies the start of a node and the node's relative position within the tree structure. The first OVERLAY command defines an overlay area named B with a depth level of one. The INCLUDE command specifies that the object module called B is part of overlay B.

The second OVERLAY command defines an overlay area named CD with a depth level of one.

The third and fourth OVERLAY commands define overlay areas named E and F with a depth level of two, which indicates that these overlays are descendants of overlay CD.

The LIBRARY commands specify that the two RTL files, MYLIB and the standard RTL, are to be 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 4-1) as an aid in determining the smallest possible task size.

The BUILD command builds the image, which consists of the root segment, overlay areas, and the subprograms.

4.4.2 Moving Common Blocks

Normally, the placement of a common block or global blocks 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 FORTRAN program in Figure 4-1, 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
```

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

Example:

```
INCLUDE MYPROG.OBJ,.MAIN
OVERLAY B,1
  .
  .
  .
LIBRARY MYLIB.OBJ
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 shared data areas which contain block data modules and shared code segments, must be separately built by Link prior to being used or referenced by tasks. A partial image is a single task segment that can be used by one or more separate tasks. A partial image has no user-dedicated location (UDL).

To build a partial image, the IMAGE parameter is specified in the ESTABLISH command. The ACCESS parameter of the ESTABLISH command is then used to specify the access privileges of the partial image. The access privilege can be read (R), execute (E), read and execute (RE), read and write (RW), or read, write, and execute (RWE). The ADDRESS parameter of the ESTABLISH command specifies the starting address of the partial image segment. The NAME parameter of the ESTABLISH command specifies a package name for a partial image.

4.5.1 Linking and Using Shared Data Areas

A program can reference data areas that can be read or written to by other tasks running on the same or different processors.

Shared data areas must be built and linked into shared image modules before they can be specified in the RESOLVE command. To build a data area, use a FORTRAN block data subprogram.

Example:

```
C THIS BLOCK DATA SUBPROGRAM BUILDS
C A DATA AREA CONSISTING OF BOTH
C NAMED COMMON AND GLOBAL COMMON
C VARIABLES

      BLOCK DATA    DATA1
      GLOBAL A,B,C,D,E
      COMMON /ABC/I,J,K
      COMMON /DEF/L,M,N
      REAL A,B,C,D,E
      DATA A,B,C,D,E,I,J,K,L,M,N/5*0.0,6*0/
      END
```

WARNING

BECAUSE OS/32 DOES NOT SUPPORT STATIC INITIALIZATION WITHIN GLOBAL TASK COMMON AREAS, BLOCK DATA SUBPROGRAMS USED FOR STRUCTURING SHARED DATA WITHIN GLOBAL TASK COMMON MUST NOT CONTAIN DATA STATEMENTS.

The following sequence of Link commands are to be used to establish a block data structure as a shared data area.

```
ESTABLISH IMAGE,ACCESS=RW
INCLUDE DATA1
EXTERNAL ABC.,DEF.,A,B,C,D,E
BUILD DATA1.IMG
END
```

In the previous command sequences, DATA1 is not only the name of the block data subprogram but also the name of the file containing the object code for the subprogram.

The ESTABLISH command specifies that a shared data area is to be built with read/write access privileges.

The INCLUDE command specifies that the object modules in the input file DATA1.OBJ are to be included in the shared data area.

These Link commands establish DATA1.IMG as a shared data area containing DATA1. Items within the shared area are arranged exactly as they are arranged within the block data structure. Each shared area can contain more than one block data structure. These structures are arranged within the shared area according to the order in which they are included by the Link INCLUDE command.

When establishing the shared area, all global variables and named common blocks whose names should be externally accessible must be listed in the Link EXTERNAL command. Common block names are limited to eight characters. If a name is less than eight characters, a period must be appended by the user to the name (e.g., ABC. and DEF.). There is no need to add the period to a global name.

When establishing a shared data area that is to be located in the global task common (memory shared by two or more processors), use the name of the global task common as the argument to the Link BUILD command. This name is determined by the TCOM command at sysgen. For example, if DATA1 is to be established as a shared area within a global task common named GTC, the Link BUILD command would be written as follows:

```
BUILD GTC
```

Where:

GTC	is the name given by the system administrator at sysgen time to that shared data area.

To link a FORTRAN task that can reference a shared data area, use the Link RESOLVE command as shown in the following example.

Example:

```
ESTABLISH TASK
MAP MOD7.MAP,XREF
OPTION DFLOAT,FLOAT,WORK=(XC00,XC00),
       SYSSPACE=XFFFF
INCLUDE MOD7.OBJ
RESOLVE DATA1.IMG
LIBRARY F7RTL.OBJ/S
BUILD MOD7.TSK
END
```

The MAP command generates an establishment summary, a 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 OPTION command specifies that the image includes both SPFP and DPFP capabilities and additional workspace for the user and additional system space for the task.

The INCLUDE command specifies that the object modules in the input file, MOD7.OBJ, are included in the image.

The RESOLVE command is used to establish a FORTRAN task image that references the shared area, DATA1. The LIBRARY command specifies that the FORTRAN RTL file, F7RTL.OBJ, is to be searched. When a program is linked, external references that were not resolved by the INCLUDE and RESOLVE commands are matched against the library 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.

The BUILD command builds the task image and stores it in file MOD7.TSK.

The END command terminates the linkage editor.

The following example includes two blockdata object modules called BDALPHA.OBJ and BDBETA.OBJ to initialize common blocks called ALPHA and BETA.

Example:

```
ESTABLISH IMAGE,ACCESS=RW,NAME=COMMONS
INCLUDE BDALPHA.OBJ
INCLUDE BDBETA.OBJ
EXTERNAL ALPHA,BETA
BUILD COMMONS.IMG
```

The 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 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.

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

The OS/32 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.

4.5.2 Linking and Using Shared Code Segments

If more than one task will be using a reentrant RTL, the RTL, or individual modules of it, can be included in a shared code segment. This segment can be built by the user as shown in the following example.

Example:

```
ESTABLISH IMAGE,ACCESS=RE,ADDRESS=F0000
INCLUDE F7RTL.OBJ,.ATAN
INCLUDE ,.SIN
INCLUDE ,.COS
INCLUDE ,.U
INCLUDE ,.V
LIBRARY F7RTL
BUILD FORTLIB.IMG
END
```

The ESTABLISH command specifies that a partial image is built with read/execute access privileges.

The INCLUDE command specifies that the object module, .ATAN, in the input file, F7RTL.OBJ, is included in the image. The following INCLUDE commands include modules .SIN, .COS, .U, and .V in the file F7RTL.OBJ specified in the previous INCLUDE command.

The LIBRARY command specifies that the FORTRAN RTL file, F7RTL.OBJ, is searched.

The BUILD command builds the partial image from the object modules specified in the INCLUDE commands and stores it in the file FORTLIB.IMG.

The END command terminates the linkage editor.

By building the Link command file as shown in the following example, references to shared code segments specified by the RESOLVE command are placed in the FORTRAN task. The shared code segment, in this case FORTLIB.IMG, must be available at program execution.

Example:

```
RESOLVE FORTLIB.IMG
LIBRARY F7RTL.OBJ/S
INCLUDE MOD3
OPTION DFLOAT,FLOAT,WORK=X1770
MAP PR1:,ALPHABETIC,XREF
BUILD MOD3
END
```

The following example includes an object file called F7RTL.OBJ to be included in a partial image that includes local entry points.

The LOCAL command of Link is used to establish entry points to shared code segment that can only be referenced by that segment.

```
ESTABLISH IMAGE,ACCESS=RE,ADDRESS=F0000
INCLUDE F7RTL.OBJ
LOCAL .DI,.DO,.TGD,.TASKID,.HYDEX,.HYEXP
BUILD F7RTL.IMG
END
```

The ESTABLISH command specifies that a partial image, F7RTL.IMG, is built with read-execute access privileges only. The ADDRESS parameter specifies that this segment is placed 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 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 BUILD command builds the partial image and stores it in file F7RTL.IMG.

The END command terminates the linkage editor.

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 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 included in the image.

The first RESOLVE command specifies that COMMON.IMG is the file containing a partial image called COMMONB. This file 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 allocated for the common block DELTA. The next 128 bytes after the first 4,096 bytes are 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.

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.

The second RESOLVE command specifies that another partial image is 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 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 SYSGEN/32. MTSYSTEM.OBJ contains no external references. A map is generated listing the names and locations of all symbols, program labels 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). n is the current revision of the OS/32 operating system.

The END command terminates the linkage editor.

CHAPTER 5
VIRTUAL TASK MANAGEMENT (VTM)

5.1 INTRODUCTION

VTM provides a 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/32 (CAL/32) and Pascal programs with some code restrictions.

VTM uses the memory address translator (MAT) to optimize run-time performance. For the 3280MPS and 3280E MPS Systems, VTM uses the virtual address translator (VAT). 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 processors equipped with MAT or VAT hardware, and OS/32 R06-02 software version and higher. Models 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 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 used as secondary storage for the virtual task.

If the VFD option is not entered, VTM allocates a temporary contiguous 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 minimum size of fd is (CTOP/256)-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 ,VTM32.OBJ, supplied with the operating system package. The VTM module is approximately 8kB in size.

5.3.4 Virtual Task Workspace

All 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 Management (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 allowed to default to 15 logical units, 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 VTM task roll eligibility after loading and before starting the task by entering the following command:

```
MODIFY 104,1
```

5.3.8 Absolute Code

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

5.4 FORTRAN OPERATIONAL RULES

The following are FORTRAN operational rules 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/32 (CAL/32) RESTRICTIONS

Supervisor call 1 (SVC1) 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, declare file variables 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 on the number of I/Os 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

VTM error conditions result in the task being paused or cancelled with end of task (EOT) code of 1 and an appropriate error message. A summary of VTM error messages is presented in Appendix C.

CHAPTER 6
RELOCATION WITHIN EXECUTIVE TASKS (E-TASKS)

6.1 INTRODUCTION

A relocatable e-task is a method by which a programmer is freed from the traditional restrictions associated with writing an e-task. Within a relocatable e-task, a programmer can specify address constants and RX3 instructions without having to relocate them manually from within the program. The programmer can also write programs in modules, which means that overlays may also be developed. Furthermore, all relocation is user-transparent, so the programmer does not have to worry about any special housekeeping or additional memory requirements.

6.2 WRITING AND LINKING A RELOCATABLE EXECUTIVE TASK (E-TASK)

This section describes how to take advantage of relocation within e-tasks.

6.2.1 Features of and Restrictions on Relocation

Any 3- or 4-byte relocatable address can be used within a relocatable e-task; this includes address constants, RX3 and RI2 instructions and EXTRNs. Halfword address constants and absolute data will not be relocated. Because of the previous restriction, only programs written in Common Assembly Language/32 (CAL/32) can be used with relocation, since some of the object code generated by the compilers for other programming languages can reference absolute address locations. All other restrictions associated with e-tasks apply (for example, shared segments cannot be specified); the exception is that the program can be linked with overlays. Common blocks can also be referenced as long as the common block is not linked as a sharable segment.

E-task relocation has been available since the OS/32 R08-01 software version. Attempting to execute a relocatable e-task on an earlier version of the operating system will cause unknown results and possibly crash the system.

6.2.2 Declaring an Executive Task (E-Task) as Relocatable

The user specifies an e-task to be relocatable at Link time via the OPTION command:

```
OPTION ETASK,RELOCATE
```

Both options must be specified in order for the relocation tables to be built. If ETASK is omitted, the RELOCATION option is ignored; if RELOCATION is omitted, an e-task is established but no relocation tables are built.

NOTE

Do not attempt to run any utility as a relocatable e-task since some of the utilities make references to absolute address locations.

6.2.3 Example of Linking a Relocatable Executive Task (E-Task)

The following example illustrates one possibility for linking an e-task as relocatable:

```
ESTABLISH TASK
OPTION ETASK,RELOCATE
INCLUDE PROG
INCLUDE SUB1
INCLUDE SUB2
LIBRARY PROG.LIB
BUILD PROG
END
```

Note that the only consideration in establishing the task was that the ETASK and RELOCATE options were specified.

6.3 FUNCTIONAL DETAILS

Link builds a relocation table for the main program and a relocation table for each overlay built. The relocation table consists of fullword address constants pointing to the data within the task image to be relocated. For the main program, this table is found within the task image file after the program itself and is pointed to within the loader information block (LIB) by the label LIB.PRN. For each overlay, the table is found on the first record following the overlay. The last entry within each relocation table is a -1 (Y'FFFF FFFF'); this is a sentinel to signal the end of the table. An entry which has its high-order byte set to X'01' is called an ignored entry, because the loader ignores that entry at load time.

6.4 MEMORY REQUIREMENTS

A relocatable e-task requires no additional memory to run since the relocation tables are used only at load time. However, an extra 256-byte buffer is reserved within the task control block (TCB) for a task established with overlays; this buffer is used to read the relocation table every time an overlay is loaded.

CHAPTER 7
THE OBJECT/32 UTILITY

7.1 INTRODUCTION

OBJECT/32 is a utility that greatly improves the performance of Link. When a library is processed through the OBJECT/32 Utility, its format is restructured from a 126-byte record length to a 2048-byte record length. In addition, modules in the library are put into an indexed file, and all labels associated with the modules appear in the beginning of this file.

The OBJECT/32 format library significantly improves Link's processing time because Link no longer has to read an entire object file to locate modules, entry points, externs, etc. All of the information Link needs is contained at the top of the library in the indexed file.

Link is modified to recognize and take advantage of the "fast-style" OBJECT/32 libraries. No changes are necessary to Link commands to process OBJECT/32 libraries. OBJECT/32 libraries are processed the same as any other library by specifying them in Link's INCLUDE and LIBRARY commands.

Using OBJECT/32 is optional. Link recognizes the current 126-byte format libraries or the 2048-byte format of an OBJECT/32 library. Processing an OBJECT/32 library through Link is transparent to the user except for the noticeable decrease in processing time.

Once a library is converted into OBJECT/32 format, it is not necessary to retain the 126-byte version. The OS/32 DELETE command can be used to delete unwanted libraries and save disk space. If necessary, the library can be returned to 126-byte format by using the OBJECT/32 EXTRACT command.

7.2 OBJECT/32 FUNCTIONALITY

The OBJECT/32 Utility has many features to aid in the creation of OBJECT/32 libraries. OBJECT/32 commands allows the user to:

- Establish new libraries as OBJECT/32 libraries.
- Include modules from other libraries (either OBJECT/32 format or 126-byte format) into an OBJECT/32 library.

- Replace modules from other libraries (either OBJECT/32 format or 126-byte format) into an OBJECT/32 library.
- Display a directory of modules in an OBJECT/32 library in alphabetical or sequential order. A module cross reference is also available.
- Delete modules or a range of modules from an OBJECT/32 library.
- Extract OBJECT/32 modules to a file and convert them back to 126-byte format (for use by the PATCH/32 Utility).
- Assign a descriptive title to an OBJECT/32 library.

The last section in this chapter provides examples of OBJECT/32 sessions.

7.3 LOADING AND STARTING OBJECT/32

Because OBJECT/32 and Link are separate programs, the Link software does not have to reside on your system to use OBJECT/32.

7.3.1 Loading OBJECT/32

The following OS/32 LOAD command loads OBJECT/32 from the system console:

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

Parameters:

taskid	is the 1- to 8-character alphanumeric string specifying the name of the OBJECT/32 task after it has been loaded into main memory.
fd	is the file descriptor of the file containing the OBJECT/32 image to be loaded into main memory. If this parameter is omitted, the default is taskid.TSK.
workspace	is a decimal number (in kilobytes) which specifies the additional area of workspace to be added for OBJECT/32 processing.

The following OS/32 LOAD command loads OBJECT/32 from a Multi-Terminal Monitor (MTM):

LOAD fd[,workspace]

Parameters:

fd	is the file descriptor of the file containing the OBJECT/32 image to be loaded into main memory.
workspace	is a decimal number (in kilobytes) which specifies the additional area of workspace to be added for OBJECT/32 processing.

7.3.2 Starting OBJECT/32

After OBJECT/32 is loaded, the OS/32 or MTM START command starts the OBJECT/32 Utility. A COMMAND file or device, LIST file or device, or LOG file or device can be specified with the START command.

Format:

START [,COMMAND=fd] [,LIST=fd] [,LOG=fd]

Parameters:

COMMAND	fd specifies the input file or device from which OBJECT/32 commands are read. If this parameter is omitted, CON: (the command input device) is the default. An error is generated if fd cannot be assigned. If the command input device is interactive, all messages are sent to it. If the command input file is batch, all messages are sent to the file specified by the LIST parameter.
LIST	fd is the file or device to be used for utility output. An error is generated if fd cannot be assigned. If this option is omitted, CON: is the default.
LOG	is the file or device used for logging a copy of all OBJECT/32 input commands. An error is generated if fd cannot be assigned. If this option is omitted, CON: is the default. Error messages and other output will be directed to the LIST file or device, not the LOG file or device.

Functional Details:

After OBJECT/32 is started, the following message is displayed:

```
Concurrent Computer Corp OBJECT/32 03-929 Rxx-xx
```

The revision number (Rxx) indicates the revision level of OBJECT/32 and the update number (-xx) indicates the update level of OBJECT/32. If the command device is interactive, a greater than sign (>) is displayed to indicate that OBJECT/32 is ready for command input.

7.4 OBJECT/32 COMMANDS

OBJECT/32 commands are listed on the following pages. These commands allow you to create and maintain OBJECT/32 libraries.

There are several unique command conventions associated with OBJECT/32:

- The format mod_a-mod_b specifies a range of modules. Both mod_a and mod_b are optional parameters. If a starting module is not specified, the first module in the library is assumed. If an ending module is not specified, the last module in the library is assumed. For example, the DELETE command may be entered with the following formats:

  ```
  DELETE modulename1-modulename5

  DELETE -modulename3
  ```

  The first example will delete the range of modules specified by the starting and ending module names.

  Since a starting module name was not specified in the second example, the first module in the library is assumed. All modules from the beginning of the library up to and including modulename3 are deleted.

- For commands that accept module ranges as valid input, entering a dash (-) without entering starting or ending module names refers to all modules in the library. For example, DELETE - will delete all modules in the library.

7.5 COMMAND COMMAND

The COMMAND command reassigns the command input file or device.

Format:

COMMAND [=] { fd / CON: }

Parameters:

fd	is the file or device used for command input. The previous command input file (defined by the START option) is closed. An error is generated if fd cannot be assigned. If a file is not specified, CON: (the console device) is the default.

7.6 DELETE COMMAND

The DELETE command deletes modules from the current OBJECT/32 library.

Format:

$$\underline{DE}\text{LETE} \left\{ \text{mod}_1 \left[,\text{mod}_2 \left[,\ldots ,\text{mod}_n \right] \quad \left[,[\text{mod}_a]-[\text{mod}_b] \right] \right] \right\}$$

$$-$$

Parameters:

mod	represents the module or range of modules to be deleted from the current library.
-	a dash (-) allows you to delete all modules from the current OBJECT/32 library.

Functional Details:

More than one module can be deleted by separating the module names with commas (,). Modules are deleted in the sequence in which they appear in the library. To view the modules in a particular library, use the DIRECTORY command (with the BRIEF parameter).

The DELETE command also allows you to delete a range of modules by entering starting and ending module names separated by a dash. Since modules are deleted in the sequence in which they appear in the library, all modules between the starting and ending module names are deleted.

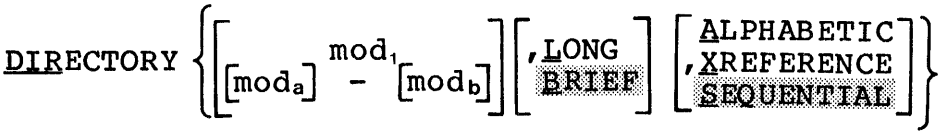
To delete all modules in the current library, simply enter a dash (-) (e.g., DELETE -). A dash indicates to the utility that there is no starting or ending module; therefore, all modules should be deleted.

An error is generated if an entered module does not exist in the current OBJECT/32 library.

7.7 DIRECTORY COMMAND

The DIRECTORY Command displays a directory of modules in the current OBJECT/32 library to the LIST device.

Format:



Parameters:

mod	represents the module name or range of module names to be displayed from the current OBJECT/32 library.
LONG BRIEF	defines the type of information to be displayed about the entered modules. Both parameters (LONG and BRIEF) display the module name and the date in which the module was included or replaced in the library. In addition to this information, the LONG parameter displays all embedded Link commands, defined location counters, and defined labels (entry points, externs, commons). If this parameter is omitted, BRIEF is the default. If the XREFERENCE parameter is specified as the sequence in which to display the modules, the BRIEF and LONG parameters are ignored.
ALPHABETIC XREFERENCE SEQUENTIAL	defines the sequence in which the modules are displayed. If ALPHABETIC is specified, modules are displayed in alphabetical order. If XREFERENCE is specified, a cross reference is generated which lists all entry points and the module(s) in which they are defined. Specifying XREFERENCE as the method of displaying modules negates the use of the BRIEF and LONG parameters. If SEQUENTIAL is specified, modules are displayed in the order in which they are included in the library. If this parameter is omitted, SEQUENTIAL is the default.

Functional Details:

An error is generated if one of the specified modules does not exist in the library.

Examples:

Listed below is an example of the DIRECTORY command using the BRIEF and SEQUENTIAL options (the defaults):

```
>DIRECTORY

Concurrent Computer Corp OBJECT/32  03-929  Rxx-xx          Page     1
Date Run: Fri Jan 06 13:57:47 1989                    Sequential Map

Library Name:  M300:ABC.LIB/120

Title: OBJECT/32 Library ABC

Module Name: C          Date Included: Fri Jan 06 13:51:54 1989
Module Name: A          Date Included: Fri Jan 06 13:51:54 1989
Module name: B          Date Included: Fri Jan 06 13:51:54 1989
```

Listed below is an example of the DIRECTORY command using the BRIEF and ALPHABETIC options:

```
>DIRECTORY ,ALPHABETIC

Concurrent Computer Corp OBJECT/32  03-929  Rxx-xx          Page     1
Date Run: Fri Jan 06 13:57:47 1989                    Alphabetic Map

Library Name:  M300:ABC.LIB/120

Title: OBJECT/32 Library ABC

Module Name: A          Date Included: Fri Jan 06 13:51:54 1989
Module Name: B          Date Included: Fri Jan 06 13:51:54 1989
Module name: C          Date Included: Fri Jan 06 13:51:54 1989
```

Listed below is an example of the DIRECTORY command using the LONG option (the default is SEQUENTIAL) for Module C:

```
>DIRECTORY C,LONG

Concurrent Computer Corp OBJECT/32  03-929  Rxx-xx          Page     1
Date Run: Fri Jan 06 13:57:47 1989                    Sequential Map

Library Name:  M300:ABC.LIB/120

Title: OBJECT/32 Library ABC

Module Name C:          Date Included: Fri Jan 06 13:51:54 1989

Location Counter Information:
Loc #      Size       Section Name      Data Pool Name
    1      44:I         IMPTOP

Defined Labels:

Name       Type      Loc #      Address     Offset      Size
C          ENTRY         1          A:I
B          EXTRN         1         40:I         64
A          WXTRN         1         3C:I          0
B          EXTRN         0          0:A
```

Listed below is an example of the DIRECTORY command using the XREFERENCE option:

Listed below is an example of the DIRECTORY command using the XREFERENCE option:

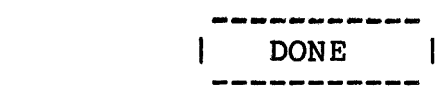
```
>DIRECTORY ,XREFERENCE

Concurrent Computer Corp OBJECT/32  03-929  Rxx-xx          Page     1
Date Run: Fri Jan 06 13:57:47 1989              Cross-Reference Map

Library Name:  M300:ABC.LIB/120

Title: OBJECT/32 Library ABC

Name           Referenced
A              WX-C          WN-A          EX-B
B              EX-C          EN-B
C              EN-C          EX-A
D              DN-B
```



7.8 DONE COMMAND

The DONE command saves changes made to the current OBJECT/32 library and terminates the OBJECT/32 Utility.

Format:

DONE

Functional Details:

If changes are made to the OBJECT/32 library, the DONE command has the combined effect of the SAVE and END commands.

If changes are not made to the library, the DONE command has the same effect as the END command.

7.9 END COMMAND

The END command terminates the OBJECT/32 Utility.

Format:

```
END
```

Functional Details:

If changes are made to an OBJECT/32 library (by using the INCLUDE, REPLACE, DELETE, or TITLE commands), a SAVE command must be issued first to save the changes made. If a SAVE command is not issued before the END command, a message is displayed "Please issue a SAVE command first". If the END command is entered a second time, OBJECT/32 terminates and all changes made to the library are lost.

7.10 ESTABLISH COMMAND

The ESTABLISH command allocates an OBJECT/32 library to be built and establishes the library as the current input library.

Format:

ESTABLISH fd [,title]

Parameters:

fd	is the OBJECT/32 library to be built. An error is generated if the filename already exists.
title	is an optional title to be given to the library for descriptive purposes. This library name is displayed as part of the DIRECTORY command.

Functional Details:

The INCLUDE command is used to add modules to a newly established object library.

7.11 EXTRACT COMMAND

The EXTRACT command extracts a copy of a specified module(s) from the OBJECT/32 library and converts them to the 126-byte format.

Format:

$$\underline{EX}\text{TRACT}\ \ fd\left[,\ \left[\begin{matrix}mod_1\ \left[,mod_2\ \left[,\ldots\ \left[,mod_n\right]\right]\right] \\ \left[mod_a\right]-\left[mod_b\right]\end{matrix}\right]\right]$$

Parameters:

fd	is the file to which the specified module(s) are dumped. If the file does not already exist, it is allocated automatically.
mod	is the module name(s) or range of module names to be extracted and returned to the 126-byte format. Modules are extracted from the OBJECT/32 library in sequential order. An error is generated if the specified modules do not exist in the current library.

Functional Details:

If module names are not specified with the EXTRACT command, the entire current OBJECT/32 library is copied to the 126-byte format and dumped to the specified filename.

The EXTRACT command must be used to extract modules that are patched using the Patch Utility. Patch will not recognize an OBJECT/32 module format. Therefore, the modules must be extracted and returned to the 126-byte format for use by PATCH/32.

7.12 GET COMMAND

The GET command retrieves an OBJECT/32 library for processing and establishes it as the current library.

Format:

GET fd

Parameters:

fd	is the OBJECT/32 library to assign for processing. If the library is already assigned by another GET command, the file is closed first. An error is generated if the file cannot be assigned or if it is not an OBJECT/32 library.

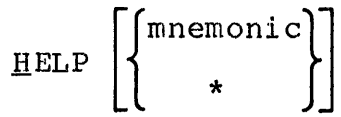
Functional Details:

Some OBJECT/32 commands (e.g., DELETE, DIRECTORY, EXTRACT, INCLUDE, REPLACE, SAVE, and TITLE) issue the message, "GET or ESTABLISH has not been issued yet" if the GET command is not issued first to retrieve (or establish) an OBJECT/32 library for processing.

7.13 HELP COMMAND

The HELP command provides a list of all OBJECT/32 commands. HELP also describes the syntax and function of each command.

Format:



Parameters:

mnemonic	is the mnemonic for an OBJECT/32 command to be described by HELP.
*	lists all OBJECT/32 commands.

Functional Details:

If a list device has been specified in the START command for OBJECT/32, HELP outputs all lists and descriptions of the OBJECT/32 commands to the list device.

For some commands, the HELP information will require more than one screen. In this case, OBJECT/32 displays a maximum of 23 lines, then prompts for a carriage return (CR) to continue the display. Any character except a carriage return ends the remainder of the display.

Example:

```
*LOAD OBJECT32,256
*START
Concurrent Computer Corp OBJECT/32  Rxx-xx
>help *

C(OMMAND)     DE(LETE)          DI(RECTORY)       DO(NE)
END           ES(TABLISH)       EX(TRACT)         G(ET)
H(ELP)        I(NCLUDE)         LI(ST)            LO(G)
P(AUSE)       R(EPLACE)         S(AVE)            T(ITLE)

For HELP on any of the above command mnemonics,
type HELP <mnemonic>

>help save

    SAVE:

    This command saves the current input library along with any
    changes which were made to the library

    Format:

        S(AVE) fd

    fd  specifies the file to be written.  If "*" is specified,
        the file will be saved  under the  name of the  current
        input file.  If a filename is specified, the  file must
        not already exist.

>
```

7.14 INCLUDE COMMAND

The INCLUDE command reads specified modules into the current OBJECT/32 library.

Format:

$$\underline{\text{IN}}\text{CLUDE}\ \ fd\left[,\ \left[mod_1\ \left[,mod_2\ \left[,\ldots\ \left[,mod_n\right]\right]\right]\right]\right]$$
$$\left[mod_a\right]-\left[mod_b\right]$$

Parameters:

fd	is the filename from which the specified modules is read. The INCLUDE command appends the modules to the end of the current OBJECT/32 library. The specified file may be either an OBJECT/32 library or a 126-byte format library.
mod	is the module name(s) or range of modules to include in the OBJECT/32 library. An error is generated if any of the specified modules exist in the OBJECT/32 library.

Functional Details:

When a command is issued that changes the OBJECT/32 library (in this case, the INCLUDE command), a SAVE command must be issued to save all changes.

Examples:

```
INCLUDE LIBRARY.1
```

Include all modules from LIBRARY.1 into the current library.

```
INCLUDE LIBRARY.1, module.1
```

Include module.1 from LIBRARY.1 into the current library.

```
INCLUDE LIBRARY.1, -module.4,module.6,module.10-
```

Include module.1 through module.4, then module.6, and module.10 through the end of LIBRARY.1.

7.15 LIST COMMAND

The LIST command opens a file or device for OBJECT/32 output.

Format:

LIST [=] { fd / CON: }

Parameters:

fd	is the file or device specified for OBJECT/32 output. An error is generated if fd cannot be assigned. If this parameter is omitted, the default list device is CON: (the console device).

7.16 LOG COMMAND

The LOG command opens a file or device for logging a copy of all OBJECT/32 input commands.

Format:

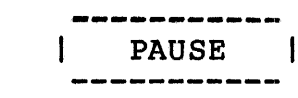
LOG [=] fd

Parameters:

fd	is the file or device used for logging a copy of all OBJECT/32 input commands. An error is generated if fd cannot be assigned.

Functional Details:

Error messages and other output go to the LIST file or device, not the LOG file or device.



7.17 PAUSE COMMAND

The PAUSE command pauses the OBJECT/32 Utility.

Format:

PAUSE

Functional Details:

The OBJECT/32 Utility can be resumed by using the OS/32 CONTINUE command.

7.18 REPLACE COMMAND

The REPLACE command reads specified modules into the current OBJECT/32 library.

Format:

$$\underline{\text{REP}}\text{LACE}\ \ fd\left[,\ \left[\text{mod}_1\ \left[,\text{mod}_2\ \left[,\ldots\ \left[,\text{mod}_n\right]\right]\right]\right]\right]$$
$$\left[\text{mod}_a\right]-\left[\text{mod}_b\right]$$

Parameters:

fd	is the library from which modules are read. The specified library may be either in OBJECT/32 format or in 126-byte format.
mod	represents the module or range of modules to be read from the specified library. More than one module may be specified (separated by commas [,]). A range of modules may be specified (separated by a hyphen [-]). If a starting module is not specified, the first module in the library is assumed. If an ending module is not specified, the last module in the library is assumed. If no modules are specified, all modules in fd are read in and replaced in the current OBJECT/32 library.

Functional Details:

If a specified module currently exists in the OBJECT/32 library, the module being read in replaces the module currently in the library. If the specified modules do not exist in the OBJECT/32 library, the module(s) are appended to the end of the library.

7.19 SAVE COMMAND

The SAVE command saves the current OBJECT/32 library along with all changes that are made to the library.

Format:

SAVE { * | fd }

Parameters:

fd	is the library name to be saved. If "*" is entered (e.g., SAVE*), the library is saved under its current name. If a new name is given to the library, the filename must not exist.

Functional Details:

When a SAVE command is issued, all modules in the library are saved under the name "fd" in OBJECT/32 format.

A SAVE command must be issued if changes have been made to a library by the INCLUDE, REPLACE, DELETE, or TITLE commands.

7.20 TITLE COMMAND

The TITLE command assigns a descriptive title to an OBJECT/32 library.

Format:

TITLE title

Parameters:

title	is the descriptive title to be assigned to the OBJECT/32 library. The title may be optionally enclosed in single or double quotation marks and can be a maximum of 80 bytes.

Functional Details:

This title displays as part of the DIRECTORY command. To remove a title from a library, enter the TITLE command and press the space bar once.

7.21 SAMPLE OBJECT/32 SESSION

Listed below are several examples of OBJECT/32 sessions:

The example listed below shows OBJECT/32 being loaded and started in an interactive environment (a COMMAND file was not specified). The user is creating a new OBJECT/32 library (LIBRARY.1) and is including modules (module.1 and module.2) from an existing library (LIBRARY.2). In addition, a range of modules are being replaced from another library (LIBRARY.3). The DIRECTORY command is used (with its defaults) to display the modules just copied into the new OBJECT/32 library. A SAVE command is issued to save the new library and convert it into the OBJECT/32 format.

```
*LOAD OBJECT32
*START
Concurrent Computer Corp OBJECT/32 03-929 Rxx-xx
>ESTABLISH LIBRARY.1
>TITLE LIBRARY NUMBER ONE
>INCLUDE LIBRARY.2,module.1,module.2
>REPLACE LIBRARY.3,module.4-module.6
>DIRECTORY

Concurrent Computer Corp OBJECT/32 Rxx-xx          Page      1
Date Run: Mon Jan 02 13:49:09 1989              Sequential Map

Library Name:  M300:LIBRARY.1/120

Title:   LIBRARY NUMBER ONE

Module Name: module.1     Date Included: Mon Jan 02 13:45:30 1989
Module Name: module.2     Date Included: Mon Jan 02 13:45:30 1989
Module Name: module.4     Date Replaced: Mon Jan 02 13:46.10 1989
Module Name: module.5     Date Replaced: Mon Jan 02 13:46:10 1989
Module Name: module.6     Date Replaced: Mon Jan 02 13:46:10 1989


>SAVE*
>END
*
```

The example below shows additional modules being replaced in LIBRARY.1. OBJECT/32 is already loaded and started.

```
>GET LIBRARY.1
>REPLACE LIBRARY.4,module.1-module.3
>DIRECTORY

Concurrent Computer Corp OBJECT/32 Rxx-xx                Page       1
Date Run: Tue Jan 03 11:15:10 1989                   Sequential Map

Library Name:  M300:LIBRARY.1/120

Title:  LIBRARY NUMBER ONE

Module Name: module.1     Date Replaced:   Tue Jan 03 11:14:30 1989
Module Name: module.2     Date Replaced:   Tue Jan 03 11:14:30 1989
Module Name: module.4     Date Replaced:   Mon Jan 02 13:46.10 1989
Module Name: module.5     Date Replaced:   Mon Jan 02 13:46:10 1989
Module Name: module.6     Date Replaced:   Mon Jan 02 13:46:10 1989
Module Name: module.3     Date Replaced:   Tue Jan 03 11:14:30 1989


>SAVE*
>END
*
```

Because the REPLACE command was issued, module.1 and module.2 originally included in LIBRARY.1 are replaced by module.1 and module.2 from LIBRARY.4 (because they had the same name). Because module.3 did not exist in LIBRARY.1, it was appended to the end of the modules in LIBRARY.1.

APPENDIX A
LINK AND OBJECT/32 COMMAND SUMMARY

LINK COMMANDS

BFILE fd [,{n | 1}]

BUILD fd [,ABORT] [,ERROR]

DCMD [{APUCOMMENT | NAPUCOMMENT}]

END

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

EXTERNAL common block $name_1$ [,...,common block $name_n$]

FFILE fd [,{n | 1}]

HELP [mnemonic | *]

| INCLUDE [fd] [-BLKDATA] [[[,{module$_1$ | *}]] [[-{module$_n$ | *}]] ,...,module$_x$]

LIBRARY fd$_1$ [,...,fd$_n$]

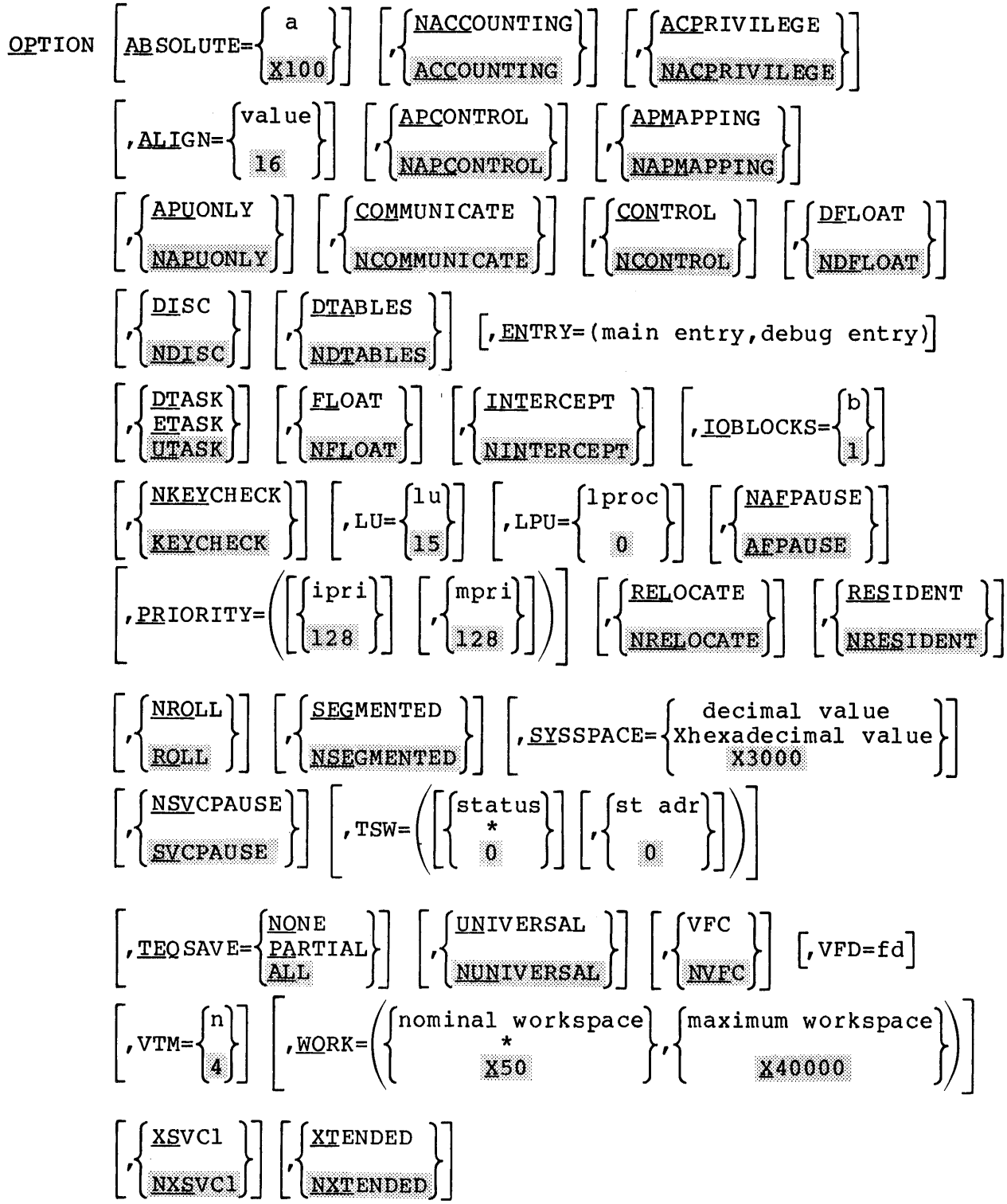
LOCAL entry point$_1$ [,...,entry point$_n$]

LOG fd

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

NDCMD

NLOG


OPTION
[ABSOLUTE={a / X100}]
[,{NACCOUNTING / ACCOUNTING}]
[,{ACPRIVILEGE / NACPRIVILEGE}]
[,ALIGN={value / 16}]
[,{APCONTROL / NAPCONTROL}]
[,{APMAPPING / NAPMAPPING}]
[,{APUONLY / NAPUONLY}]
[,{COMMUNICATE / NCOMMUNICATE}]
[,{CONTROL / NCONTROL}]
[,{DFLOAT / NDFLOAT}]
[,{DISC / NDISC}]
[,{DTABLES / NDTABLES}]
[,ENTRY=(main entry,debug entry)]
[,{DTASK / ETASK / UTASK}]
[,{FLOAT / NFLOAT}]
[,{INTERCEPT / NINTERCEPT}]
[,IOBLOCKS={b / 1}]
[,{NKEYCHECK / KEYCHECK}]
[,LU={lu / 15}]
[,LPU={lproc / 0}]
[,{NAFPAUSE / AFPAUSE}]
[,PRIORITY=([{ipri / 128}] [,{mpri / 128}])]
[,{RELOCATE / NRELOCATE}]
[,{RESIDENT / NRESIDENT}]
[,{NROLL / ROLL}]
[,{SEGMENTED / NSEGMENTED}]
[,SYSSPACE={decimal value / Xhexadecimal value / X3000}]
[,{NSVCPAUSE / SVCPAUSE}]
[,TSW=([{status / * / 0}] [,{st adr / 0}])]
[,TEQSAVE={NONE / PARTIAL / ALL}]
[,{UNIVERSAL / NUNIVERSAL}]
[,{VFC / NVFC}]
[,VFD=fd]
[,VTM={n / 4}]
[,WORK=({nominal workspace / * / X50},{maximum workspace / X40000})]
[,{XSVC1 / NXSVC1}]
[,{XTENDED / NXTENDED}]

OVERLAY overlay name [,{level / 1}]

PAUSE

POSITION COMMON={name / (name$_1$,...,name$_n$) / *}[,TO={nodename / .ROOT}]

RESOLVE [fd][,NAME=package name]
[,ACCESS={R / E / RE / RW / RWE}][,ADDRESS=m0000]
[,STRUCTURE=(name$_1$[/size$_1$] [,...,name$_n$][/size$_n$])]
[,SIZE=([min[,max]])]

REWIND fd

TITLE title

VOLUME voln

WFILE fd [,{n / 1}]

COMMAND[=] { fd / CON: }

DELETE { mod1 [,mod2 [,..., modn] [,[moda]-[modb]] }

DIRECTORY { [[moda] mod1 - [modb]] [,LONG / BRIEF] [,ALPHABETIC / ,XREFERENCE / ,SEQUENTIAL] }

DONE

END

ESTABLISH fd [,title]

EXTRACT fd [, [mod1 [,mod2 [,... [,modn]]] / [moda]-[modb]]]

GET fd

HELP [{ mnemonic / * }]

INCLUDE fd [, [mod1 [,mod2 [,... [,modn]]] / [moda]-[modb]]]

LIST [=] { fd / CON: }

LOG [=] fd

PAUSE

```
REPLACE  fd[, [mod1 [,mod2 [,... [,modn]]]]]
                    [moda] -[modb]

SAVE     {*}
         {fd}

TITLE  title
```

APPENDIX B
ERROR MESSAGE SUMMARY

This appendix provides an alphabetical list and description of Link and OBJECT/32 messages.

There are instances when Link will construct messages as needed based on the following generic form. The following format is used to indicate that a supervisor call 7 (SVC7) error has occurred.

Format 1:

x(ERROR y) ON z TO fd

Where variable x is the type of error, y is the hexadecimal status, z is the SVC7 function attempted, and fd is the file name. See Table B-1 for the error types and status.

Example:

BUFFER ERROR (ERROR 8) ON ASSIGN TO D8:BERDC.OBJ/P

In this example, buffer error is the type of error, the hex status 8 refers to system space and ASSIGN is the function attempted to a file named BERDC.OBJ/P on volume D8.

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.

CANNOT ALLOW SHARABLE CODE - 'ETASK' ALREADY SPECIFIED

An OPTION SEGMENTED or a RESOLVE command is issued after OPTION ETASK was specified.

CANNOT ENABLE E-TASK SUPPORT - SHARABLE CODE HAS ALREADY BEEN SPECIFIED

An OPTION ETASK command is issued after an OPTION SEGMENTED or a RESOLVE command was specified.

CHECKSUM ERROR FILE: x MODULE: y RECORD: z

An invalid checksum is detected while reading an object file.

CHECKSUM ERROR ON INPUT LIBRARY - EXPECTING XXXX, FOUND YYYY

OBJECT/32 discovered an object record whose checksum was calculated as XXXX, but the checksum found was YYYY. The specified object file is probably corrupt. Recompile the program and try again.

COMMAND FORMAT ERROR

While processing a command, OBJECT/32 encountered a syntax error within the command (e.g., a missing command, too many arguments, etc.).

COMMAND NOT PERMITTED

Command is not valid for the type of build or is not permitted as in an embedded command in an object module.

COMMON x ENCOUNTERED IN MORE THAN ONE PARTIAL IMAGE

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

COMMON BLOCK x, UNREFERENCED

The common block named is never referred to.

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

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

CONTINUATION NOT PERMITTED

An attempt is made to continue a command embedded in an object module.

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. In batch mode, Link terminates with an end of task code 3.

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

There is either an extra right parenthesis or a missing left parenthesis.

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.

| GET OR ESTABLISH HAS NOT BEEN ISSUED YET

| In OBJECT/32, the GET or ESTABLISH commands must be issued | first to assign a current object library.

| INPUT LIBRARY IS NOT AN OBJECT32 FORMAT LIBRARY

| A GET command was issued that specified a file which was not | built by OBJECT/32.

| INTERNAL FAILURE n

| An internal error occurred within OBJECT/32 which would cause | hazardous results if execution continued. If OBJECT/32 is | running in interactive mode, it pauses. If OBJECT/32 is | continued or is running in the batch mode, it aborts with an | end of task (EOT) code of 3.

| INSUFFICIENT MEMORY - PLEASE RELOAD WITH A LARGER SEGMENT SIZE

| OBJECT/32 did not have enough work space to build its | internal tables and aborted with an EOT code of 2. OBJECT/32 | should be linked with a minimum of 44K of work space. To | correct this error, reload OBJECT/32 with a minimum of 46K | segment size increment.

INSUFFICIENT WORK SPACE

Link is not loaded with enough workspace. Link terminates with an EOT code of 3, returns to command mode, and clears itself as if an image was built with all options reset to initial load values.

INVALID BLOCK DATA LOADER ITEM ENCOUNTERED: X

While processing a block data segment within an object file, OBJECT/32 found a loader item which is invalid for a block data segment. A second message is displayed indicating the record number, byte offset, and filename in which the error occurred.

The specified object file was probably corrupt. To correct the error, recompile the program in question and try again.

INVALID CHARACTERS IN NAME

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

INVALID COMBINATION OF OPERANDS

A particular combination of operands is invalid.

INVALID COMMAND

While processing a command line, Link or OBJECT/32 encountered an unrecognizable command. To correct this error, ensure each command entered on the last command line is a valid Link or OBJECT/32 command.

INVALID DELIMITER

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

INVALID FILENAME SPECIFIED

The filename specified within an OBJECT/32 command is not a valid OS/32 filename.

INVALID FILE DESCRIPTOR

A syntax error occurred in the specified fd.

INVALID KEYWORD

Misspelled keyword.

INVALID LOADER ITEM ENCOUNTERED: X

While processing an object file, OBJECT/32 found a loader item it did not recognize. The object file (represented by

X) is probably corrupt. A second message is displayed indicating the record number, byte offset, and filename in which the error occurred. To correct this error, recompile the program and try again.

INVALID NUMERIC VALUE

A numeric value is expected but not encountered.

INVALID PARAMETER

An invalid parameter is specified in a command.

INVALID PARAMETER LENGTH

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

INVALID PARAMETER SPECIFIED

While processing a command, OBJECT/32 encountered a parameter that it did not recognize or is outside of the valid range for that parameter.

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 above error message.

INVALID SEQUENCE NUMBER ON INPUT LIBRARY - EXPECTED XXXX,
FOUND YYYY

In an object file, the first two bytes of each record contain a sequence number. This number starts at -1 for the first record, -2 for the second, etc. OBJECT/32 discovered an object record which did not follow this sequence. OBJECT/32 expected to find a sequence number of XXXX, but instead found YYYY. A second message displays indicating the record number, byte offset, and filename in which the error

occurred. To correct this error, recompile the specified program and try again.

INVALID START OPTION

While processing its START options, OBJECT/32 encountered an option it did not recognize. OBJECT/32 aborts with an EOT code of 2.

I/O ERROR ON READ FROM LOGICAL UNIT N:

During OBJECT/32 processing, an error occurred while reading from lu n. Another specific message displays indicating the reason for the read failure.

I/O ERROR ON WRITE TO LOGICAL UNIT N:

During OBJECT/32 processing, an error occurred while writing to lu n. Another more specific message displays indicating the reason for the write failure.

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 is 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 is not specified.

MISSING RIGHT PARENTHESIS

There is a missing right parenthesis.

MODULE INCOMPLETE FILE: x MODULE: y

An end of file condition is detected before the end of program loader item in an object module.

MODULE xxxxxxx ALREADY EXISTS

In OBJECT/32 an INCLUDE command is issued, but the specified module name (xxxxxxx) already exists in the current library. If the specified module is to replace the current module, use the REPLACE command. OBJECT/32 will not allow modules with the same name in an object library.

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 builds the image but no initialization of that common block is performed. After the task image is built, the task common contains 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 specified module name (xxxxxxx) was not found. This error can occur in any command that accepts a module name, more than one module name, or a range of module names as input. To correct this error, ensure that the module name is spelled correctly. In OBJECT/32, ensure that the current object library contains the specified module(s).

MODULE name2 NOT FOUND OR IS LOCATED BEFORE MODULE namel

In OBJECT/32 commands that accept more than one module name or module name ranges as input, the second module name (name2) must be located sequentially after the first module name (namel) in the current object library.

n AMBIGUOUSLY DEFINED SYMBOLS

Entry points are defined in parallel paths and are 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 MULTIPLY DEFINED SYMBOLS

The specified number (n) of entry points are encountered which are 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 is 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.

NO MODULES EXIST IN THE CURRENT LIBRARY

In OBJECT/32, a DIRECTORY command is issued, but the current object library was empty.

NUMERIC VALUE OUT OF RANGE

A numeric operand is 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 is expected but not encountered.

OVERLAY DEFINED OUT OF ORDER

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

PLEASE ISSUE A SAVE COMMAND FIRST

In OBJECT/32 a GET, ESTABLISH, or END command was issued, but the current object library was modified. To correct this error, issue a SAVE command to save the changes made. If a SAVE command is not issued and a GET, ESTABLISH, or END command is issued a second time, all changes made to the library is lost.

PROGRAM TRANSFER ADDRESS IN PROGRAM module IN AN OVERLAY

A program transfer address (PTA) (starting address) is specified for the task in a module that is in an overlay node. Link ignores the specified PTA and uses the task's default starting address.

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.

REQUIRED OPERAND MISSING

OBJECT/32 unexpectedly encountered the end of a command. To correct this error, ensure that all required operands are entered on the command line and that a command does not end with a comma.

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 is 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 are encountered.

UNABLE TO ALLOCATE VOLN:FILENAME.EXT/ACCT#:

During an ESTABLISH or SAVE command, OBJECT/32 is unable to allocate the filename which was specified by the command. A second message displays indicating why the allocate failed. Correct the error as described in the second message and try again. If the action to be taken is unacceptable, specify a different filename.

UNABLE TO ASSIGN VOLN:FILENAME.EXT/ACCT# TO LOGICAL UNIT N:

During any command in which a filename may be specified, OBJECT/32 is unable to assign the filename. The lu to which OBJECT/32 is trying to assign is shown as "N". A second message displays indicating the reason why the assign failed.

UNABLE TO DELETE VOLN:FILENAME.EXT/ACCT#:

During a SAVE command, OBJECT/32 is unable to delete the old library in order to rename the temporary library to the original library name. Two additional messages display. The first message indicates why the delete failed. The second message states "Updated library now in VOLN:OBJECT32.EXT/ACCT#", which indicates the filename where the updated library may be found.

UNABLE TO RENAME VOLN:OBJECT32.EXT/ACCT# TO VOLN:FILENAME.EXT/ACCT#

During a SAVE command, OBJECT/32 is unable to rename the temporary library to the original library name. Two additional messages display. The first message indicates the reason why the rename failed. The second message states "Updated library now in VOLN:OBJECT32.EXT/ACCT#", which indicates the filename where the updated library may be found.

UPDATED LIBRARY NOW IN VOLN:OBJECT32.EXT/ACCT#

During a SAVE command, OBJECT/32 is unable to delete the old library or rename the new library. VOLN:OBJECT32.EXT/ACCT# is the name of the temporary file which is used to save the new library.

VTM TASK WORKSPACE IS GREATER THAN 64K BYTES

When a FORTRAN task is linked as a virtual task, the user task (u-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

The current maximum symbol table size is inadequate. Link should be loaded with a larger workspace. Link terminates with an EOT code of 3.

WARNING: ABSOLUTE SPACE LESS THAN 100

Less than 100 bytes of absolute code were reserved for the User-dedicated Locati n (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 loation counter (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 if lu=255 is entered with the DTABLES option. If the program is to be debugged using DEBUG/32, lu254 cannot be assigned by the program.

WARNING: MORE THAN 16 SEGMENTATION REGISTERS REQUIRED

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

WARNING: n AMBIGUOUS REFERENCES

External references are 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 is 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 is 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 is 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 1MB ADDRESS SPACE

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

WARNING: TASK REQUIRES MORE THAN 12MB ADDRESS SPACE

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

x (ERROR y) ON z TO fd

A SVC7 error occurred. Variable x is the type of error, y is the hexadecimal status, z is the SVC7 function, and fd is the file. See Table B-1 for the error types and status.

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

An SVC1 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-1 SVC7 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 OR BUFFER	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.

On an ALLOCATE, ASSIGN, or DELETE function an error code of 8 is indicative of a BUFFER error. On a DELETE or FETCH ATTRIBUTES function an error code of 8 is indicative of a SYSTEM SPACE error.

TABLE B-2 SVC1 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 off-line).
	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 specifies the code that describes the type of memory error fault that occurred. These codes are defined in Table C-1.

TABLE C-1 VTM MEMORY FAULT CODES

MEMORY FAULT CODES	MEANING
00	Supervisor call (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.

TASK FD ASGN-ERR - CODE=xx

Error in assigning task file. xx is the SVC7 error status.

VIRT FD ALLO-ERR - CODE=xx

Error in allocating temporary file. xx is the SVC7 error status.

VIRT FD ASGN-ERR - CODE=xx

Error in assigning VFD file. xx is the SVC7 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 input/output (I/O) transfer. xxxx is the SVC1 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 SVC1 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 taking -1 and performing an EXCLUSIVE-OR operation on each halfword of data in the record (except the checksum halfword itself).

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.

Table D-1 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.

TABLE D-1 OBJECT CODE LOADER ITEMS

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.

TABLE D-1 OBJECT CODE LOADER ITEMS (Continued)

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 date	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.

TABLE D-1 OBJECT CODE LOADER ITEMS (Continued)

LOADER ITEM	DATA FORMAT	DESCRIPTION
25	12 bytes absolute data	Absolute data.
26	14 bytes absolute data	Absolute data.
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.

TABLE D-1 OBJECT CODE LOADER ITEMS (Continued)

LOADER ITEM	DATA FORMAT	DESCRIPTION
3E	62 bytes absolute data	Absolute data.
3F	64 bytes absolute data	Absolute data.
40	66 bytes absolute data	Absolute data.
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.

TABLE D-1 OBJECT CODE LOADER ITEMS (Continued)

LOADER ITEM	DATA FORMAT	DESCRIPTION
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.
5B	120 bytes absolute data	Absolute data.
5C	1 byte location counter (LOC) number 8 bytes section name 8 bytes data pool name	Define PURE LOC.
5D	Reserved for future use	Reserved.
5E	Reserved for future use	Reserved.
5F	1 byte LOC number 3 bytes address data	Load program address.
60	1 byte LOC number 2 bytes address data	Defined counter.
61	1 byte LOC number 4 bytes address data	Defined counter relocatable address.
62	1 byte LOC number 2 bytes absolute data 2 bytes address data	Instruction with address based on a defined LOC.
63	1 byte LOC number 3 bytes absolute data 3 bytes address data	Instruction with an address based on a defined LOC.
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.

TABLE D-1 OBJECT CODE LOADER ITEMS (Continued)

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	Reserved for future use	Reserved.
69	1 byte character count, 0-80 bytes of title	Library title.
6A	8 bytes name 4 bytes date 3 bytes absolute length 3 bytes impure length 3 bytes pure length 2 bytes sequence number 4 bytes displacement 2 bytes sequence number 4 bytes displacement 2 bytes sequence number 4 bytes displacement 4 bytes relocation entry	Start of module table: Module name Date ("C" language format) module included in library Length of Absolute segment Length of Impure segment Length of Pure segment Sequence number of extrn table. Byte offset in library of extrn table. Sequence number of start of data. Byte offset in library to start data. Sequence number of next module. Byte offset in library of next module. Number of relocatable data items in module.
6B	(none)	End of module table.
6C	8 bytes name 3 bytes displacement 3 bytes length	New-format block data item.
6D	1 byte LOC number 8 bytes section name 8 bytes data pool name 3 bytes length	New-format PURE LOC.
6E-FF	Reserved for future use	Reserved.

INDEX

W,X,Y,Z

Document Comment Form

In reference to...

OS/32 LINK Reference Manual — 48-005 F00 R05

We try to make our documentation easy to use, easy to understand, and free from errors. We invite your comments and suggestions to assist us in improving our documentation to suit your needs.

Please send us comments, corrections, suggestions, etc. Use the SCR system to report software documentation or software problems.

I think this manual...

	Strongly Agree	Agree	Disagree	Strongly Disagree
is easy to read	☐	☐	☐	☐
is easily understood	☐	☐	☐	☐
is concise & to the point	☐	☐	☐	☐
covers the subject	☐	☐	☐	☐
has enough detail	☐	☐	☐	☐
is well organized	☐	☐	☐	☐
provides easy-to-locate information	☐	☐	☐	☐
is aesthetically pleasing	☐	☐	☐	☐
has clear illustrations	☐	☐	☐	☐
has enough illustrations	☐	☐	☐	☐
has meaningful examples	☐	☐	☐	☐
has a helpful index	☐	☐	☐	☐

My other comments...

Please make any additional specific comments. (Include chapter, page, table or figure number.)

_____

_____

_____

_____

About myself...

Job Function: ☐ Dev. Engineer ☐ Sys. Analyst ☐ Sys./App. Prog.
☐ Technician ☐ Administrator ☐ Casual user
☐ Service Eng. ☐ Operator ☐ Other _____

What hardware system are you using? _____

_____

What revision level of system software are you using? _____

_____

Name/Title: _____

Company/Organization: _____

Address: _____

May we contact you? ☐ Yes ☐ No

Telephone: _____ Date: _____

FOLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

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

POSTAGE WILL BE PAID BY ADDRESSEE

Concurrent Computer Corporation
2 Crescent Place
Oceanport, NJ 07757

ATTN:
TECHNICAL SYSTEMS PUBLICATIONS DEPT.

FOLD FOLD

STAPLE STAPLE

9410

Document Comment Form

In reference to...

_____ _____

Manual Title *Number & Revision Level*

We try to make our documentation easy to use, easy to understand, and free from errors. We invite your comments and suggestions to assist us in improving our documentation to suit your needs.

Please send us comments, corrections, suggestions, etc. Use the SCR system to report software documentation or software problems.

I think this manual...

	Strongly Agree	Agree	Disagree	Strongly Disagree
is easy to read	☐	☐	☐	☐
is easily understood	☐	☐	☐	☐
is concise & to the point	☐	☐	☐	☐
covers the subject	☐	☐	☐	☐
has enough detail	☐	☐	☐	☐
is well organized	☐	☐	☐	☐
provides easy-to-locate information	☐	☐	☐	☐
is aesthetically pleasing	☐	☐	☐	☐
has clear illustrations	☐	☐	☐	☐
has enough illustrations	☐	☐	☐	☐
has meaningful examples	☐	☐	☐	☐
has a helpful index	☐	☐	☐	☐

My other comments...

Please make any additional specific comments. (Include chapter, page, table or figure number.)

_____

_____

_____

_____

About myself...

Job Function: ☐ Dev. Engineer ☐ Sys. Analyst ☐ Sys./App. Prog.
☐ Technician ☐ Administrator ☐ Casual user
☐ Service Eng. ☐ Operator ☐ Other _____

What hardware system are you using? _____

_____

What revision level of system software are you using? _____

_____

Name/Title: _____

Company/Organization: _____

Address: _____

May we contact you? ☐ Yes ☐ No

Telephone: _____ Date: _____

FOLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

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

POSTAGE WILL BE PAID BY ADDRESSEE

Concurrent Computer Corporation
2 Crescent Place
Oceanport, NJ 07757

ATTN:
TECHNICAL SYSTEMS PUBLICATIONS DEPT.

FOLD FOLD

STAPLE STAPLE

9410