Publication Number S29-430R06

OS/16 MT2

OPERATOR'S MANUAL

PERKIN-ELMER

Computer Systems Division
2 Crescent Place
Oceanport, N.J. 07757

PAGE REVISION STATUS SHEET

PUBLICATION NUMBER S29-430

TITLE OS/16 MT2 Operator's Manual

REVISION R06 DATE September 1979

PAGE	REV.	DATE	PAGE	REV.	DATE	PAGE	REV.	DATE
i/ii	R06	9/79	3-20 thru			7-4	R05	2/79
iii thru iv	R05	2/79	3-21 thru 3-23	R05	2/79	7-5	R02	8/76
v	R06	9/79	3-24 thru 3-35/			7-6 thru 8-1 thru 8-7/	R05	2/79
vi thru ix/x	R05	2/79	3-36	R05	2/79	8-8	R06	9/79
1-1/ 1-2	R04	1/78	4-1	R05	2/79	9-1 thru 9-4	R05	2/79
2-1 thru 2-2	R06	9/79	4-2 thru 5-1	R06	9/79	10-1 thru 10-3	R05	2/79
2-3 thru 2-4	R05	2/79	5-2	R02	8/76	10-4	R02	8/76
2-5	R06	9/79	5-3	R05	2/79	10-5	R05	2/79
2-6	R05	2/79	5-4	R04	1/78	10-6	R06	9/79
3-1 thru 3-2	R04	1/78	5-5	R05	2/79	10-7	R04	1/78
3-3	R02	8/76	5-6	R06	9/79	10-8 thru 10-9/		
3-4	R04	1/78	5-7 thru 5-8	R03	6/77	10-10	R05	2/79
3-5	R05	2/79	5-9	R02	8/76	11-1 thru 11-2	R06	9/79
3-6	R02	8/76	5-10	R04	1/78	11-3 thru 11-4	R04	1/78
3-7	R04	1/78	5-11	R03	6/77	11-5 thru 11-6	R06	9/79
3-8	R05	2/79	5-12	R05	2/79	11-7 thru 11-8	R04	1/78
3-9	R04	1/78	5-13	R03	6/77	12-1	R05	2/79
3-10	R05	2/79	5-14 thru 5-15	R04	1/78	12-2	R04	1/78
3-11	R06	9/79	5-16 thru 5-19/			12-3/ 12-4	R02	8/76
3-12	R05	2/79	5-20	R02	8/76	13-1 thru 13-4	R02	8/76
3-13 thru 3-14	R04	1/78	6-1 thru 6-2	R04	1/78			
3-16	R05	2/79	6-3	R05	2/79			
3-17	R02	8/76	6-4	R04	1/78			
3-18 thru 3-19	R03	6/77	7-1	R05	2/79			
			7-2	R02	8/76			
			7-3	R04	1/78			

PAGE REVISION STATUS SHEET

PUBLICATION NUMBER S29-430

TITLE OS/16 MT2 Operator's Manual

REVISION R06 DATE September 1979

PAGE	REV.	DATE	PAGE	REV.	DATE	PAGE	REV.	DATE
13-5	R05	2/79	A8-1	R05	2/79			
13-6 thru 13-12	R02	8/76	A8-2	R02	8/76			
			A9-1	R05	2/79			
14-1 thru 14-8	R03	6/77	A9-2	R02	8/76			
14-9	R04	1/78	A10-1/ A10-2	R05	2/79			
14-10 thru 14-12	R03	6/77	A11-1/ A11-2	R05	2/79			
14-13	R05	2/79	A12-1/ A12-2	R02	8/76			
14-14	R03	6/77						
14-15	R05	2/79	A13-1/ A13-2	R05	2/79			
14-16	R03	6/77						
14-17 thru 14-18	R05	2/79	A14-1/ A14-2	R03	6/77			
A1-1 thru A1-2	R06	2/79	I-1 thru I-4	R06	9/79			
A1-3	R05	2/79	I-5/ I-6	R05	2/79			
A1-4	R02	8/76						
A2-1	R02	8/76						
A2-2	R04	1/78						
A2-3 thru A2-4	R05	2/79						
A3-1/ A3-2	R06	9/79						
A4-1/ A4-2	R06	9/79						
A5-1/ A5-2	R04	1/78						
A6-1 thru A6-4	R02	8/76						
A7-1/ A7-2	R05	2/79						

PREFACE

This manual describes the procedures for operating an INTERDATA computer with the Operating System OS/16 MT2. The information contained herein is for the operator's reference. Programming interface information for the OS/16 MT2 is contained in:

OS/16 MT2 Programmer's Reference Manual, Publication Number 29-429.

Other manuals related to OS/16 MT2 are:

OS/16 MT2 System Planning and Configuration Guide, Publication Number 29-431
OS/16 MT2 Pocket Guide, Publication Number 29-433
OS/16 MT2 Program Logic Manual, Publication Number 29-434
16-Bit Series Reference Manual, Publication Number 29-398
OS/16 Mini I/O System User's Manual, Publication Number 29-491.

MANUAL ORGANIZATION

This manual contains detailed reference material needed to control an OS/16 MT2 system through the system console. In addition, a complete description of the OS/16 Task Establisher is given, as well as a survey of the utility programs supplied with an OS/16 MT2 system.

Chapter 2 discusses the procedures for loading either the INTERDATA Supplied Starter systems, or a user generated system. This chapter should be understood before attempting to load an OS/16 MT2 system from Disc or Tape.

Chapter 3 discusses procedures for controlling an OS/16 MT2 system through the OS/16 MT2 Command Processor. System console operations and the full OS/16 MT2 Operator Command Language are defined. Chapter 3 assumes that the reader is familiar with the concepts discussed in Chapter 1 of the *OS/16 MT2 Programmer's Reference Manual,* Publication Number 29-429. Of special interest are the sections describing memory management, task management, and roll out.

Chapter 4 discusses OS/16 MT2 error handling as it pertains to the console operator.

Chapter 5 describes the facilities of the OS/16 Task Establisher (TET/16). This chapter provides a detailed description of TET/16 commands, and examples of its use.

Chapter 6 discusses the operation of the Common Assembler CAL/16 (memory-based) and CAL/16D (disc-based).

Chapter 7 describes the Disc Integrity Check utility.

Chapter 8 describes the Backup utility.

Chapter 9 describes the Output Spooler utility.

Chapter 10 surveys the utility programs and Language Processors available for execution under OS/16 MT2. For a more detailed discussion, refer to the list of documentation provided in the *OS/16 MT2 System Planning and Configuration Guide*, Publication Number 29-431.

Chapter 11 discusses the procedures for unpackaging and backing up an OS/16 MT2 System on either Magnetic Tape or Disc. Those procedures may also serve as examples for the creation and maintenance of user libraries.

Chapter 12 is a guide to the File Management facilities available to the console operator. It assumes the reader is familiar with the commands described in Chapter 3.

Chapter 13 provides a tutorial discussion of the facilities available through the Command Substitution System capability of the Command Processor.

Chapter 14 describes the use of the High Level Operator Command Package.

Appendix information to be added.

Table of Contents

APPENDICES

FIGURES

TABLES

CHAPTER 1
INTRODUCTION

SYSTEM DESCRIPTION

OS/16 MT2 is an Operating System that provides task management for multiple-task environments for the 16-Bit series of INTERDATA Processors. Both background and foreground facilities are provided so that program preparation can proceed concurrently with real-time system operation. Built-in functions of OS/16 MT2 include system control via the operator's console, interrupt handling and I/O servicing. Data file management features are provided for any system equipped with direct-access storage media.

OS/16 MT2 is a compatible subset, from the standpoint of the using programs, of OS/32 MT.

OS/16 MT2 protects the foreground environment from the effects of undebugged background tasks. Memory can be protected via the Memory Protect Controller. Both static and dynamic protection mechanisms are provided for I/O devices and direct-access files.

This manual is intended as an operator's reference manual for OS/16 MT2. For information on the programming interface of OS/16 MT2 refer to:

OS/16 MT2 Programmer's Reference Manual, Publication Number 29-429.

Other manuals related to OS/16 MT2 are:

OS/16 MT2 System Planning and Configuration Guide, Publication Number 29-431
OS/16 MT2 Pocket Guide, Publication Number 29-433
OS/16 MT2 Program Logic Manual, Publication Number 29-434
16-Bit Series Reference Manual, Publication Number 29-398
OS/16 Mini I/O System User's Manual, Publication Number 29-491.

NOTATION

This section defines the notation used to describe the command languages discussed in this manual.

1. Items that are shown in upper-case letters must be specified exactly as shown. If one or more letters are underlined, this indicates a minimum abbreviation and the item may be specified by any number of characters from the minimum to the full item.

2. Items that are shown in lower-case letters must be replaced by a value defined in the description of the command.

3. Optional items are enclosed in square brackets:

   [optional item]

4. Alternate items are listed vertically, enclosed in braces:

   { choice a
   { choice b

5. An ellipses (...) is used to denote an optional repetition of the previous item.

6. Numeric values are given in both decimal and hexadecimal, where appropriate as:

   Decimal (Hexadecimal)
   e.g. 1(1), 10(A), 32(20)

CHAPTER 2
LOADING THE OS/16

INTRODUCTION

As described in the OS/16 MT2 Packaging Document, a ready-to-load OS/16 MT2 system comes in six forms:

a non-disc system supplied on magnetic tape, cassette, and disc (STARTER 1),
a fully overlayed disc system supplied on disc only (STARTER 2, DEVELOPMENT SYSTEMS 1, 2, and 3)
a non-overlayed disc system supplied on magnetic tape, cassette and disc (STARTER 3)

The operating instructions for the STARTER systems are given below. All data to be entered is given in hexadecimal.

NON-DISC SYSTEMS

Loading OS/16 From A Non-Disc Device

The 16-Bit Relocating Loader (REL Loader) is used to load from a non-disc device. The OS/16 MT2 Functional Program Package contains the REL Loader as the first program on Magnetic Tape or Cassette, followed by STARTER1 and STARTER3.

To load the REL Loader, the 50 Sequence is used:

1. Mount the Magnetic Tape, Cassette or Paper Tape containing the REL Loader. Position at load point.

2. Enter the 50 Sequence into low memory.

   Location 50 should be D500
   52 should be 00CF
   54 should be 4300
   56 should be 0080

   Set location '78' to the device number and Output Command of the load device.

Magnetic Tape	85A1
Cassette	45A1
Paper Tape Reader	1399
Teletype Reader	0294

   On a Series Sixteen, set the number of file marks to be skipped into the halfword at location X'7E'.

3. Enter the following Illegal Instruction New PSW.

   Location 34 should be 0
   36 should be 50
   22 should be 80

   NOTE
   On systems with extended memory insure that bits 8-11 of the PSW status field are zero.

4. Initialize the Processor and execute starting at location 34. The Illegal Instruction new PSW causes the 50 sequence to be entered with the proper PSW.

5. The tape should now move as the REL Loader is loaded. When the Wait light comes ON and the tape stops, loading is complete. If the REL Loader is on a different media than the OS to be loaded, mount the OS Magnetic Tape, Cassette or Paper Tape, and set location 78 to the correct value as in (2).

6. Address the REL Loader. Start address:

FB00 for a 64KB or greater Processor
DB00 for a 56KB Processor
BB00 for a 48KB Processor
9B00 for a 40KB Processor
7B00 for a 32KB Processor
5B00 for a 24KB Processor
3B00 for a 16KB Processor
1B00 for a 8KB Processor

When starting the REL loader, the operator must set the proper number of file marks to be skipped before loading, if the OS is being loaded from Magnetic Tape or Cassette.

The data register on the display panel must be set to 0 to skip no file marks, to 100 to skip 1 file mark (position of STARTER1), or to 200 to skip 2 file marks (position of STARTER3).

On the Model 70, 74, 80 or 85 display panel, set the data switches to the correct value before depressing EXECUTE to start the REL loader.

On the Series Sixteen, depress "<" to load the OS after the REL loader enters the wait state.

On all other display panels, enter:

DTA – start address of the REL loader
ADD
DTA – file mark skip value
RUN

7. The tape should now move as the OS is loaded. When loading is complete, the tape stops and the OS ID is output on the system console:

OS/16 MT2 01-00 where 01-00 is the revision and update level.

The OS is now ready to accept commands.

Loading OS/16 MT2 Configured With No Command Processor

Follow steps (1) through (6) above.

The tape should now move as the OS is loaded. When loading is complete, the tape stops and the OS begins scheduling tasks. This system has no system console.

Errors In Loading From a Non-Direct Access Device

If the REL Loader does not load properly (Wait light fails to come ON or display panel is non-zero after tape stops) check the low memory locations:

22
34-37
50-57
78

to verify that the 50 sequence is entered correctly. If so, then a hardware error may exist.

Check REL Loader device to ensure that it is turned ON and is On-Line to the system.

Ensure REL Loader tape is positioned correctly.

Try a different copy of the REL Loader.

If the OSID is not output to the system console after OS tape stops, check display panel for REL Loader error code or OS/16 crash code. See *16-Bit Loader Descriptions Manual,* Publication Number 29-231 for REL Loader codes.

A system crash code of X'66666666' indicates system console unavailable. Check to verify that the system console is turned on and is On-Line to the system.

Loading OS/16 MT2 with the Boot Loader

1. Mount the device containing the OS/16 MT2 Boot Loader at load point, and initialize the Processor.

2. Mount the Disc Volume and wait until the ready light comes ON. Do not disable the hardware Write Protect at this time.

3. Enter the 50 Sequence and Boot Loader control information into low memory.

   Set location 22 to 80
   34 to 0 (Set Illegal Instruction New PSW)
   36 to 0050
   50 to D500 (Set 50 Sequence)
   52 to 00CF
   54 to 4300
   56 to 0080

   NOTE

   On extended memory processors insure that bits 8-11 of the PSW status field are zero.

   78 to the Boot Loader input device/cmd
   - 85A1 Magnetic Tape
   - 45A1 for Cassette
   - 1399 for Paper Tape Reader
   - 0294 for TTY Reader
   - C186 for Floppy Disc drive 0 (C196 for drive 1, etc.)

   7A to the device number and code of the disc containing the OS
   - C631 for 2.5MB Disc
   - C633 for 10MB Disc (removable platter)
   - C732 for 10MB Disc (fixed platter)
   - FC34 for 40MB Disc
   - FC35 for 67MB Disc
   - FC36 for 256MB Disc
   - C137 for Floppy Disc

   7C to the Disc controller and SELCH address
   - B6F0 for 2.5 or 10MB Disc
   - FBF0 for 40, 67 or 256MB Disc
   - 0000 for Floppy Disc drive 0 (0001 for drive 1, etc.)

   7E to the identification number of the OS to be loaded. This number is specified by the user as the file extension when the OS is established by TET/16. For STARTER1 the value is 0001. For STARTER2, the value is 0002.

   Execute the Processor at Location 34. The illegal instruction New PSW causes the 50 Sequence to be entered with the proper PSW.

4. The Boot Loader loads the OS image which outputs the system ID to the system console:

   ```
   OS/16 MT2   02-00
   *fd:        voln        PROT
   ```

   Where fd and voln are the device mnemonic and volume name, respectively, of the OS volume.

   The OS is now ready to accept commands.

   After initializing itself, the system marks the volume from which it was loaded, On-Line as the OS volume, with Write protection. If this volume is to be written on, it should be marked on unprotected, by the operator.

Loading OS/16 MT2 Configured With No Command Processor

Follow steps (1) through (3).

The Boot Loader loads the OS image into memory. After initializing itself, the system marks on-line with write protection every disc configured in the system which is hardware protected, and marks on-line without write protection all discs which are not hardware protected. The system then begins scheduling tasks. There is no system console.

Loading OS/16 MT2 With a 7/16 LSU Or ALO

Refer to the *OS/16 MT2 LSU Direct Access Loader Instruction Manual,* Publication Number 29-534 or the *OS/16 MT2 ALO Direct Access Loader Instruction Manual,* Publication Number 29-533 for information on loading OS/16 with the Loader Storage Unit (LSU) or the Automatic Load Option (ALO), respectively.

Loading OS/16 MT2 With the 5/16 LSU (M51-102)

When bootstrapping from a floppy disc on the 5/16, the OS/16 MT2 Boot Loader must be resident on sector 2 of the floppy disc. Ready the appropriate drive and depress initialize. The OS is booted from the first ready drive starting at drive 0. The OS extension number must be 0000 to be bootstrapped on the 5/16. The floppy controller address must be C1

Loading OS/16 MT2 With a Series Sixteen ALO

The Series Sixteen ALO accepts and displays boot load device information on the ASCII console. The user may load from a number of predetermined device configurations. The following table lists these devices in the order of load priority:

DEVICE	DEVICE NAME	DEVICE ADDR	CONTROLLER/ DRIVE ADDR	SELCH ADDR
10Mb Disc	DSC2	C7	B6	F0
10Mb Disc	DSC1	C6	B6	F0
Floppy Disc	FLP1	C1	0	
67Mb Disc	DSC5	FC	FB	F1
800 BPI Mag Tape	MAG1	85		
1600 BPI Mag Tape	MAG2	C5		
10Mb Disc	DSC4	C7	36	F1
10Mb Disc	DSC3	C6	36	F1
Floppy Disc	FLP2	C1	1	
Floppy Disc	FLP3	C1	2	
Floppy Disc	FLP4	C1	3	
67Mb Disc	DSC6	FD	FB	F1
67Mb Disc	DSC7	EC	EB	F2
67Mb Disc	DSC8	ED	EB	F2
800 BPI Mag Tape	MAG3	95		
1600 BPI Mag Tape	MAG4	D5		

When the Initialize switch is depressed, the messages:

```
SERIES SIXTEEN CPU nKB
LOAD dddd.002?
```

are output on the ASCII console. n is the number of kilobytes of memory on the system and dddd is the highest priority ready device that contains an OS with an OSid equal to .002.

Example:

```
SERIES SIXTEEN CPU 64KB
LOAD DSC1.002?
```

For magnetic tape devices ".002" is not output. A magnetic tape must contain the 16-bit Rel Loader followed by an OS in 16-bit object format.

If the user types "Y" followed by carriage return, the loader attempts to load from the above device. If any other character is entered or there was no ready device containing an OS with OSid equal to .002, the loader lists all devices on the system, and asks the user to enter a device name and OSid. "OK" printed next to the device name indicates that the device is ready to load.

Example:

```
LOAD DEVICES
DSC2      OK
DSC1      OK
FLP 1     OK
FLP2
FLP3
FLP4
MAG1      OK
MAG3
ENTER DEVN.OSID
```

The user should now enter a device name followed by a period, a three character OSid, and a carriage return.

Example:

DSC1.004

The Loader now loads this OS from the specified device and the message *OS/16 MT2 XX-YY appears on the console. For magnetic tape devices, the processor enters the "wait" state after the Rel Loader is loaded. Enter "<" from the ASCII console. If a file mark is between the Rel Loader and the OS to be loaded, enter a second "<".

The following error messages may appear:

?	Invalid device name or OSid. Reenter.
NO SUCH OS	No OS found with the OSid specified. Reenter.
IOER ddss REBOOT	Device input error where dd is the device number and ss is the hardware device status. The process must be restarted.

To load an OS from a device whose device address, device code, controller address or selch address does not appear in the table of devices, the following modification can be made to the ALO program. After the "ENTER DEVN.OSID" prompt has been output to the console, depress the HALT/EXE button and enter

```
@28EC
=xxyy      (xx=device code, yy=device number)
=wwzz      (ww=controller address, zz=selch address ) (0 for mag tape devices)
@2762
<
```

The list of available devices will appear. Enter DSC2.aaa where aaa is the OS id of the OS to be loaded.

Bootstrapping from a Floppy Disc Using the 50 Sequence

When bootstrapping OS/16 from a floppy disc using the 50 sequence, the OS/16 MT2 Boot Loader must be resident on sector 2 of the floppy disc. The OS package, when shipped on floppy disc, comes with the Boot Loader in image form on sector 2 of floppy number one (FL01). If the user desires to put the Boot Loader on additional floppys for his own use, copy the entire physical floppy with OSCOPY.

The user should verify that the image file starts on sector 2 by use of the FILES command (see Chapter 3).

Error In Loading From Disc

If the Boot Loader does not load properly, (OS ID does not appear on system console within 50 seconds after Boot Loader tape stops or display panel does not display Zeros), check the low memory locations:

34-37
50-57
78-7F

to verify that the 50 Sequence is entered correctly. Display the PSW status. If the 50 Sequence is correct and status is not Zero, a processor hardware error may exist.

Check the Disc to verify that the Ready Light is ON.

Check the system console to see if it is turned ON and is On-Line to the system (display panel value of 66666666 indicates system console hardware error).

Check the Boot Loader device to verify that it is turned ON and is On-Line to the system. If the tape does not move, there may be a Processor or Boot Loader device hardware error. If tape was completely read in on TTY Reader, turn TTY OFF and ON to generate error status. OS should then boot in.

If a system crash code is displayed, check the code for indications of the problem.

If code 0001 is displayed, the boot loader was unable to find an OS file with the extension given in location X'7E'.

Try a different copy of the Boot Loader.

Check the disc address, disc device code, controller address and SELCH address (X'7A' X'7D') to verify they are correct for the specific hardware configuration.

Check Console Device configured in system (Teletype or PASLA Interface) to verify that it is correct for the specific hardware configuration.

On systems with extended memory insure that bits 8-11 of the PSW status are zero before attempting to set memory locations.

Tailoring Starter

The Starter systems are configured with standard device numbers. If the particular configuration contains devices on non-standard device numbers, follow the procedure in the OS/16 MT2 System Planning and Configuration Guide, Appendix 2.

NOTE

The Starter systems are configured for console CRT support. If the system console device is a teletype consult the appropriate Starter map supplied with the package. Find the address of symbol DBCONA. Modify location DBCONA - X'12' to contain X'1002', where 2 is the TTY device address, then start the system at X'60'.

Restarting the Operating System

At times it may be convenient to restart the operating system without reloading. This may be accomplished by starting the system at location X'60'. This may not be done if the CUP statement DELETE INITIAL is specified.

Preparation for Loading Tasks

Refer to Chapter 3 for a description of the operator commands, and Chapter 10 for examples of running utility programs.

A system with no foreground partitions and no disc management is initialized with all available memory in the background partition (e.g. STARTER 1). The TASK command is not recognized since there can only be one task in the system. Any object code format utility or user task can be loaded into the system directly, using the resident background loader (LDBG).

Any system with disc management, and/or at least one foreground partition, is initialized with all available memory in the dynamic system space area. Before any task can be loaded, a suitable partition must be set up for it, using the SET PARTITION command.

On a system with at least one foreground partition, the TASK command must be entered before any action can be taken on behalf of a loaded task.

On a system with disc management, the default system volume name may or may not have been specified when the system was configured. If it was not (this applies to the Starter systems), then it must be specified by the VOLUME command.

On a system with ROLL support, a disc volume on the system must be marked on-line for ROLL if any tasks are to be rolled out. Note that on this type of system, it is not essential to mark on a ROLL volume if you do not intend to make use of the ROLL feature. However, a warning message will be given when partitions are set up using SET PART.

To be able to write to the OS volume (which is automatically marked on-line with protect), it must first be marked on-line without protect.

For example, a common sequence of commands entered immediately after a 2-partition, disc-based system is loaded, is as follows:

```
MA    DSC1:,ON        mark OS volume on-line without protect
MA    DSC2:,ON,R      mark second volume on-line as ROLL volume
VOL   MT16            set default volume name
SET   PART 1/F000     set background partition
LD    TET             load TET/16 into background
TA    .BG             currently selected task is background task
AS    3,PR:           make logical unit assignments
AS    5,CON:
AS    7,CON:
ST                    Start TET/16
```

CHAPTER 3
CONSOLE OPERATIONS AND OPERATOR COMMANDS

SYSTEM CONSOLE DEVICE

This chapter deals exclusively with systems that are configured with the Command Processor module.

The OS/16 MT2 system is controlled by the console operator through a device called the system console. The devices supported as system console are described in Chapter 8 of the *OS/16 MT2 Programmer's Reference Manual.* This device has a special relationship to the system; the Command Processor Task reads command input from this device and writes system messages to it. Tasks may log messages to the system console without reference to its device name.

The system console may be assigned to tasks for ordinary I/O purposes, just as any other device; however, all I/O requests to this device are intercepted by the Command Processor, which performs them on behalf of the calling task. If the system console is an ASR TTY, the punch unit is not supported. The Reader unit may be accessed in the same manner as any other device on the system.

Prompts

When the console operator is expected to enter data at the system console, a prompt is output to serve as a reminder. This prompt takes one of the following forms:

```
*          (command request)
TASKID>    (data request)
```

The command request prompt (*) is output whenever the system is ready to accept another command.

The data-request prompt (TASKID>) is output whenever a task is attempting to perform a Read I/O request from the system console. The TASKID field of this prompt is the name of the task requesting data. In the case of the background task, the TASKID is ".BG".

The console operator should satisfy the data request as soon as practical, since system messages are queued until the data request is satisfied.

BREAK Key

If a task is in the process of reading from, or writing to the system console, the operator can interrupt this I/O in order to enter a command by depressing the BREAK Key (on some devices, the ESCAPE Key) of the console device. This forces the system into command mode for the entry of one command line. After one command line has been accepted, the user I/O to the console is restarted. This process is transparent to the user task which requested the I/O.

The BREAK Key may also be used by the operator to discard further system responses to a command. This is particularly useful in cases such as the EXAMINE and DISPLAY commands, where large quantities of data may be output at the system console.

Input Editing Functions

When entering commands or data at the system console, the operator may make corrections to his input line. A back arrow character (←) or BACKSPACE (on some devices, CONTROL-H) causes the previous character to be ignored. A hash mark character (#) causes the entire input line to be ignored and a carriage return-line feed sequence to be output. The operator should then enter a new line. Note that no new prompt character is output.

COMMAND SYNTAX

Commands are accepted one line at a time. A command may not be spread over two or more lines. Multiple commands may appear on the same line, separated by semicolons (;). A command line is terminated by a carriage return.

Commands are composed of:

- Mnemonics
- Decimal numbers
- Hexadecimal numbers
- Task identifiers
- File Descriptors

Mnemonics

Mnemonics are shown in this manual in upper-case letters. Mnemonics may be abbreviated with any number of characters from a minimum abbreviation to the full mnemonic. Minimum abbreviations are selected so as to resolve ambiguities between mnemonics while remaining as short as possible. Minimum abbreviations are underlined in this manual, as follows:

REWIND

For example, given the above command:

REW	
REWI	These are all legal forms of the
REWIN	command shown above.
REWIND	
RE	Illegal, too short.
REWA	Illegal, misspelled.
REWINDZ	Illegal, too long.

Decimal and Hexadecimal Numbers

The OS/16 MT2 command structure uses decimal rather than hexadecimal operands for almost every purpose. The only major exception to this rule is in the case of addresses, which are always expressed in hexadecimal.

Leading zeros may always be omitted in numerical operands, whether decimal or hexadecimal.

Hexadecimal addresses are normally four characters in length. Addresses may be five characters for extended memory systems. Valid addresses are in the range 0-FFFF, 18000-1FFFF, 28000-2FFFF, 38000-3FFFF, 48000-4FFFF, 58000-5FFFF, and 68000-6FFFF. See Chapter 6 of the *OS/16 MT2 System Planning and Configuration Guide,* Publication Number 29-431, for details on extended memory systems.

Task Identifiers

Task identifiers must consist of from one to eight characters; the first character must be alphabetic and the rest must be alphameric. Thus:

TASK3	
FRED	These are all valid identifiers.
X	
T997XY25	
34TASK	Invalid, first character not alphabetic.
T43.2	Invalid, non-alphameric character.
TASK12345	Invalid, more than eight characters.

File Descriptors

File Descriptors (generally abbreviated fd in this manual) are composed of three fields:

voln:filename.ext

Voln is the name of the volume on which the file resides or the device mnemonic of a device. It may be from one to four characters, the first character being alphabetic and the rest alphameric. Filename is the name of the file. It may be from one to eight characters, the first character being alphabetic and the rest alphameric. Ext is the filename extension field. It may be from zero to three characters, which must be alphameric.

Voln need not be specified, the default being the system volume. If voln is not entered, the colon (:) separating voln and filename must not be entered.

Ext need not be specified; the default is generally the blank extension, but some commands may make use of a different default value. If ext is to be defaulted, the period (.) separating filename and ext should not be entered.

File Descriptors may refer to devices or direct access volumes as well as to direct-access files. In the case of a device, the voln field is the four-character device mnemonic or volume ID, and filename and ext should not be entered. The colon following voln must always be entered in this case.

Examples of legal File Descriptors are:

PACK:FRED.TSK	
FRED.TSK	The same File, if PACK is the system volume
FRED	The same File, if TSK is the default extension.
ABC:FOO	Default extension, specified volume.
CARD:	Name of a device.
DSC1:	Name of a device.

Optional Operands

Some commands have optional operands. These are annotated with brackets surrounding the entire optional part of the command, as follows:

COMMAND aaaa[, [bbbb] [,cccc]]

In this example, the operand aaaa is not optional, while the operands bbbb and cccc are optional. If cccc is specified and bbbb omitted, the absence of bbbb is denoted by two successive commas.

ORDER xxxx [,yyyy [,zzzz]]

In this example, operand zzzz may not be entered without operand YYYY. This is shown by the nested brackets. Legal forms of this command are:

OR	xxxx
OR	xxxx,yyyy
OR	xxxx,yyyy,zzzz

Whereas in the previous example, legal forms are:

COM	aaaa
COM	aaaa,bbbb
COM	aaaa,bbbb,cccc
COM	aaaa,,cccc

General Syntactic Rules

Multiple commands may appear on a line, separated by semi colons (;).

Certain commands must appear last on a line, or must be the only command on the line. These special commands are discussed in the sections dealing with the individual commands.

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

ERROR RESPONSE

If command input is not acceptable to the Command Processor, or an error condition is detected while processing a command, an error message is output to the system console. The general format of the message is:

XXXX-ERR YYYY ZZZZ #= N

where XXXX is a descriptor of up to four characters (such as MNEM, CSS, FORM) which indicates the general class of error.

YYYY is a descriptor of up to four characters which further identifies the particular type of error. The XXXX and YYYY error descriptors are defined in Appendix 2 and are listed with each command description as appropriate. ZZZZ is a field which is output when further information is helpful in identifying the error. It consists of a keyword followed by a value, such as LU=5 or FD=PACK:FILE.

The #=N field indicates in which command, of multiple commands on a line, the error was detected. If N=1, this field is omitted.

The error response to an unrecognized command is:

MNEM-ERR #=N

* Any command following an erroneous command is ignored.

The error messages generated in response to certain File Management errors may contain references to Logical Units used by the Command Processor. Table 3-1 lists the Logical Units of the Command Processor and their use.

TABLE 3-1. COMMAND PROCESSOR LOGICAL UNITS

LU	USE
0	Console Device
1	Log Device
2	Work Unit
3	Work Unit
4	BUILD,$BUILD
5	First CSS level
6	Subsequent CSS levels
.	.
.	.
.	.

GENERAL SYSTEM COMMANDS

The following commands have a global effect on the system or display global system information:

SET TIME
DISPLAY TIME
VOLUME
SET LOG
DISPLAY MAP
SET PARTITION

Set Time

The SET TIME command should be entered when the system is first loaded and after any power failure. It may be entered at any other time that the system clock is incorrect. The day is updated automatically at midnight, however, the month and year must be updated by the operator. The format of this command is:

SET TIME [mm/dd/yy][,hh:mm:ss]

where:
mm = month
dd = day
yy = year
hh = hours (24-hour clock)
mm = minutes
ss = seconds

All operands are in decimal. Example:

SET TIME 2/24/75,3:35:00

Alternatively (by SYSGEN option), the date operand may be entered in the format: dd/mm/yy.

If a SET TIME command is entered while there are uncompleted time intervals, (see SVC 2 code 23 – *OS/16 MT2 Programmer's Reference Manual*), the tasks which initiated the uncompleted intervals are affected in the following way according to the type of interval:

1. Seconds from midnight. The date is updated; this has no effect on any time of day interval even if the date entered differs from the previous date entered. The time difference is used to adjust all seconds from midnight intervals.

2. Milliseconds from now. Elapsed time intervals are unaffected by a change in the time by a SET TIME.

   For example, if the current date is 11/22/74 and the current time is 11:50 AM and there are three intervals outstanding:

   1. Time of day interval set to complete an 11/22/74 at 2:00 PM.

   2. Time of day interval set to complete on 11/23/74 at 8:00 AM.

   3. Elapsed time interval set to complete on 11/22/74 at 1:00 PM.

If a SET TIME 11/21/74,10:50:00 is entered, the time intervals are as follows:

1. Time of day interval set to complete on 11/21/74 at 2:00 PM.
2. Time of day interval set to complete on 11/22/74 at 8:00 AM.
3. Elapsed time interval set to complete on 11/21/74 at 12 NOON.

Possible error responses to SET TIME are:

FORM-ERR		Command syntax error; e.g., SET TIME 12/16/74/1:00:00
PARM-ERR	VAL	Operand error: e.g., SET TIME 68/74/18,1:00:00
PARM-ERR	PRQD	Operand missing: e.g., SET TIME 1/1/74,

Display Time

This command is used to display the current date and time to this system console or to a specified file or device. Its format is:

DISPLAY TIME [,fd]

The optional operand fd specifies the file or device to which the display is to be output; if omitted, the display is output to the system log. The display has the following format:

mm/dd/yy hh:mm:ss

or alternatively (by SYSGEN option):

dd/mm/yy hh:mm:ss

Possible error responses to DISPLAY TIME are:

FORM-ERR		Command syntax error; e.g., DISP TIME/PR:
PARM-ERR	MNEM	Invalid Display option; e.g., DISP TM
PARM-ERR	FD	Invalid output File Descriptor; e.g., DI TIM,135
ASGN-ERR	type	Output fd could not be assigned for reason denoted by type
IO-ERR	type	I/O error denoted by type encountered on output fd.

Volume

This command is used to set or display the name of the system default volume. Any commands that do not explicitly specify a volume name use the system volume as a default. The format of this command is:

VOLUME [voln][/SPL]

where voln is a one to four character volume identifier. No test is made to ensure that the volume is actually on line at the time the command is entered. If voln is omitted, the current value of the system default volume is displayed, along with the current OS overlay file and Roll and Spool volumes as appropriate. /SPL specifies voln as the system spool volume, and is valid only for systems with spooling support. All spool files are allocated on volume voln.

Possible error responses to VOLUME are:

FORM-ERR		Command syntax error; e.g., VOL FRED:
PARM-ERR	VAL	Parameter error; e.g., VOLUME 157

Set Log

This command is used to set the system log device. The system log receives a copy of all system console I/O. This copy includes:

- All command lines entered from the console;
- All responses to these commands (other than prompts);
- All messages logged by tasks.
- All task input responses.

The format of this command is:

SET LOG [fd [,COPY]]

The copy is produced on the file or device specified by fd.

In systems configured with CLOCK support, all logged I/O is preceded by the current time:

hh:mm:ss (Logged I/O)

The log may be changed at any time by another SET LOG command. If no operands are specified, logging is terminated. Logging is automatically terminated under the following conditions:

I/O error on the log device
System initialization

When logging is terminated, the system console device receives all output directed to the system log. The command that would have been logged when an I/O error occurred is not executed and must be re-input.

The SET LOG command may be used for two primary purposes. These are:

- To provide a historical record of system operation, often on a Magnetic Tape or Direct Access File.
- To allow system output, e.g., displays, log message, etc., to proceed on a high-speed device rather than on a system console.

If the optional COPY operand is specified, the system console receives all outputs that it would have received if no SET LOG command were in effect. This facility would normally be used when the logging is basically for historical purposes. If COPY is not specified, however, the system console receives no outputs other than prompts. This is the case when logging is directed at a high-speed printer device used by the console operator as an adjunct to the system console.

The Log device may be shared with user task output.

Possible error responses to SET LOG are:

FORM-ERR		Command syntax error; e.g., SET LOG/PR:
PARM-ERR	MNEM	Invalid set option; e.g., SET FOX
PARM-ERR	FD	Invalid File Descriptor; e.g., SET LOG PACK.X.Y
ASGN-ERR	type	Log device/file could not be assigned for reason denoted by type; e.g., device off-line or assigned for exclusive use to a task
IO-ERR	type	I/O error denoted by type, encountered on output device/file

Display Map

This command is used to display a memory map of the entire system, including all partitions and system space, to the console or to a specified file or device. The format of the command is:

DISPLAY MAP [,fd]

The optional operand fd indicates the file or device to which the map is to be output; if omitted, the display is output to the system log.

The format of the map is shown in Figure 3-1.

#	NAME	START	STAT	PRI
	.BG	6300		
1	ABC	9000	A	37
2	XYZ	A000	P	126
3		B000		
4	PDQ35	C000	RDO	13
5		E000		
	.SYS	E000		
	FBOT	F7B0		
	MTOP	10000		

Figure 3-1. Display Map Format

The NAME field is the name of a task or of a partition. Where no name is shown, a vacant foreground partition is indicated. The names .BG and .SYS indicate the Background and System Space partitions, respectively. Any name other than these is the name of a Foreground task.

The START field indicates the starting address of the partition.

The PRI field indicates the priority in decimal of all tasks currently in Foreground or Background partitions. The STAT field indicates the Status of these tasks, as follows:

D Dormant
P PAUSEd
A "Active;" i.e., in any state other than Dormant or PAUSEd

A task that is displayed on the map as "Active" may in fact be in a Wait state; if this information is desired, the console operator may use the TASK and the DISPLAY PARAMETERS commands to get the actual Wait Status halfword of a given task. The status may be preceded by an "R" indicating the task is memory resident. It may be followed by an "O" indicating that this task caused the task previously in this partition to be rolled out.

The last two lines of the display contain the current value of FBOT and MTOP (bottom of used System space and memory top, respectively).

Possible error responses to DISPLAY MAP are:

FORM-ERR		Command syntax error; e.g., DISP MAP/PR:
PARM-ERR	MNEM	Invalid Display option; e.g., DISP MOP
PARM-ERR	FD	Invalid File Descriptor specified; e.g., DISP MAP, 123:
ASGN-ERR	type	Output Device/File could not be assigned for reason denoted by type; e.g., device is off-line or assigned for exclusive use to a task.
IO-ERR	type	I/O error denoted by type encountered on output device/file

Set Partition

This command is used to partition the user and system space in the system. If Roll support is configured in the system, the command also allocates the Roll files required by the specified partitioning. The format of the command is:

SET PARTITION pname/address [,pname/address] . . .

where pname is:

1. A decimal number, indicating one of the Foreground partitions configured in the system,

or

2. .SYS indicating the System Space partition,

and address specifies the new starting address of the designated partition in hexadecimal. If the optional Memory Protect Controller is configured in the system, the starting address of Foreground Partition 1 and of .SYS must be on a 1K boundary; otherwise, the address must be on a halfword boundary. There is no restriction on the order of parameters. Partitions are arranged in memory as illustrated in Figure 3-2.

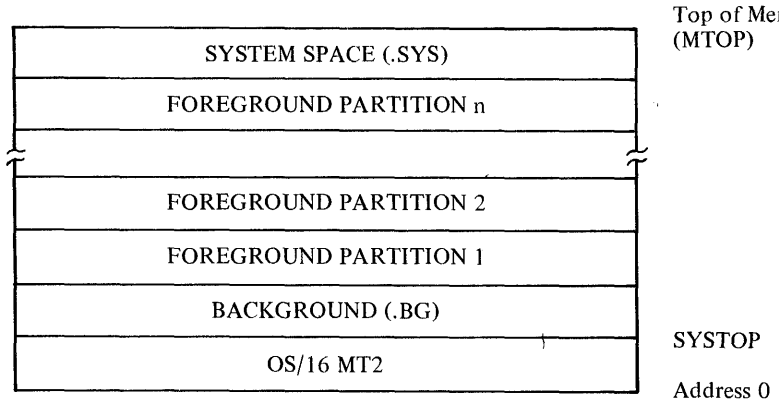


Figure 3-2. Partition Format

The size of a partition is set by specifying the start address of the next higher partition. The SET PARTITION command may be used to reduce the number of partitions by specifying the same starting address for two more partitions, thus giving a Zero size to all but the last such partition.

The entire command is processed before any change is made to the partitioning of the system. If a partitioning error is detected, the partitions are left unchanged. If an error is detected in allocating the Roll files, the partitions are changed and an error message is logged. This command is rejected if:

1. pname specifies a number greater than the number of configured foreground partitions,
2. a partition affected by a new start address is non-vacant,
3. address specifies an address less than SYSTOP (top of OS) or greater than FBOT (bottom of used System Space),
4. address specifies a starting address greater than the starting address of the next higher partition, or less than the starting address of a lower partition. The exception to this is that, if the effected partitions are vacant, the starting addresses of these higher, or lower, partitions are set to the specified address, thus causing the start addresses of vacant partitions to be kept in memory order.
5. in an extended memory system, a partition may not cross a 32KB boundary except for the first 32KB boundary.

After initial loading, an OS/16 MT2 System is normally partitioned with all start addresses equal to the first address not occupied by the system code (SYSTOP). A SET PARTITION command must be issued before a task may be loaded. If the SET PARTITION command is omitted from the system at system generation time, the start-up address of .SYS is preset to MTOP (no system space).

Assume an OS/16 MT2 system configured with 64K and four Foreground partitions, and SYSTOP has a value of 20K (5000 hexadecimal). Table 3-2 illustrates the effect of various SET PARTITION Commands entered in the following order:

TABLE 3-2. SET PARTITION EXAMPLES

COMMAND	PARTITION START	COMMENTS
NONE	.BG 5000 1 5000 2 5000 3 5000 4 5000 .SYS 5000	State after initial load of OS.
SE PA 1/E000	.BG 5000 1 E000 2 E000 3 E000 4 E000 .SYS E000	Create .BG with 36K, 8K system space. Partitions 1, 2, 3 and 4 are Zero size.
SE PA 3/C000,4/D000	.BG 5000 1 C000 2 C000 3 C000 4 D000 .SYS E000	Start address of 1 and 2 adjusted down to maintain Zero size. .BG reduced to 28K. Partitions 3 and 4 each 4K.
SE PA 4/A000	.BG 5000 1 A000 2 A000 3 A000 4 A000 .SYS E000	Start address of 1, 2 and 3 are adjusted if partitions 3 and 4 are vacant and .BG is dormant. Otherwise the command is rejected.
SE PA 1/A000,2/C000,4/E000	.BG 5000 1 A000 2 C000 3 C000 4 E000 .SYS E000	Partitions 1 and 3 each 8K; 2 and 4 Zero size.

TABLE 3-2. (Continued)

COMMAND	PARTITION START	COMMENTS
SE PA 4/E800	.BG 5000 1 A000 2 C000 3 C000 4 E800 .SYS E800	Partition 3 enlarged to 10K, system space start adjusted upward to maintain zero size of partition 4.

For examples of SET PARTITION in an extended memory environment, see Chapter 6 of the *OS/16 MT2 System Planning and Configuration Guide,* Publication Number 29-431.

Possible error responses to SET PARTITION are:

PARM-ERR	PRQD	Required parameter missing; e.g., SET PAR;
PARM-ERR	MNEM	Something other than a decimal number or .SYS specified for pname; e.g., SET PA .SUS/E000.
PARM-ERR	VAL	Invalid decimal number specified for pname; invalid address specified for address; invalid alignment of address; e.g., SET PA 0/ABCDE
STAT-ERR	VAL	Address specified results in out of order partition; e.g., SE PA 1/B000, 2/A000 or if address is not on a 1K boundary when memory protect is included in the system.
STAT-ERR	ACTV	Partition bordering on a specified boundary contains active task.
FORM-ERR		Command syntax error; e.g., SET PAR 1=ABC0
ALLO-ERR	TYPE	Roll file could not be allocated for reason denoted by type. The partitions are set up as specified. This is a warning only.

UTILITY COMMANDS

This group of commands is useful in debugging, or in building Command Substitution System files, but otherwise has no effect on the system or its tasks:

BIAS BUILD
EXAMINE ENDB
MODIFY

Bias

This command is used to set a base address for the EXAMINE and MODIFY commands. Its format is:

BIAS [{ address / * }]

The operand address is a hexadecimal bias to be added to the address or addresses given in any subsequent EXAMINE or MODIFY command. If the operand is omitted, all addresses specified in subsequent EXAMINE and MODIFY commands are treated as unbiased; that is, they are assumed to be absolute physical addresses. If * is specified, the bias is set to be the physical address of the first location of the currently selected task's partition (see the TASK command).

A BIAS command overrides all previous BIASes.

System initialization sets the BIAS to Zero.

Possible error responses to BIAS are:

FORM-ERR		Command syntax error; e.g., BI,*
PARM-ERR	VAL	Parameter error; e.g., BI X
FUNC-ERR	TASK	* specified, no currently selected task

Examine

This command is used to examine the contents of memory. There are two formats:

(a) EXAMINE address [,n]
(b) EXAMINE address/address

The EXAMINE command using format (a) causes the contents of the memory location specified by address (as modified by any previous BIAS command) to be displayed. The decimal operand n specifies the number of halfwords to be displayed. If n is omitted, one halfword is displayed.

Using format (b), all data from the first address to the second is displayed in hexadecimal. The BIAS is added to both addresses. The command is rejected if the second address is less than the first address. The memory area to be examined may not cross extended memory modules (example: EXAMINE 0/3FFFF).

All addresses presented are rounded down to halfword boundaries by the system. The command is rejected if an address plus the bias is not less than the top of memory

Possible error responses to EXAMINE are:

FORM-ERR		Command syntax error; e.g., EXAM, 100-200
PARM-ERR	VAL	Parameter syntax error; e.g., EXA X, 16
PARM-ERR	PRQD	Required parameter missing; e.g., EXA

Modify

The MODIFY command is used to change the contents of memory.

MODIFY address, [[data], [data] ...]

causes the contents of the halfword location specified by address (modified by any previous BIAS command) to be replaced with data. The modify address must be aligned on a halfword boundary and must be less than the top of memory.

If the operand data is omitted, the modify address has its contents replaced with zero. Each data field consists of 0-4 hexadecimal digits which are to be put into memory starting at the location specified by address. Any string of data less than four characters is right-justified and left-zero filled.

If an invalid data operand error is detected, the locations previous to the location to be updated by the erroneous data have been updated. In this case, the EXAMINE command should be used to verify the effects of the MODIFY. The MODIFY command is not valid across the 32KB boundaries of extended memory modules (example: MODIFY 1FFFC,0,0,0,0 is not valid).

Possible error responses to MODIFY are:

FORM-ERR		Command syntax error; e.g., MOD 124/5
PARM-ERR	VAL	Parameter value error; e.g., MOD 123,0 (not halfword boundary)
PARM-ERR	PRQD	Required parameter missing; e.g., MO;

Build

BUILD and ENDB permit the user to copy data from the system console to an arbitrary device or file (These commands may also be entered from a CSS file. Following a BUILD command, subsequent lines from the console are not treated as commands, but as data, and are copied to the device or file until an ENDB is encountered. The format of these commands is:

BUILD fd
...
ENDB

The operand fd is the device or file. If fd refers to a non-existent Direct-Access File, an Indexed File by that name is allocated, with a logical record length equal to the SYSGENed command buffer length, blocksize of 1, index blocksize of 1, and keys of 0000.

The BUILD command must be the last command on its input line. Further data appearing on that line is treated as a comment and is ignored.

The ENDB command must appear in the first four characters of the line; any subsequent characters in that line are ignored.

The BUILD command may be entered from the console only if no CSS files are active. The BUILD command may be entered from a CSS file.

Possible error responses to BUILD are:

FORM-ERR		Command syntax error; e.g., BUILD/PTRP
PARM-ERR	FD	Invalid File Descriptor or no Direct-Access File support
ASGN-ERR	type	Output file/device could not be assigned for reason denoted by type
FUNC-ERR	SEQ	CSS file active, entered from system console
PARM-ERR	PRQD	Required fd not specified; e.g., BUILD;
ALLO-ERR	type	Specified fd could not be allocated for reason denoted by type

No error response is possible from ENDB if a BUILD statement is being processed. If ENDB is not entered as the first four characters in the command line, the line is copied to the BUILD file. If no BUILD statement has been entered previously, the response to ENDB is:

FUNC-ERR	SEQ	ENDB command entered without previous BUILD.

TASK RELATED COMMANDS

The following commands are related to particular tasks executing in the OS/16 MT2 environment:

LOAD	ASSIGN
LDBG	DISPLAY LU
LFGR	CLOSE
TASK	OPTIONS
START	SET PRIORITY
PAUSE	DISPLAY PARAMETERS
CONTINUE	SEND
CANCEL	DISPLAY REGISTERS

Load Image

This command is used to load image format tasks into a Foreground or Background partition. It is also used to load a resident library or task common partition. The format of this command is:

LOAD taskid [,fd] [,*]

The taskid field specifies the name to be assigned to the task to be loaded. The fd field specifies the file or device from which the task is to be loaded. If fd specifies a Direct-Access File and no extension is specified, a default extension of .TSK is used. If fd is omitted, a Direct-Access-File name of taskid.TSK is used. This command is not affected by the TASK command; it may be entered at any time.

The specified task is loaded into an available partition (a partition whose address matches the least significant 4 digits of the task established bias). This command is rejected if:

- an available partition is not vacant and the system does not support Roll.
- If Roll is supported, but there is a task already rolled out of an available partition or the task currently occupying the partition is now lower priority than the task being loaded or is not rollable.
- There is no partition whose starting address matches the task's established bias.

Resident Library and Task Common partitions are loaded in the same way as tasks. The * option indicates that the task should be loaded only in the partition specified when the task was established (extended memory systems only).

Some examples of the LOAD command are given in Table 3-3.

TABLE 3-3. LOAD COMMAND EXAMPLES

ACTION	COMMAND
Load task from Paper Tape Reader. Give task ID of ABC.	LOAD ABC,PTRP:
Load task from file SYS:CAL.TSK where SYS is the default volume. Use CAL as task ID.	LO CAL
Load a Resident Library partition from file VOL:RTL. (blank extension). Give task ID of RLIB.	LOA RLIB,VOL:RTL.

Possible error responses to LOAD are:

FORM-ERR		Command syntax error; e.g., LOAD ABC/TSK.TSK
PARM-ERR	FD	Invalid file description specified; e.g., LO ABC,X.B.C
PARM-ERR	TASK	Invalid TASK ID specified; e.g., LOAD 123
ASGN-ERR	type	Specified device/file not assigned for reason denoted by type.
LOAD-ERR	type	SVC 6 error encountered during load. Specific error denoted by type.

Load Background (Object Code)

LDBG is used to load object format tasks into the Background partition. Its format is:

LDBG fd [,bias]

The parameter fd specifies the file or device from which the task is to be loaded. If fd specifies a Direct-Access-File and no extension is specified, an extension of .OBJ is used.

The optional parameter bias specifies the address in hexadecimal at which to start loading the object format task. This address must be within the background partition.

If the optional bias parameter is omitted, a default value of partition start address plus size of UDL is used. The UDL is taken as

X'24'	if the system has no floating point support
X'44'	if the system has SPFP support
X'84'	if the system has DPFP support

The loader sets the task's options RES and either NOFLOAT, FLOAT and/or DFLOAT to match the support offered by the system.

Possible error responses to LDBG are:

FORM-ERR		Command syntax error; e.g., LD MAG1:/6000
FUNC-ERR	SEQ	Background partition not dormant
PARM-ERR	FD	Invalid File Descriptor; e.g., LDBG 1:ABC
PARM-ERR	VAL	Invalid bias value; e.g., LDBG PTRP:.1ABCD
ASGN-ERR	type	Specified fd could not be assigned for reason denoted by type field
LOAD-ERR	DCHN	Ref-Def loop error encountered in task
LOAD-ERR	CKSM	Checksum error detected
LOAD-ERR	SEQ	Sequence numbers out of order in input
LOAD-ERR	OBJ	Illegal object format item detected
LOAD-ERR	MEM	Attempt to load into an address outside of the background partition
IO-ERR	type	I/O error denoted by type field encountered during load

Load Foreground (Object Code)

LFGR is used to load object format tasks into a foreground partition. Its format is:

LFGR n,taskid [,fd] [,bias] [,CSEG]

where

n specifies the foreground partition number, defining the partition into which the task is to be loaded.

taskid specifies a 1 to 8 character name to be assigned to the task once it is loaded.

fd specifies the file or device from which the task is to be loaded.

bias specifies the hexadecimal address within the partition from which to start loading the object code task.

CSEG is an option used on extended memory systems only. It specifies that task common or sharable library segments used by the task are resident in the second 32KB memory module. See Chapter 6 of the OS/16 MT2 Planning and Configuration Guide, Publication Number 29-431, for details on extended memory.

If the optional fd parameter is omitted, the extension '.OBJ' is appended to the taskid to form a direct access device file descriptor. If fd is given with no extension, the default '.OBJ' is appended to the given filename.

If the optional bias parameter is omitted, a default value of partition start address plus size of UDL is used. The UDL is taken as

X'24'	if the system has no floating point support
X'44'	if the system has SPFP support
X'84'	if the system has DPFP support

The loader sets the task's options RES and either NOFLOAT, FLOAT and/or DFLOAT to match the support offered by the system.

Possible error responses are:

FORM-ERR		Command syntax error
FUNC-ERR	SEQ	Partition not vacant
PARM-ERR	FD	Invalid file descriptor
PARM-ERR	VAL	Invalid bias given
ASGN-ERR	type	Fd could not be assigned for reason denoted by type field
LOAD-ERR	NPRT	No such partition
LOAD-ERR	CKSM	Checksum error
LOAD-ERR	SEQ	Sequence numbers in object code out of order
LOAD-ERR	OBJ	Illegal object code item
LOAD-ERR	MEM	Attempt to load outside partition
LOAD-ERR	PFUL	Partition occupied
IO-ERR	type	I/O error occurred during load

Task

This command is used to set or display the currently-selected task. Many OS/16 MT2 task-related commands affect only the currently-selected task. The TASK command is used to select one of the tasks in the system, either Foreground or Background. Subsequent task-related commands affect this task. The format of the command is:

TASK [taskid]

where taskid is the name of some foreground task in the system, or is the name of the background task, .BG. If taskid is omitted, the currently selected task is displayed in the message:

TSKID=taskid

The commands affected by TASK are:

- All task-related command except LOAD, LFGR, and LDBG.
- Magnetic Tape and File Control Command, Format (b)
- Some CSS Commands
- BIAS command, with * operand

For example:

T ABC	Set current task = ABC
CL 2,3,4	Close ABC's LU 2,3,4
AS 2,CARD:	Assign ABC's LU 2 to device CARD
T XYZ	Set current task = XYZ
CAN	Cancel task XYZ
ST B100	Start task XYZ at address B100
T .BG	Set current task = background
PAUSE	Pause the Background task
...	

NOTE

When CSS is started, the current value of TASK is associated with the CSS file as the currently-selected CSS task. If a CSS file executes a TASK command, it affects only that CSS file's commands, and does not change the value of TASK associated with the console.

If the currently selected task is a Foreground task which is deleted from the system or if no TASK command has been entered, there is no currently selected task. Task related commands (other than LOAD, LFGR, or LDBG) are rejected with a FUNC-ERR TASK if there is no currently-selected task. In a Background only system, the currently selected task is always set to .BG, and the TASK command is not recognized.

Possible error responses to TASK are:

FORM-ERR		Command syntax error; e.g., TA .BG,ABC
PARM-ERR	TASK	Invalid TASK ID; e.g., TAS 1234;
FUNC-ERR	TASK	Specified TASK ID not found in system.

Start

This command is used to start a task executing. The currently-selected task is started, if it is dormant or Paused; otherwise, the command is rejected. The format of this command is:

START [address] [,args to prog]

The operand address represents the address at which the program is to be started. If address is omitted, the currently-selected task is started at the transfer address specified when established; or the default transfer address set when the task was loaded.

The optional field, args to prog, contains arguments that are to be passed to the task for its own decoding and processing. All characters between the comma beginning the field and the next terminator (semi-colon or carriage return) are moved to memory beginning at UTOP. The characters are terminated in memory by a carriage return. If this operand is omitted, a carriage return is stored at UTOP. If there is not enough memory between UTOP and CTOP to pass all the characters, the command is rejected with a FUNC-ERR ARG. In multiple command lines, a START command can only be the last command on the line.

Possible error responses to START are:

FORM-ERR		Command syntax error; e.g., START 100/5
PARM-ERR	VAL	Parameter value error; e.g., START *73,GO
FUNC-ERR	SEQ	Task not Dormant or Paused
FUNC-ERR	TASK	No currently selected task
FUNC-ERR	ARG	Insufficient memory between UTOP and CTOP to pass all ARGS to PROG.

Some examples of a START command are:

ST B138	Start task at X'B138' and pass a carriage return to the program
ST C100,NOSEQ, SCRAT	Start task at X'C100' and pass 'NOSEQ,SCRAT' to the program
ST ,1000,ABC	Start task at transfer address and pass '1000,ABC' to the program.

Pause

This command causes the currently-selected task to pause as though it had issued an SVC 2 CODE 1, PAUSE (See OS/16 MT2 Programmer's Reference Manual). Its format is:

PAUSE

Any I/O Proceed ongoing at the time the task is Paused, is allowed to proceed to completion after the PAUSE. If the task is in any wait state at the time the PAUSE command is entered, all wait conditions must be removed before the pause can become effective. This command is rejected if the task is dormant or paused at the time the command is entered.

Possible error responses to PAUSE are:

FORM-ERR		Command syntax error; e.g., PAUSE .BG
FUNC-ERR	SEQ	Task Paused or dormant
FUNC-ERR	TASK	No currently selected task

Continue

This command resumes execution of a task which is paused. Its format is:

CONTINUE

Possible error responses to CONTINUE are:

FORM-ERR		Command syntax error; e.g., CONTINUE .BG
FUNC-ERR	SEQ	Task not paused
FUNC-ERR	TASK	No currently selected task

Cancel

The CANCEL command terminates a task and sets the task's return code to 255. The format of this command is:

CANCEL

If the task is non-resident, it is removed from the system at this time, all outstanding I/O is terminated and the task's LUs are closed. If the task is resident, it is not removed from the system; its LUs are not closed, but are checkpointed. This command may be entered even when the currently-selected task is dormant. It has no effect on a resident task that is already dormant, unless preceded by an OPTIONS NONRES command; it may be used to remove a non-resident task, which has been loaded but not started, from the system. The normal response to this command is:

taskid: END OF TASK 255

Possible error responses to CANCEL are:

FORM-ERR		Command syntax error; e.g., CANCEL .BG
FUNC-ERR	TASK	No currently selected task

Assign

This command assigns a device or file to one of a task's Logical Units. The format of this command is:

ASSIGN lu, fd [, [access-priv] [,keys]]

where lu is the LU number in decimal, fd is the File Descriptor specifying the device or file to be assigned, access-priv is the desired access privilege, and keys specifies the Write-Read protection keys of the file or device. The keys are set with ALLOCATE and can be changed with REPROTECT.

Access-priv may contain one of the following:

SRO	Sharable Read-Only
ERO	Exclusive Read-Only
SWO	Sharable Write-Only
EWO	Exclusive Write-Only
SRW	Sharable Read-Write
SREW	Sharable Read, Exclusive Write
ERSW	Exclusive Read, Sharable Write
ERW	Exclusive Read-Write

If access-priv is omitted, SRW is assumed. The command is rejected if the requested access privilege cannot be granted. If a console device is assigned for SRO access, the console paper tape reader is assigned to the logical unit.

The optional operand, keys, specifies the Write and Read protection keys for the file. The keys are specified as a hexadecimal halfword (1 to 4 digits), the first byte of which represents the Write Key and the second byte, the Read Key. If omitted, the default is 0000. These keys are checked against the appropriate existing keys for the file or device; the command is rejected if the keys are invalid.

An assigned Direct-Access File is positioned at the end of the file for access privileges SWO and EWO; it is positioned at the beginning of the file for all other access privileges.

This command is rejected if the specified LU is assigned and the currently selected task is not dormant. To reassign an LU for an active task, the LU must first be CLOSEd.

Possible error responses to ASSIGN are:

FORM-ERR		Command syntax error; e.g., AS 1/CR:
PARM-ERR	PRQD	Required parameter missing; e.g., AS,CR:
PARM-ERR	FD	Invalid File Descriptor; e.g., AS1,PACK:ABC:TSK
PARM-ERR	VAL	Invalid keys specified; e.g., ASS11,CR:,,12345
PARM-ERR	MNEM	Invalid access privilege mnemonic; e.g., ASSIGN 1, CR:,SWR
ASGN-ERR	type	The assign failed for reason denoted by type.
FUNC-ERR	TASK	No currently selected task

Display LU

This command permits the operator to display all assigned Logical Units of the currently-selected task. Its format is:

DISPLAY LU [,fd]

where the optional operand fd signifies the file or device for the display. If the optional operand is omitted, the display is output to the system log.

An example of the display output is shown in figure 3-3.

LU	PRIV	FD/NAME
1	SRO	CR:
2	ERW	VOLN:ABC.OBJ
3	SWO	PR:
4	SRW	VOLN:SCRATCH.
5	SRW	TTY:
10	SWO	NULL:

Figure 3-3. Display Logical Unit Format

Possible error responses to DISPLAY LU are:

FORM-ERR		Command syntax error; e.g., DISP LU/PR:
PARM-ERR	MNEM	Invalid Display option; e.g., DISP LO
PARM-ERR	FD	Invalid File Descriptor; e.g., DISP LU,123
IO-ERR	type	I/O error denoted by type encountered on output device/file
FUNC-ERR	TASK	No currently selected task

Close

This command permits the operator to close (de-assign) the files or devices assigned to one or more of the currently-selected task's Logical Units. The format of this command is:

CLOSE { lu [,lu] ... / ALL }

where the lu operand or operands are decimal numbers signifying the Logical Unit(s) to be de-assigned. If ALL is specified, all the LU's of the currently-selected task are closed.

Closing an unassigned LU does not produce an error message. Closing LUs of active tasks is permitted. Users should be aware that this may cause inadvertent loss of data on files being written to.

If an operand is invalid, previous valid operands in the same command are processed. In this case, use the DISPLAY LU command to verify the state of the logical units.

Example of the CLOSE command are:

CL 1,3,5	Close LUs 1,3 and 5
CLOSE AL	Close all LU's of task

Possible error responses to CLOSE are:

FORM-ERR		Command syntax error; e.g., CLO 1/2/3
PARM-ERR	VAL	Invalid LU specified; e.g., CLO 112
PARM-ERR	PRQD	Required parameter missing; e.g. CL 2,,3
CLOS-ERR	type	Close failed for reason denoted by TYPE field
FUNC-ERR	TASK	No currently selected task

Options

This command is used to specify or to change certain options of the currently selected task. The format of this command is:

OPTIONS opt [,opt] ...

where opt may be any one or more of the following options:

AFC	Continue after arithmetic fault (message logged)
AFP	Pause after any arithmetic fault
RES	Task is memory-resident
NONRES	Task is to be removed from memory at EOT
FLOAT	Task requires Single Precision floating point registers
NOFLOAT	Task requires no Single Precision floating point registers
DFLOAT	Task requires Double Precision floating point registers
NODFLOAT	Task requires no Double Precision floating point registers
SVCP	Treat SVC 6 as illegal SVC (.BG only)
SVCC	Treat SVC 6 as NOP (.BG only)
UT	Task is to be a user task (non-privileged)
ET	Task is to be an Executive task
ROLL	Task may be rolled out
NOROLL	Task may not be rolled out
COMP	Task uses BOSS/DOS/RTOS SVC1 format parameter block
NOCOMP	Task uses OS/16 MT2 SVC 1.

The options are paired, but any option may be entered in any order. If both members of a pair are entered, the last one entered is accepted as the correct option. Thus:

OPTIONS RES,NONR

specifies non-resident. The LDBG command has no effect on existing options of the background task.

The ROLL and NOROLL, AFP and AFC, COMP and NOCOMP, UT and ET options are normally set up at task establishment time, but may be modified by the console operator. This command may be entered at any time, regardless of the task's state. Note that the sequence OPT NON;CANCEL always causes the currently-selected task to be removed from memory. The sequence OPT RES;CANCEL always causes the currently-selected task to enter the dormant state.

Note that for a specific task to be eligible for roll-out:

1. The OS must be configured with ROLL support.
2. The task must have the ROLL option set.
3. A disc volume on the system must have been marked on-line as the ROLL volume.
4. There must be no other task currently rolled out from that partition.
5. The task must not be in a wait state.

The SVCP and SVCC options apply only to the Background task; they are ignored if specified for a Foreground task.

Note that if a task uses floating point arithmetic, OPT FLOAT and/or OPT DFLOAT must be specified when the task is established so that the UDL includes the register save areas.

Note that for a task to use floating point arithmetic (SPFP and/or DPFP)

1. The OS must be configured with the appropriate software floating point support, whether or not the processor has hardware floating point support.
2. The task must have the appropriate floating point option set.
3. The task's UDL must include the floating point register save areas. (Both TET/16 and the resident object code loaders reserve these areas by default).

Possible error responses to OPTIONS are:

FORM-ERR		Command syntax error; e.g., OPT ET/NO RO
PARM-ERR	MNEM	Invalid option specified or option not supported
FUNC-ERR	TASK	No currently selected task

If an operand is invalid, previous valid operands in the same command are processed. In this case, use the DISPLAY PARAMETERS command to verify the state of the task options.

Set Priority

This command is used to modify the priority of the currently-selected task. Its format is:

SET PRIORITY n

where n is a decimal number from 10 to 249 inclusive for User Tasks and from 0 to 255 inclusive for Executive tasks. The priority of the currently-selected task is set to n, subject to the following restriction:

If the task is a Foreground task, its priority may not exceed the maximum priority set up at task establishment time. If n is greater than the maximum established priority, then the latter is taken instead of n. This is not considered an error. In order to increase this priority, it is necessary to reestablish the task.

Display Parameters

This command is used to display certain parameters pertinent to the currently-selected task. The display appears on the console device, or alternatively on a device or file selected by the operator. The format of this command is:

DISPLAY PARAMETERS [.fd]

Parameters displayed are:

TASK	TASKID
CTSW	Current TSW status field
CLOC	Current location field of TSW/PSW
CPSW	Current task PSW
STAT	Task's Wait status halfword
TOPT	Task Options
CTOP	Top of user's partition
UTOP	Top of user task
UBOT	Bottom of user task
SLOC	Starting location
NLU	Number of Logical Units
MPRI	Task's maximum priority
SVOL	System Volume ID

The CLOC may be a program space address or an address in a system subroutine being executed on behalf of the task. NLU is given in decimal. SVOL is the ASCII System Volume ID. As such it is not specifically related to the currently-selected task, but it is given here for operator convenience.

TOPT is given in hexadecimal; the definition of task option bits is given in Table 3-4.

TABLE 3-4. TASK OPTION BITS

Bit	Hex Mask	Meaning
0	8000	0 – U-task; 1 – E-task
1	4000	0 – AFP; 1 – AFC
2	2000	0 – NOFLOAT; 1 – FLOAT
3	1000	0 – NONRES; 1 – RES
4	0800	0 – NOCOMP; 1 – COMP
5	0400	0 – Foreground task; 1 – Background task
6	0200	0 – SVCP; 1 – SVCC
7	0100	0 – NOROLL; 1 – ROLL
8		Reserved for system use
9	0040	0 – NODFLOAT; 1 – DFLOAT
10	0020	0 – NOCSEG;1 – CSEG (Extended Memory Systems Only)
11-15		Reserved for system use

STAT is given in hexadecimal; the definition of Wait status bits is given in Table 3-5.

TABLE 3-5. TASK WAIT STATUS BITS

Bit	Mask	Meaning if Set (1)
0	8000	Dormant
1	4000	I/O Wait
3	1000	Roll Wait
5	0400	Console Wait (PAUSED)
6	0200	Time Wait
7	0100	Trap Wait
8	0080	SVC 2 Wait
9	0040	SVC 5 or SVC 6 Wait
10	0020	SVC 7 Wait
11	0008	DCB Wait

CTSW is given in hexadecimal. For a definition of the status portion of the TSW, see *OS/16 MT2 Programmer's Reference Manual.*

For extended memory systems, the CPSW may be used to convert user program addresses into physical memory addresses. See Appendix 14.

Possible error responses to DISPLAY PARAMETERS are:

FORM-ERR		Command syntax error; e.g., DISP PARM/PR:
PARM-ERR	MNEM	Invalid Display option specified; e.g., DISP XYZ
PARM-ERR	FD	Invalid File Descriptor; e.g., DISP PA,ABCDE:
IO-ERR	type	I/O error denoted by type encountered on output device or file
FUNC-ERR	TASK	No currently selected task

Send

This command enables the operator to send a message to the currently selected task via the command processor.

The format of the command is

SEND (up to 64 character string)

The message is passed to the selected task following standard SVC 6 procedures. The message data passed to the task begins with the first non-blank character following SEND and ends with a carriage return as a terminator.

The receiving task must be capable of receiving messages.

For example:

```
TASK   TASKID
SEND   EXAMPLE MESSAGE (C/R)
```

will send the following message to task TASKID:

```
.CMDP  EXAMPLE MESSAGE (C/R)
```

The possible error responses to SEND are:

FUNC-ERR	TASK	No current task
PARM-ERR	PRQD	No message given
PARM-ERR	VAL	Message too long
FUNC-ERR	SEQ	Command out of sequence
SEND-ERR	NMSG	Task could not receive message

Display Registers

This command causes the sixteen user registers to be displayed to the console or a specified file or device. The registers can be examined while the task is active or dormant. The registers listed while the task is dormant are all viewed at the same time. However, the display registers command for an active task guarantees that only four registers are viewed at one time because an active task may be changing the registers while the OS is displaying them. Registers 0,1,2, and 3 are taken at the same time; registers 4,5,6, and 7 are viewed at the same time; registers 8,9,A, and B are examined at the same time; registers C,D,E, and F are examined at the same time. The format of this command is:

DISPLAY REGISTERS [,fd]

where:

fd specifies the file or device to which the display is output. If omitted, the display is output to the console.

Example/Response:

```
D R
0 - 3   0000    0C84    0000    0000
4 - 7   0000    0104    0D2E    029C
8 - B   0000    0000    0000    00B8
C - F   0000    0000    500D    2020
```

Possible error responses to DISPLAY REGISTERS are:

FORM-ERR	Command syntax error; e.g., DISP REG/
PARM-ERR	Invalid parameter; e.g., DISP YYZX
FD-ERR	Invalid file descriptor; e.g., DISP R,135
ASGN-ERR	Output device/file could not be assigned
IO-ERR	IO error encountered on output device/file

DEVICE AND FILE CONTROL COMMANDS

The following set of commands is used for device and file control. These commands are not affected by the setting of the currently selected task:

ALLOCATE	MARK	WFILE
DELETE	DISPLAY DEVICES	REWIND
XDELETE	FRECORD	RW
RENAME	FFILE	INITIALIZE
REPROTECT	BRECORD	SAVE
FILES	BFILE	PRINT
DISPLAY FILES		

Allocate

This command is used to create a Direct-Access File. There are two formats:

(a) ALLOCATE fd,INDEXED [, [lrecl] [/ [bsize] [/isize]] [,keys]]

(b) ALLOCATE fd,CONTIGUOUS, fsize [,keys]

the operand fd identifies the file to be allocated. Format (a) is used to allocate an Indexed File; format (b) is used to allocate a Contiguous File.

If INDEXED is chosen, the next operand, lrecl, is optional and specifies the logical record length. It cannot exceed 65,535 bytes. Its default is 108 bytes. It may optionally be followed by a slash mark (/) which delimits lrecl from bsize. The bsize operand specifies the block size, in 256-byte sectors, of the data blocks to be used for buffering and de-buffering operations on the file. If bsize is omitted, the default value is 1 sector (256 bytes). isize specifies the block size, in sectors of the index blocks. If isize is omitted, the default value is 1 sector (256 bytes). Note that, in order to assign this file, sufficient room must exist in system space for a buffer of the stated size. Therefore, if bsize or isize is very great, the file may not be assigned in some memorybound situations. isize and bsize may not exceed 255. lrecl, bsize and isize are specified as decimal numbers.

If CONTIGUOUS is chosen, the file size operand, fsize, is required and specifies the total allocation size in 256 byte sectors. This size may be any value up to the number of contiguous sectors existing on the specified volume at the time the command is entered. fsize is specified as a decimal number.

The last operand, keys, is optional. This operand specifies the Write and Read protection keys for the file. These keys are in the form of a hexadecimal halfword, the first byte of which signifies the Write key and the second byte the Read key. If this parameter is omitted, both keys default to zero.

Examples of the ALLOCATE command:

```
AL THISFILE,INDEX
```

allocates on the system volume an Indexed file named THISFILE. (blank extension) with a logical record of 108 bytes, a data blocksize of 1 sector, an index blocksize of 1 sector, and protection keys of zero.

```
AL PROGRAM.TSK,CO,64
```

allocates on the system volume, a Contiguous File named PROGRAM.TSK, whose total length is 64 sectors (16KB) and protection keys are zero.

```
AL FRED:EXAMPLE.OBJ,I,132
```

allocates on the volume FRED, a file named EXAMPLE.OBJ, whose logical record length is 132 bytes. The data block size and index block size of this file both default to one sector; the protection keys default to zero.

```
AL MORT:GREATBIG.BLK,IND,132/4/1
```

allocates on the volume MORT, an Indexed File named GREATBIG.BLK, whose logical record length is 132 bytes, using a data block size of 4 sectors and an index block size of 1 sector. The protection keys default to zero. Note that whenever this file is assigned, the system must have 1.25KB of available system space (the data block size plus the index block size) for buffers.

```
AL SAM:DATABASE.X,I,480,AA44
```

allocates on the volume SAM, an Indexed File named DATABASE.X, whose logical record length is 480 bytes, data block size and index block size are 1 sector, Write protection key is AA and Read key is 44. Note that the logical record length is permitted to exceed the data block size.

Possible error responses to ALLOCATE are:

FORM-ERR		Command syntax error; e.g., ALLO CO=64
PARM-ERR	MNEM	Invalid file type specified; e.g., ALLO ABC,CH
PARM-ERR	PRQD	Required operand missing; e.g., ALL ,64
ALLO-ERR	type	Allocate failed for reason denoted by type
PARM-ERR	FD	Invalid File Descriptor specified
PARM-ERR	VAL	Invalid size or keys specified; e.g., ALL ABC,IN,123456

Delete

This command is used to delete a Direct-Access-File. Its format is:

```
DELETE fd [,fd] ...
```

where fd identifies the file to be deleted. To be deleted, the file must not be currently assigned to any LU of any task. This command is rejected if there are no Direct-Access Devices in the system.

Possible error responses to DELETE are:

FORM-ERR		Command syntax error; e.g., DEL PACK:A/PACK:B
PARM-ERR	FD	Invalid File Descriptor; e.g., DEL A:1
DELE-ERR	type	Delete failed for reason denoted by type

Xdelete

This command is identical to the DELETE command except that no error is generated if the file does not exist. The XDELETE command can be used in a CSS procedure to insure that a file does not exist.

Rename

This command is used to change the name of an unassigned Direct-Access-File, a Direct-Access Volume, or Device. Its format is:

RENAME oldfd, newfd

Examples:

```
REN VOL1:MYFILE.CUR,MYFILE.OLD
REN MT01:,MT02:
```

The volume ID field of the new File Descriptor, newfd, may be omitted for Direct-Access Files. If it is entered, the system ignores it. To rename a Direct-Access Volume, oldfd must specify a volume name, including the colon. Attempts to rename the Null device are rejected.

Possible error responses to RENAME are:

FORM-ERR		Command syntax error; e.g., REN A:B C:D
ASGN-ERR	type	Old fd currently assigned
RENM-ERR	type	Rename failed for reason denoted by type
PARM-ERR	FD	Invalid File Descriptor specified
IO-ERR	type	I/O error denoted by type encountered updating the directory of volume
PARM-ERR	VAL	Specified old volume name not on line to system

Reprotect

This command is used to modify the protection keys of an unassigned Direct-Access-File or Device. Its format is:

REPROTECT fd, keys

where fd is the name of the file or device and keys is a hexadecimal halfword whose left byte signifies the new Write keys and whose right byte signifies the new Read key.

Possible error responses to REPROTECT are:

FORM-ERR		Command syntax error; e.g., REPR CR:/FF12
PARM-ERR	VAL	Invalid keys; e.g., REPR CR:,12345
ASGN-ERR	type	fd currently assigned or Reprotect failed for reason denoted by type
PARM-ERR	PRQD	Keys not specified; e.g., REPR CR:

Display Files

Files

This command is used to display information from the directory of a Direct-Access Volume on the system log device or the specified file or device. Its format is:

$$\left\{\begin{matrix}\text{FILES} \\ \text{DISPLAY FILES,}\end{matrix}\right\} [\text{voln:}] [\text{filename}] [\text{.ext}] [\text{,fd}]$$

where

voln: specifies the volume from which the information is to be displayed. If omitted the system default volume is assumed.

filename specifies the filename of interest. If given, all files not having this filename are ignored.

.ext specifies the extension of interest. If given, all files not having this extension are ignored.

The parameter fd specifies a file or device other than the system log to which the display is to be output.

The information displayed is defined in Table 3-6.

TABLE 3-6. FILES INFORMATION

Field	Meaning
FILENAME	Filename of file. Files are listed in directory order.
EXT	Extension of file.
TYPE	File type. CO for Contiguous Files, CH for Chained Files (created under OS/32 only), or IN for Indexed Files.
FLBA	First logical block address of file. Absolute sector number in hexadecimal. For Contiguous Files, this field contains a pointer to the first sector of the file. For Indexed Files, this field contains a pointer to the first Index block; if zero, the file does not have any Index blocks.
LLBA	Last logical block address of file. Absolute sector number in hexadecimal. For Contiguous Files, this field contains a pointer to the last sector of the file. For Indexed Files, this field contains a pointer to the last Index block.
LENGTH	For a Contiguous File, the length of the file in number of sectors allocated. For Indexed (or Chained), the logical record length specified at allocate time. In either case, a decimal number.
KEYS	The Write and Read protection keys of the file in hexadecimal in the form WWRR WW = Write key, RR = Read key.
CSEC/NLR	For Contiguous Files, this field contains the value of the current logical record number in decimal. For Indexed (or Chained) Files, it contains the number of logical records in the file, in decimal.
BKSZ	For Contiguous File, this field is blank. For Indexed (or Chained) Files, it contains the physical block size of the data blocks in the file in decimal number of sectors.
INBS	For Contiguous (or Chained) Files, this field is blank. For Indexed Files, it contains the block size of the Index blocks in decimal number of sectors.
CREATE	Date created (systems with CLOCK support only).
WRITE	Date last written to (systems with CLOCK support only).

Examples:

FILES MT16: or DISPLAY FILES MT16:

Lists all files contained on volume MT16 onto the console device.

FI or D F

Lists all files contained on the default system volume onto the console.

FI TEST or D F TEST

Lists all files with filename TEST contained on the default system volume onto the console.

FI FIXD:.OBJ,PR:

Lists all files with extension OBJ contained on volume FIXD onto the printer.

FI.,PR:

Lists all files with a blank extension contained on the default system volume onto the printer.

FI PACK:MYFILE.CAL

Lists file MYFILE.CAL contained on volume PACK onto the console.

The possible error responses to FILES (DISPLAY FILES) are:

PARM-ERR	FD	Invalid File Descriptor specified for voln or fd field; e.g., FILES 123:
PARM-ERR	VAL	Volume specified not on line to system
FORM-ERR		Command syntax error; e.g., FI ABC:/PR:
ASGN-ERR	type	Specified output device or file not assigned for reason denoted by type
IO-ERR	type	I/O error, denoted by type, encountered reading directory of specified volume or outputting to specified file or device.

Mark

The MARK command is used to:

- place a device Off-line
- place a device On-line
- specify system overlay volume
- specify system Roll volume

The format of this command is:

MARK fd, { ON [{ NEW / ROLL / PROTECT / OS }] ... / OFF [,OS] }

The mnemonic name of the device is specified by fd. If the device is the System-Console Device, the Null device, is assigned (Non Direct-Access device) or has any assigned files (Direct-Access Device), the command is rejected.

While a device is off-line, it cannot be assigned to any U-task. E-tasks are permitted to assign off-line devices.

If the device being MARKed ON or OFF is a Direct-Access Device, the fd used in the command is not the volume identifier, but the actual device mnemonic. For example, to MARK OFF a Disc name DSC1 which currently contains a volume named FRED, the operator enters:

```
MA DSC1:,OFF
```

This action removes the volume FRED from the system. The volume may now be changed, if DSC1 is a Removable-Cartridge Disc. To make the new volume known to the system, the operator enters:

```
MA DSC1:,ON
```

This causes the Volume Descriptor of the volume on DSC1 to be read, and the new volume ID (if the volume was changed) is made known to the system. The system responds with:

```
fd: voln
```

indicating the name of the volume placed on line. Removable Cartridges must not be dismounted from the system without using the MARK command. Failure to do so will cause a subsequent attempt to mark the volume on to be rejected, until the Disc Integrity Check Utility is run.

If the optional parameter, PROTECT, is specified in a MARK ON command, the device is marked as Write Protected. All assignments for access privileges other than Shared Read Only (SRO) and Shared Read/Write (SRW) are rejected with a privilege error. Shared Read/Write is changed to SRO. A Write File mark command to any file on the device is also rejected. This option may be used for any device regardless of the state of any hardware Write Protect feature.

For disc volumes, the system responds with:

```
fd: voln PROT
```

indicating that the volume is write protected. Hardware write protected discs will automatically be marked on with protect, even if PROTECT is not specified.

The optional parameter, NEW, is used to mark on discs that have been used on OS/32, OS/16 MT2 R03 or earlier revisions of OS/16. NEW may be specified on any MARK ON command provided the disc volume is not protected. The NEW option forces building of the extended directory structure above the existing directory (required for OS/16 R04 and above).

The optional parameter, OS, is used to specify that the volume being MARKed ON contains the system overlay file for systems that are overlayed. The File Descriptor of the system overlay file is:

voln:filename.ext

where voln is the name of the volume mounted on the device specified in the MARK command and filename.ext are the filename and extension specified to the OS/16 Boot Loader when the OS was loaded. The volume specified as system overlay volume must contain an exact copy of the file from which the OS was Boot-Loaded.

A MARK command specifying the OS parameter may be entered at any time to change the volume containing the System Overlay File. The OS parameter need only be specified on a MARK ON when changing to a different OS volume. This may not be used if contiguous files are deleted from the system.

Specifying the parameter OS in a MARK OFF command closes the System Overlay File, preventing any operator commands or system overlay requests from being satisfied. This command should only be entered to reboot an OS, boot in another OS or when the OS disc pack must be removed or a different pack inserted. If the parameter OS is specified in a system configured without system overlays, the command is rejected with a PARM-ERR MNEM message.

Specifying the parameter ROLL in the MARK ON command, indicates that the volume being placed On-line is to be used for allocation of the system ROLL Files. If ROLL is specified, the following actions are affected:

- If any volume is marked on-line as the ROLL Volume, the ROLL Files it contains are deleted, if no task is rolled out. If a task is rolled out, the MARK command is rejected.

- ROLL Files are allocated for the current partitioning on the specified Volume after it is placed on-line.

If Roll support is not included in the system, specifying ROLL causes the command to be rejected with a PARM-ERR MNEM message.

Possible error responses to MARK are:

FORM-ERR		Command syntax error; e.g., MARK LP:,ON/OS
PARM-ERR	FD	Invalid File Descriptor specified; e.g., MARK 123,ON
PARM-ERR	MNEM	Invalid keyword specified; e.g., MA CR:,OGG
PARM-ERR	VAL	OS parameter specified MARKing OFF volume not specified as System Overlay Volume, or for Non-direct-Access Device.
IO-ERR	type	I/O error denoted by type encountered in reading Volume Descriptor of Direct-Access Device being MARKed ON.
ASGN-ERR	type	Specified device fd or Overlay file could not be assigned for reason denoted by type.
ALLO-ERR	type	Roll file could not be allocated for reason denoted by type
STAT-ERR	PRIV	Device specified is assigned (Non-direct-Access Device) or contains files that are assigned (Direct-Access)
STAT-ERR	NOFF	Specified device was dismounted without being marked off. The volume can be marked on with PROTECT. If OS is specified, the disc volume is set on-line. Otherwise, the disc is left in its previous state. Disc Integrity Check must be run on the volume.
PARM-ERR	PRQD	Specified device contains the OS System Overlay File and parameter OS not specified.
DUPL-ERR	voln	Volume being MARKed specifies a duplicate volume name
VOLU-ERR		Volume being MARKed specifies an invalid volume name

Display Devices

This command allows the operator to determine the device number, keys, On-line/Off-line state, the volume name (for On-line Direct-Access Devices), and the state of software write protection of all devices in the system. The format of this command is:

DISPLAY DEVICES [,fd]

The operand fd specifies the device to which the display is routed; if omitted, the display goes to the System Console.

An example of the display output is shown in Figure 3-4.

```
NAME   DN   KEYS
CON     2   FFFF
NULL    0   0000
PTRP   13   0000
CR      4   0000
PR     62   0000
MAG1   85   0000
DSC1   C6   0000   WORK   PROT
DSC2   C7   0000   FIXD
DSC3   FC   0000   OFF
```

Figure 3-4. Display Devices Output Format

Possible error responses to DISPLAY DEVICES are:

FORM-ERR		Command syntax error; e.g., DIS DEV/PR:
PARM-ERR	MNEM	Invalid display option specified; e.g., DI DEVO
PARM-ERR	FD	Invalid File Descriptor specified; e.g., DI DEV,123
ASGN-ERR	type	Specified device/file not assigned for reason denoted by type.
IO-ERR	type	I/O error denoted by type encountered on Output Device or File

Magnetic Tape and File Control Commands

This set of commands allows the operator to manipulate Magnetic Tapes, Cassettes and Direct-Access-Files from the System Console. The general format of these commands is:

op[fd] [,lu]

The operator op may be any one of the following:

REWIND	Rewind
RW	Rewind (alternative operator)
FRECORD	Forward-space one record
FFILE	Forward-space to filemark
BRECORD	Backspace one record
BFILE	Backspace to filemark
WFILE	Write filemark

The mnemonics REWIND and RW are both accepted for the REWIND command, for compatibility with previous operating systems and certain utility programs.

The operand fd is the file descriptor of the device or file being positioned. LU is the Logical Unit number of the current task to which the device or file is assigned. Both are optional, but one or the other must be specified.

The operand fd need only be specified when positioning a device which is not currently assigned to any Logical Unit. If positioning a file, the file as assigned to the given Logical Unit of the currently selected task is manipulated. This is because the same file could be assigned to several tasks or to several Logical Units of the same task. The current position pointer of each of these assignments could be different. Only the selected File Control Block is modified. If no LU is specified, the first LU found (starting with LU0) is the one that is positioned.

For example:

```
REW MAG1:
```

rewinds device MAG1.

```
FR PACK:SOMEFILE.OBJ,4
```

causes the file PACK:SOMEFILE.OBJ, as assigned to Logical Unit 4 of the currently selected task, to be positioned forward one record.

```
FF ,2
```

causes the device or file assigned to LU2 to be forward-space to a filemark.

Possible error responses to these commands are:

FORM-ERR		Command syntax error; e.g., REW CR:/2
PARM-ERR	VAL	Invalid LU specified; e.g., REW ABC,40
PARM-ERR	FD	Invalid File Descriptor specified; e.g., BF 123
IO-ERR	type	I/O error denoted by type encountered on specified device or file
ASGN-ERR	type	File or device could not be assigned for the reason denoted by type
FUNC-ERR	TASK	No currently selected task; required for form (b) only
STAT-ERR	LU	LU is not assigned to specified file (form (b) only).

Initialize

This command is used to prepare and clear all files from a formatted Disc Cartridge for use under OS/16 MT2. Its format is:

INITIALIZE fd,voln [,READCHECK]

The operand fd specifies a Direct-Access Device which is Off-line to the system.

voln specifies the volume name, of one to four alphameric characters starting with an alphabetic, which is to be given to the volume mounted on the specified Direct-Access Device. If the optional operand, READCHECK, is specified, each sector on the volume is accessed to check for errors. If any sector were flagged as defective during formatting, READCHECK must be specified to avoid allocating a defective sector.

The initialization process consists of:

- Clearing the allocation map and setting as allocated:
  1. Volume Descriptor (sector 0, track 0, cylinder 0)
  2. Sectors required for the bit map.
  3. Sectors required for initial directory blocks
  4. Any defective sectors (if READCHECK specified)
- Allocating and initializing one cylinder of directory blocks (3 tracks on a 40MB Disc, 1 track on a 67MB, 256MB, Floppy Disc).
- Initializing the Volume Descriptor

Before initializing a Disc Cartridge, it must be formatted using the Common Disc Format program. Possible error responses to the INITIALIZE command are:

FORM-ERR		Command syntax error; e.g., INIT DSC1:/ABC
PARM-ERR	FD	Invalid File Descriptor specified; e.g., INIT ABC,XYZ
PARM-ERR	VAL	Invalid volume name specified; e.g., INIT DSC1:,1234A or fd does not specify a Disc Device.
PARM-ERR	MNEM	READCHECK not specified as third parameter; e.g., INITIAL DSC1:,A,X
PACK-ERR		Not enough good sectors to allocate allocation map and directory. Pack should be reformatted.
STAT-ERR	PRIV	Specified device not Off-line.
IO-ERR	type	I/O error denoted by type encountered during initialization process.

Save

This command is used to save a memory image load module of the operating system on a Direct-Access File suitable for loading by the OS/16 Boot Loader (see Chapter 1). Its format is:

SAVE fd

The operand fd specifies the file to be used to save the memory image load module of the system. If the file specified does not exist, a Contiguous File is allocated. An image of memory from location Zero to the last halfword before the start of the Background partition is written to the specified file.

This command may be used to save a copy of the operating system with patches for subsequent loads. The command may not be used on an Overlayed System. On a system with DELETE INITIAL, SAVE must only be used before any tasks are loaded.

Possible error responses to SAVE are:

FORM-ERR		Invalid syntax; e.g., SAVE OS16MT2.000,R
PARM-ERR	FD	Invalid File Descriptor specified; e.g., SAV 123
ALLO-ERR	type	Allocate error denoted by type encountered
ASGN-ERR	type	Error denoted by type encountered assigning specified file
IO-ERR	type	I/O error denoted by type encountered Writing OS image to file.

Print

The PRINT command is valid only in systems SYSGENed with the SPOOL option. It causes the OS to send a message to the SPOOLER, requesting that the specified file be printed. If the SPOOLER successfully receives the message, the file is placed on the print queue.

The format of this command is:

PRINT [fd[,n][,D][,T]]

where:

fd is the file to be printed.
n is a decimal number between 1 and 9 specifying the number of copies of the file to be printed.
D is specified if the file should be deleted after printing.
T is specified if the file should be placed at the top of the print queue.
If no parameters are entered, the names of the print files currently in the print queue are displayed on the console.

Examples:

```
PRINT   PCB.CAL
PRINT   PCB.CAL,2
PRINT   TEMP,D
```

Possible error responses to this command are:

MNEM-ERR	Spooling feature not in system.
FORM-ERR	Command syntax error; e.g., PRI P*B.CAL
FD-ERR	Invalid file description; e.g., PRI 1ABC:X.Y/P
SEND-ERR PRES	Spooler not loaded
SEND-ERR NMSG	Spooler did not receive message

COMMAND SUBSTITUTION SYSTEM (CSS)

The Command Substitution System (CSS) is an extension to the OS/16 MT2 Command Language. It provides the user with the ability to establish files of commands which can be called from the console or other CSS Files and executed in a defined sequence. In this way, complex operations can be carried out by the operator with only a small number of commands.

CSS provides more than just the ability to switch the operating system command input stream to a batch device:

- A set of logical operators are provided to control the precise sequence of commands to be obeyed.
- Parameters can be passed to a CSS file so that general sequences can be written which take on specific meaning only when the parameters are substituted.
- One CSS file can call another, in the manner of a subroutine, so that the experienced user can develop complex command sequences.

A CSS file is simply a sequential text file. It could be a Deck of Cards, a Punched Paper Tape, a Magnetic Tape, or a Disc File. On Disc, a CSS file is best suited to the Indexed File structure.

This is an example of a simple CSS file:

```
*THIS IS A SIMPLE EXAMPLE CSS FILE
TASK .BG
ALLOCATE XXXDIS.TST,IN
ASSIGN 2,PRT1:;*LU2=LINEPRINTER
LDBG COPY
START
$EXIT
```

Note the use of the semicolon, which allows more than one command on the same line. Note also the use of the asterisk to introduce a comment.

High Level Operator Command Package

The OS/16 MT2, High Level Operator Command Package is implemented as a set of CSS files. These perform a variety of the commonly used program preparation and development sequences. The package contains the following:

COMMAND	DESCRIPTION
FORT	Perform FORTRAN compile, and task establishment
FORTCLG	Perform FORTRAN compile, task establishment, load and start
CAL	Perform CAL/16 assembly
CALCLG	Perform CAL/16 assembly, load and start
MAC	Perform CAL MACRO expansion, and assembly
MACCLG	Perform CAL MACRO expansion, assembly, load and start
EDIT	Edit a file using OS Edit
ESTAB	Establish an object program using the OS/16 Task Establisher (TET/16)
COPYA	Copy an ASCII file using OS Copy
COPYB	Copy a Binary File using OS Copy
COPYT	Copy an established task, resident library or overlay using OS Copy
RUN	Load and start a task
SYSGEN	Generate an OS/16 MT2 operating system
DEFAULT.ASN	Assign default Logical Units for a task

For complete information refer to Chapter 14.

The following sections define the use of CSS.

Calling CSS Files

A CSS File is called and executed by naming it in a stream of commands. Any valid File Descriptor (fd) can be used, provided that there is no conflict with any of the ordinary operator commands. If the file extension is omitted an extension of CSS is assumed. The CSS call is the last command recognized on a command line.

In other words, the operator can cause a file of commands to be executed simply by entering the name (fd) of the CSS file. The error message, MNEM-ERR, is returned if the file does not exist as specified.

Parameters are passed to a CSS file by appending them to the call. The first parameter is separated from the file name by a space; all other parameters must be separated by commas. Null parameters are permitted. Leading blanks are suppressed when parameters are passed.

The following are valid CSS calls:

```
RUN              (Calls CSS file RUN.CSS)
CR:              (Calls CSS file in Card Reader)
JUMP A,B,C       (Calls CSS file JUMP.CSS with three parameters A, B, and C)
JUMP.CSS A,B,C,  (Same as above)
JUMP ,,C         (Calls CSS file JUMP.CSS with three parameters, the first two of which are null).
```

Use of Parameters

Within a CSS File a parameter to that file is referenced by means of the special symbol '@'. The first parameter is referenced by @1, the second by @2, etc. A straight-forward text substitution is employed.

Thus, a CSS File RUN might consist of:

```
TASK    .BG
LDBG    @1
START   @3,@2
etc.
```

This would then be called:

```
RUN  PROGRAM,NOLIST,A148
```

Before each command of the CSS File is decoded, it is pre-processed, and any reference to the parameter is substituted with the text of the parameter. Thus, the file RUN with the previous call would be executed as:

```
TASK    .BG
LDBG    PROGRAM
START   A148,NOLIST
etc.
```

In general a reference to a parameter is of the form:

@n

where n is a decimal number indicating which parameter argument the user is referencing. Arguments are numbered starting with 1. Argument 0 is a special argument, and is defined in the last 2 paragraphs in this section.

Being a decimal number, a reference variable is terminated by a non-decimal character. For example, to reference variable 12,

@12 or @12ABC or @12.EXT

are valid expressions.

Notice that this mechanism allows concatenation. For instance, if in the above file, RUN, the second command were

```
LOAD .BG,@1.TSK
```

then only files containing established tasks would be presented to the loader.

Concatenation of numbers requires care. 123@1 is permitted and would expand correctly, but @1123 is a reference to parameter number 1123.

A reference to a non-existent parameter is considered to be null.

Parameter @0 is a special parameter. It is used to reference the name of the CSS File in which it is contained. Parameter @0 is replaced, during the pre-processing of the command line, with the name of the File Descriptor in precisely the style used to call the file.

This mechanism can be used to assign the CSS File itself to an LU of a program. By this means the data for a program can be included in the CSS File itself. However, the program must read precisely the right number of data items or else subsequent CSS processing may fail. This is only valid for Non-Direct-Access Files, since assigning a file would position the file to the beginning as far as the task was concerned.

Interaction of CSS with Background and Foreground

CSS is essentially a Single-Stream Processor. It is not possible, in the general case, to write a CSS File that can fully control a complex Foreground/Background System. In order to control such a system, manual intervention by the console operator is usually required.

It is assumed that Foreground Systems are controlled, under normal circumstances, by Supervisor Calls (SVC 6) between the Foreground tasks, and that the console operator is only required to intervene in abnormal cases. The Background System, however, is expected to be controlled fairly often by the operator, or by CSS Files (if the background is being run in a batch-like mode).

In batch mode, CSS control is only desirable between tasks; job-control commands are not usually desired while the tasks are running. Therefore, CSS is "keyed" to the state of the background task. While a task in the background partition is in any state other than dormant, the active CSS file is not accessed. If a START command in a CSS file is directed towards a background task, the rest of the commands on the command line are executed but no new CSS command line is read until the background task either goes to EOT or is cancelled. The console operator has full control over the system at these times. While the Background task is Dormant, Command lines are read from active CSS files.

The state of Foreground tasks has no affect on CSS activity.

While CSS is active, the operator is still able to enter commands from the System Console. The execution of these commands may be delayed while a CSS command is being executed; however, this delay should not be excessive under normal circumstances.

If a CSS File is active, any attempt to call another CSS File from the System Console is rejected.

COMMANDS EXECUTABLE FROM A CSS FILE

All commands normally available to the operator at the console can be used in a CSS File, as well as a number of commands specifically associated with the CSS facility. These additional commands are described next.

Most of the CSS commands start with the character $. If a log of commands is being kept, the $s help to emphasize where CSS has been used, but the $ has no special meaning.

CSS Files are permitted to affect Foreground tasks; a TASK command read from a CSS File establishes the currently-selected CSS task. Commands from the console are not affected by TASK commands read from CSS, and CSS is not affected by TASK commands read from the console. All task related commands and CSS return code testing encountered in a CSS File affect the currently-selected CSS task.

When a CSS File is activated from the console, the currently-selected CSS task is set equal to the currently-selected task. If the currently-selected CSS task is deleted from the system, any subsequent task-related or CSS return code testing commands are rejected with a FUNC-ERR TASK

$EXIT and $CLEAR

These two commands are provided for exiting from CSS files. Command $EXIT causes control to return to where the CSS file was called. Control returns to a higher level CSS file. CSS processing terminates if there is no higher level CSS file.

Command $CLEAR causes CSS processing to terminate unconditionally. This command may be entered at the system console to abort an active CSS File at any time.

$JOB and $TERMJOB

These commands delimit a CSS job. The CSS job concept is a defensive mechanism which protects one user from the errors of a previous user. A CSS job consists of all the operator commands and tasks loaded and started between a $JOB and $TERMJOB pair. The $JOB command delimits the start of a CSS job. The Return Code of the currently selected CSS task is reset to Zero.

The $TERMJOB Command delimits the end of a CSS job. Most errors encountered in executing operator commands in a CSS job cause the remaining statements in the CSS File to be skipped until a $TERMJOB is encountered.

By separating independent users into CSS jobs delimited by $JOB and $TERMJOB statements they can be safely batched. in a CSS File, eliminating the chance that errors in one job may affect another job.

Logical Operators

There are ten logical operators available. All logical operators start with the three characters $IF and allow one argument (e.g., $IFE 255, $IFX B.CSS, $IFNULL @1).

Each logical statement establishes a condition which is tested by the CSS Processor. If the result of this test is 'true', then commands up to a corresponding $ELSE or $ENDC command are executed. If the test gives 'false' result these same commands are skipped.

The $ENDC command delimits the range of a logical operator; however, nesting is permitted, so each $IF must have a corresponding $ENDC.

Command $ELSE reverses the effect of the preceding $IF.

In Figure 3-5, the ranges of the various conditionals are indicated by brackets.

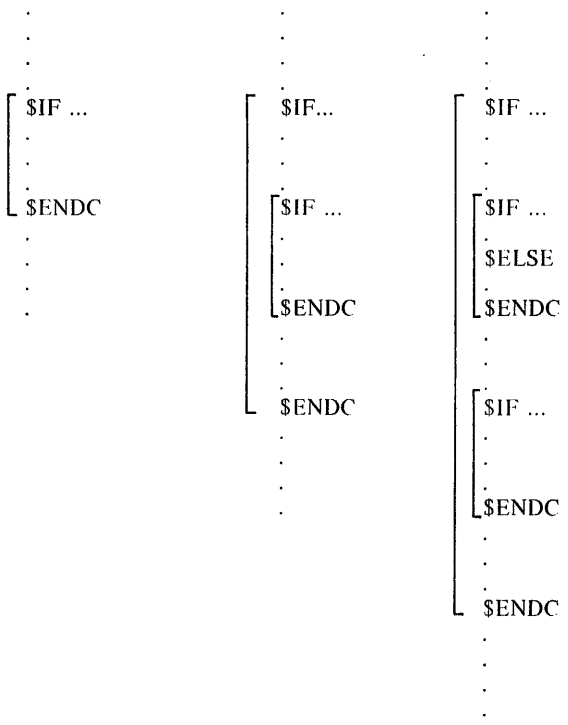


Figure 3-5. Range of CSS Conditionals

There is no practical restriction on the depth of nesting.

The logical operators fall into three categories: Return Code testing, File Existence testing, and Parameter Existence testing.

Return Code Testing

The Return Code is a halfword quantity maintained by the system (see the description of SVC 3 in the *OS/16 MT2 Programmer's Reference Manual*) for each partition.

It is set in any of the following ways:

SET CODE n – This command, which can be included in a CSS File or entered at the console, sets the Return Code for the currently-selected CSS task to n.

$JOB – As part of its start job function, this command resets the Return Code for the currently-selected CSS task to Zero.

Command Error – Any command error causes the CSS mechanism to skip to $TERMJOB (assuming that a $JOB has been executed; if not, control returns to the console). To indicate that the skip has taken place, the Return Code for the currently-selected CSS task is set to 255.

$SKIP – This command has the same affect as a command error.

EOT (SVC 3,n) – When any task terminates by executing the EOT program command (SVC 3,n) the Return Code for that task is set to n.

CANCEL – When a task is CANCELed, the Return Code for that task is set to 255.

There are six commands available for testing the return code of the currently selected CSS task:

$IFE	n	Test Return Code equal to n
$IFNE	n	Test Return Code not equal to n
$IFL	n	Test Return Code less than n
$IFNL	n	Test Return Code not less than n
$IFG	n	Test Return Code greater than n
$IFNG	n	Test Return Code not greater than n

In all cases if the test gives a 'false' result, CSS skips commands until the corresponding $ELSE or $ENDC. If such skipping attempts to skip beyond a $TERMJOB or End of File, a command error is given.

File Existence Testing

There are two commands concerned with the existence of files:

$IFX	fd	Test fd for existence
$IFNX	fd	Test fd nonexistence

If the test gives a 'false' result, CSS skips to the corresponding $ELSE or $ENDC. The previous restriction on skipping also applies.

Parameter Existence Testing

There are two commands concerned with the existence of parameters:

$IFNULL	@n	Test @n null
$IFNNULL	@n	Test @n not null

If the test gives a 'false' result, CSS skips to the corresponding $ELSE or $ENDC, with the same restriction as previously stated.

In addition, a combination of parameters can be simultaneously tested. For example,

```
$IFNULL @1@2@3
```

In effect, this tests the logical OR of @1,@2, and @3 for null. If any of the three is present, the test results in 'false'.

Listing Directives

Two commands are provided to control the listing of CSS files as they are executed:

```
$COPY
$NOCOPY
```

The $COPY Command causes subsequent commands to be listed, in their expanded form, after parameter substitution. The listing takes place on the console or the log device, according to the options selected in a previous SET LOG command.

The $NOCOPY Command switches off the listing (the $NOCOPY statement is logged). The default is $NOCOPY.

CSS File Construction

There are two command pairs provided for construction of CSS Files:

BUILD, ENDB, and $BUILD, $ENDB

BUILD and ENDB

The BUILD command causes succeeding command lines to be copied to a specified file, up to but excluding the corresponding ENDB command. The format of the BUILD command is:

```
BUILD fd
```

where fd is the new CSS File. If fd does not already exist, it is created. (An Indexed File is allocated with a logical record length equal to the command buffer length specified at System Generation, data and index blocksize of 1, and keys of 0000).

BUILD can be issued from the console or from within a CSS File. No nesting of BUILD commands is possible. The processing of BUILD ends when the first ENDB command is encountered, so any attempt to nest BUILD commands results in a corrupt CSS file being constructed.

The BUILD command must be the last command on its input line. Any further information on the line is treated as comment and is not copied to the new CSS file.

The ENDB command must be the only command on a line, and must occupy the first four character positions on the line. Any further information on the line is treated as comment and is ignored.

A $BUILD. . . .$ENDB sequence can be nested inside a BUILD. . . .ENDB pair.

$BUILD and $ENDB

These commands operate in a similar manner to BUILD and ENDB, except that before each line is copied to the CSS File, the CSS Pre-Processor substitutes any parameters in the line. It follows that $BUILD is only used from within a CSS File so that parameters can be passed to it. The $BUILD command has the following format:

$BUILD fd

where fd is the new CSS File. If fd does not already exist, it is created, (as with BUILD).

As with BUILD, no nesting of $BUILD is possible. A corrupt CSS File results if the attempt is made.

$BUILD must be the last command on its input line, any further information is treated as comment and is ignored.

$ENDB must be the only command on its input line, and it must occupy the first five character positions on the line. Any further information is treated as comment and is ignored.

A BUILD. . . .ENDB sequence can be nested within a $BUILD. . . .$ENDB pair.

TABLE 3-7. CSS COMMAND SUMMARY

Command	Meaning
$JOB	Start next job, reset Return Code
$TERMJOB	End of Job, any error skip in last job stops at this command with Return Code = 255, otherwise, Return Code is defined by the job itself.
$EXIT	Exit from CSS File.
$CLEAR	Return control to console.
SET CODE n	Set Return Code to N.
$IFE n	If Return Code equals n, continue executing commands, otherwise skip to corresponding $ENDC, $ELSE.
$IFNE n	If Return Code not equal to n, continue executing commands, otherwise skip corresponding $ENDC, $ELSE.
$IFL n	If Return Code less than n, continue executing commands, otherwise skip to corresponding $ENDC, $ELSE.
$IFNL n	If Return Code not less than n, continue executing commands, otherwise skip to corresponding $ENDC, $ELSE.
$IFG n	If Return Code greater than n, continue executing commands, otherwise skip to corresponding $ENDC, $ELSE.
$IFNG n	If Return Code not greater than n, continue executing commands, otherwise skip to corresponding $ENDC, $ELSE.

TABLE 3-7. (Continued)

Command	Meaning
$IFX fd	If fd exists, continue executing commands, otherwise, skip to corresponding $ENDC, $ELSE.
$IFNX fd	If fd does not exist, continue executing commands, otherwise skip to corresponding $ENDC, $ELSE.
$IFNULL @n	If parameter does not exist, continue executing commands, otherwise skip to corresponding $ENDC, $ELSE.
$IFNNULL @n	If parameter exists, continue executing commands, otherwise skip to corresponding $ENDC, $ELSE.
$ELSE	Reverses action of previous conditional.
$ENDC	Delimits above conditionals.
$COPY	Switch-on listing.
$NOCOPY	Switch-off listing.
$BUILD fd	Construct CSS File with parameter substitution.
$ENDB	End of $BUILD.
BUILD fd	Construct CSS File without parameter substitution.
ENDB	End of BUILD.
$SKIP	Skip to $TERMJOB.

CHAPTER 4
SYSTEM ERROR HANDLING

ERROR TYPES

There are three kinds of error conditions that can occur during operation of OS/16 MT2: recoverable errors, task crashes, and system crashes.

A recoverable error occurs when the system detects a condition which requires operator intervention in order to continue. In this case, the system prints a message describing the condition, (e.g., Command syntax error). The operator may then issue commands to correct the error.

A task fault occurs when a malfunction is detected during the execution of a user task and the system cannot continue executing the task without destroying the system or user information. At this point, the system prints an error message which describes the error detected (e.g., Illegal Instruction, Illegal SVC) and the location of the instruction causing the error. The user task is PAUSEd. The operator may correct the error condition and use the CONTINUE command to proceed. The task may be aborted with a CANCEL command.

A system crash occurs when a hardware or software malfunction is detected during execution of system code and the system cannot proceed without running the risk of destroying information, either on some peripheral or in memory. At this point, the system attempts to display a crash code on the display panel or ASCII console and to write the crash code in dedicated memory. It then stops. System crash codes are given in Appendix 4.

SYSTEM CRASH RECOVERY

The System Crash Handler is designed to minimize destruction of user data resulting from system error and to preserve the state of the system so that the error can be fixed. Recovery procedures for system crash are given below with the main intent of preserving the information necessary to find and fix the error. The only positive action that should be taken is to run the Disc Integrity Check Utility (03-080) if any files were assigned at the time of the crash.

At the time of a system crash:

- –the crash code is displayed on the Console Display Panel or ASCII console
- –register 5 contains a pointer to the System Journal
- –register 6 contains a pointer to the last entry in the System Journal
- –the registers at the time of the crash are stored at Entry CRSHSV

On encountering a system crash, the crash code, contents of the registers, the contents of the System Journal and the sequence of operator commands leading up to the crash should be recorded. See Appendix 4 for the definition of system crash codes.

The system journal is a record of past events. Important events, such as scheduling, queue termination interrupts and I/O termination are entered. An entry consists of five halfwords, the journal code followed by the contents of registers R12-R15. For a description of the journal codes, refer to Appendix 12.

System error conditions caused by machine malfunction interrupts occur for memory parity error, primary power failure, and power restoration.

A memory parity error causes a system crash if it occurs in the system or a task crash if it occurs in the task.

POWER FAIL/RESTORE

When a power fail is detected by the Processor an interrupt occurs and OS/16 MT2 saves the registers and prepares itself for another interrupt upon power restoration.

A system message

PWR RSTR – RESET DEV

appears on the Console Device (if any) when power has been restored. After the message is printed, operator intervention is required to reset any devices that may be in an Off-line or Write-protected state as a result of the power failure. The operator may then continue the power restoration process by depressing EXECUTE or RUN on the Console Display Panel.

All non-direct access I/O operations, in progress when power failed, are aborted and an error status is returned to the task. Direct-Access I/O operations are retried when power is restored.

The power restore module schedules power restore traps for those tasks which require knowledge of the power failure. It is the user's responsibility to see that tasks which require such knowledge have the Power Restoration Trap Enable Bit set in the current TSW. When there is a power outage, a TSW swap occurs upon restoration of power and the task traps to a module to recover from the power outage. All other tasks are paused. On systems having a console device (i.e. configured with the Command Processor module) a message is output:

TASKID: PAUSE

Upon CONTINUE, return is made to normal processing.

If the OS console device or a resource-sharing user terminal device is powered down or turned off-line, the device will become "not-available" to the system after a ten second retry period. If the device is restored before ten seconds, it will again become available for normal operation. The BROADCAST command may be used to wake-up a "not-available" user terminal.

CHAPTER 5
TASK ESTABLISHMENT

INTRODUCTION

This chapter describes TET/16, the OS/16 MT2 Operating System Task Establisher and provides examples of its use.

A task may be a single program or a group of programs linked together (see Figure 5-1). TET processes object format programs, links their external references and produces a memory-image task for loading and running under OS/16 MT2. External references to Task Common and to Reentrant Library segments are also processed.

A task is established by a single execution of TET, resulting in one or more Load Modules of absolute memory image code. An established task consists of a main segment with optional overlays. The task and each overlay may be built on its own separate Disc File. Task Common and a Reentrant Library segment may be referenced by the user task.

The establishment procedure is a two-pass process, requiring two inputs of the same program stream. On the first pass, TET compiles a symbol table of external references and definitions. On the second pass, the actual load module is built. A scratch file can be used to save the input of pass one for input to pass two.

TET/16 is also used to build a boot loadable OS/16 MT2 system onto a Contiguous Disc File.

The command stream that directs TET's actions can be input in batch mode or interactively. An operator uses the commands to specify programs for inclusion in the task, as well as task options.

CONFIGURATION REQUIREMENTS

TET/16 requires 12KB of memory space for its code, plus approximately 1.5KB for dynamic operations and as much space as is required to contain a dictionary of all external references and definitions in the programs of the task being established. TET may build task modules using a Contiguous Disc File as work storage or build the task in memory, and then write it to a file or to a device. If the task is built in memory, there also must be enough space to hold the largest load module built.

Required devices include input and output binary devices for the input object code and output image code, an ASCII device for TET command input, and an ASCII print device for error, warning, and prompt messages to the operator. ASCII device requirements can be met by multiple assignment of a CRT or TTY, but high speed devices are recommended for binary input and output. A scratch file to hold pass one input for pass two is a recommended option. A disc to build on is also recommended, since building in memory limits the size of the task that can be built.

TET/16 may also be run under OS/32. A 32-bit object of TET is provided with the package.

SYSTEM ENVIRONMENT

The object programs for inclusion in a task can be input from any number of files. A task can include multiple overlays, Task Common and Reentrant Library references. Certain run-time task conditions can be preset by TET. These include the task's priority, its initial task status, and whether it is resident and/or rollable.

A task is established for a given partition (Background or Foreground). The start address is specified at establishment time. If the task references Task Common or a Reentrant Library, the start addresses for those partitions must also be given.

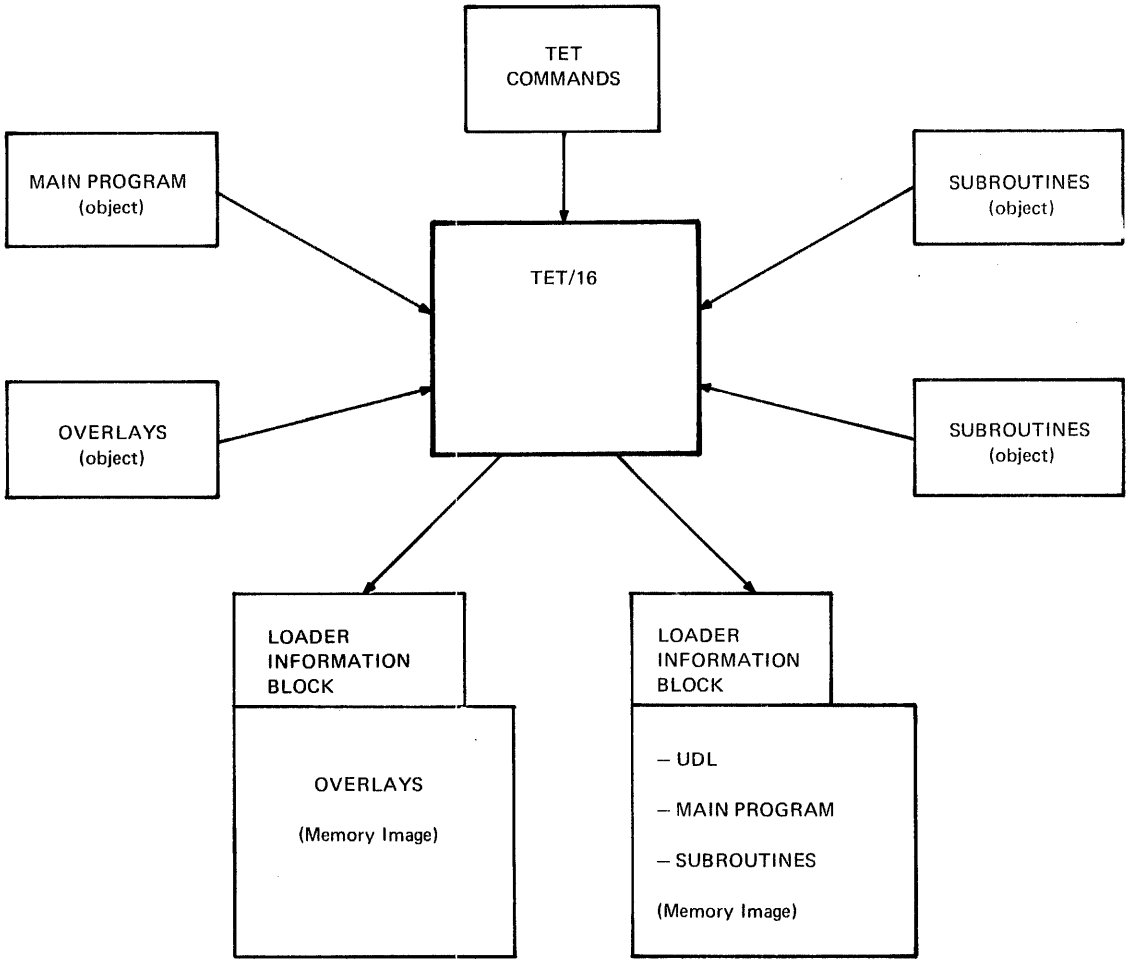


Figure 5-1. Task Establishment

A task can have one or more overlay areas, positioned above the root, within the task's partition. The size of each area is determined by the largest overlay to use that area. Overlays for a task can be built sequentially onto a single file, or onto individual files. Any local common block defined in one or more overlays but not defined in the root is positioned in the first overlay to define it. Overlays which use the same overlay area must not call one another.

Two types of common are supported: local common and task common. Local common is contained entirely within a task's own segment. It is referenced by the main program and the subroutines of a task (e.g., FORTRAN common). TET allocates space for local common as defined by the included programs. Task common is a special segment which allows common references between tasks. It is symbolically referenced by a task by the common block label TSKCOM. The Task Common block itself must be established by TET/16 if it is to be loaded into its own partition.

By default, TET/16 initializes the User Dedicated Locations (UDL) for a task to zero, and loads relocatable code immediately following the UDL. The user may specify a bias for the first relocatable program to be included, and may load absolute code anywhere above the start of the partition.

The handling of command errors and I/O device problems depends upon the mode of operation:

Interactive – TET defaults to interactive mode for a command device having the attribute interactive. This normally covers all TTY and CRT type devices. In this mode, an illegal command causes an error message but is otherwise ignored. TET pauses and waits for operator intervention when I/O device errors or repositioning problems occur.

Batch – TET defaults to batch mode for a command input device having the attribute non-interactive. In this mode, an illegal command causes TET to abort since the next command does not take account of the error. I/O device problems requiring operator intervention are treated as for interactive mode.

Remote – TET aborts on any unrecoverable error, assuming no operator to be present. TET should be run in this mode when run under CSS control.

A TET/16 load module consists of a Loader Information Block (LIB) followed by the absolute memory image of the task in 256-byte records. The LIB is a 20-byte record containing data pertaining to the load module required by the OS.

TET/16 COMMANDS

TET/16 commands are followed by one or more blanks followed by optional parameters. Parameters are separated by commas, and the parameter string may be terminated by a blank or a carriage return.

The valid TET/16 operator commands are:

INTERACTIVE – Requests interactive mode of operation.

BATCH – Requests batch mode of operation.

REMOTE – Requests remote mode of operation. This command should be used when running under CSS or any unattended batch-mode system.

ESTABLISH TASK
RL
TCOM
OS

This required command begins and initializes TET/16 processing. Its parameter specifies whether a Task segment, Reentrant Library segment, Task Common segment or Operating System is to be established.

BIAS[pstart [,rstart]]

This command specifies the start address of the partition for which a task, Reentrant Library or Task Common segment is to be established.

Parameter pstart is the address and is a hexadecimal number of up to five digits. If omitted, the bias defaults to the start of the partition in which TET/16 is currently running. In an extended memory system the default option should not be used unless TET is running in the background partition.

This command must be given after an ESTABLISH TA,RL or TC command and before the first INCLUDE command. It is not valid for ESTABLISH OS.

The optional parameter rstart is only valid when establishing a task. It allows the user to override the default bias for relocatable code, which is normally set immediately above the UDL. Parameter rstart is a hexadecimal address of up to five digits, and must be greater than or equal to pstart.

TSKCOM tstart

To resolve references to a Task Common block, a TSKCOM command must be entered.

Parameter tstart is the start address of the Task Common partition, as a hexadecimal number of up to 4 digits.

It must be entered after the ESTABLISH TA command and before the first INCLUDE command.

The established task's LIB notes that Task Common is required for the task to run. The task aborts during loading if the required Task Common is not present.

This command is only valid for Task establishment or Reentrant Library establishment.

OPTIONS opt [,opt] ...

The OPTIONS command defines one or more of the optional OS parameters associated with a task, as defined by Table 5-1.

TABLE 5-1. TET OPTIONS COMMAND

OPTION	MEANING
UT	User Task (default)
ET	Executive Task
NOFLOAT	No Single Precision Floating Point (default)
FLOAT	Single Precision Floating Point
NODFLOAT	No Double Precision Floating Point (default)
DFLOAT	Double Precision Floating Point
NONRES	Not Memory Resident (default)
RES	Memory Resident
AFP	Arithmetic Fault Pause (default)
AFC	Arithmetic Fault Continue
SVCP	Illegal SVC 6 Pause (default)
SVCC	Illegal SVC 6 Continue
CSEG	Task Common or Sharable Library in the second 32KB module
NOCSEG	No Task Common or Sharable Library in the second 32KB modules (default)
NOROLL	Task Not Rollable (default)
ROLL	Task Rollable
UDL	Task needs UDL (default)
NOUDL	Task contains UDL
NOCOMP	Task uses OS/16 MT2 SVC 1 format (default)
COMP	Task uses BOSS/DOS/RTOS SVC 1 format

This command is optional, but if entered, it must follow the ESTABLISH command and precede any INCLUDE command.

The option information halfword is placed in the task's LIB, and eventually into the Task Control Block (TCB) at load time.

The option mnemonics are processed in order, from left to right. For contradictory options, the last one entered is taken. If an illegal mnemonic is found, all options up to that point have been processed. Those following are ignored.

The UDL option requests TET/16 to clear the UDL for the task, and bias relocatable code immediately above it. The UDL provided includes the Floating Point Register save area/s if the FLOAT and/or DFLOAT options are used.

The NOUDL option requests TET/16 to bias relocatable code from the start of the partition. In this case, the user must include the UDL as part of the program.

Note that if a task uses floating point arithmetic, OPT FLOAT and/or OPT DFLOAT as appropriate must be specified when the task is established so that the UDL includes the register save area/s.

CSEG is used in extended memory systems to indicate that Task Common and Reentrant Library routines are in the second 32KB module. See Chapter 6 of the *OS/16 MT2 System Planning and Configuration Guide,* Publication Number 29-431, for details on the CSEG option and examples of extended memory systems.

INCLUDE [fd] [,program label]

To include programs from a file or device as part of the load module, an INCLUDE command is required. The optional parameter fd specifies a File Descriptor (VOL:FILENAME.EXT). If given, the file or device is assigned for input. Note that disc files will be positioned at the start. If fd is not specified, the file or device currently assigned to Logical Unit 1 is assumed as the input device. In this case, no repositioning occurs. The optional program label parameter names a program in the Input File. This command selects from the named file, one program or the entire file of programs for inclusion in the task being established. The optional program label parameter causes only the specified program to be located and included, leaving the file or device positioned at the end of that program. The dummy program label '*' may be used to include one program from the current position. If a program label is not specified, the entire file is included up to end of medium or end of file. The optional program label parameter is not valid for RL establishment.

As the input file is read, TET creates a dictionary of external program references. A copy of each included program or programs is written to the Scratch File, if present.

EDIT [fd]

After inclusion of one or more programs, it is possible to edit a file. EDIT includes all programs having program labels referenced by the already-included code. The optional operand, fd, names a file to be edited from, defaulting to the file currently assigned to LU 1.

When TET edits a file, it copies the entire file to the Scratch Device (if one is present) and notes those programs required to build the load module. During Pass Two, those not required are skipped.

This command can be used where a task requires a number of subroutines from a subroutine library. It is only valid for Task Establishment. The edit process terminates when an end-of-medium, end-of-file or the label "ENDVOL" is encountered.

RESOLVE [fd] , pstart [,rstart]

To resolve external references to a Reentrant Library routine (such as a FORTRAN Run-Time Library), a RESOLVE command must be used. The RESOLVE is used after the program referring to the RL has been included with an INCLUDE or EDIT command. This command resolves all references for which ENTRY points can be found in the Reentrant Library object module. The optional operand, fd, specifies the file on which the Reentrant Library object module is to be found (defaulting to LU 1). pstart is the start address of the Reentrant Library partition, specified as a hexadecimal number of one to four digits.

The task's LIB specifies that the Reentrant Library Segment is required for the task to run. An attempt to load the task is aborted if its required Reentrant Library segment is not present.

During the RESOLVE, no programs are copied to the scratch since none are actually included as part of the task.

The optional parameter rstart allows the user to override the default bias for relocatable code. It is used only for extended memory application. See the *OS/16 MT2 System Planning and Configuration Guide,* Chapter 6.

GET xxxx

This command is used to provide space for GET STORAGE calls (SVC 2,2). xxxx is a hexadecimal number of up to 4 digits specifying the number of bytes to be reserved for GET STORAGE calls. If this command is not specified, a default value of X'400' bytes (the amount necessary for an executing FORTRAN program) is assumed; this command can be given with a parameter of Zero to save space if no GET STORAGE calls are to be issued.

OVERLAY name [,NEW]

An OVERLAY statement is used to indicate that an overlay for a task being established is about to be included.

The parameter name is the name of the overlay as referenced in the fetch overlay SVC parameter block of the calling program. It must have up to 6 alphanumeric characters, with the first character alphabet.

The optional parameter NEW specifies that a new overlay area is to be set up for this and subsequent overlays.

TET interprets this statement as ending the definition of a main segment and starting the definition of an overlay. Each overlay must be completely defined (with INCLUDE and EDIT statements) before another OVERLAY statement is presented in the command stream. All overlays to share one area must be processed before any overlays for another area. After all overlays have been defined for one overlay area, the area is set to the size of the largest.

The OVERLAY command must precede the INCLUDE and EDIT commands that define its contents, and these must precede any other OVERLAY statement or the BUILD command that terminates all pass 1 processing.

PRIORITY rp,mp

This optional command sets the initial and maximum priorities for the task. Where rp is the initial priority, mp is the maximum priority. Both rp and mp must be decimal numbers between 10 and 249 for user tasks and between 0 and 255 for executive tasks, with mp less than or equal to rp.

If omitted, a default value of 128 is assumed for both parameters. This command may be given during pass one, after an ESTABLISH TA command.

TSW status [,start address]

This command specifies initial settings of the task's TSW status, and optionally provides a start address for the task. The TSW (Task Status Word) status is defined in Chapter 2 of the *OS/16 MT2 Programmer's Reference Manual.* The status parameter and the optional start address are one to four-digit hexadecimal numbers. If no start address is given, the last transfer address to be found in the object module is used as task start address. If none is found, the address of the first location above the UDL is taken by default. This command may be given during pass one, after an ESTABLISH TA command.

BUILD { TASK / RL / TCOM / OVLY / OS } [,fd]

After the first pass definition of the segment is complete, the Load Module must be built. A BUILD command signals the end of pass one and the beginning of pass two. The required first parameter specifies the type of Load Module to be built. The optional File Descriptor specifies a file onto which the Load Module is to be output. File Descriptor fd defaults to the file or device currently assigned to Logical Unit 2. If the file specified by fd or assigned to Logical Unit 2 is a Contiguous Disc File, TET builds the load module directly on that file. If the file is not a Contiguous Disc File, TET builds in memory and outputs the load module to the file. If the form of the fd parameter is valid for a Disc File, but the file does not exist, TET/16 attempts to allocate and assign a Contiguous File of the correct length, to the nearest 1024 bytes. If building on disc, TET rewinds the Disc File before the build operation. If building in memory, TET outputs the resultant load module to the specified file or device with no repositioning.

The BUILD command reads the Input Program File again in the same order as the first pass. If a Scratch File is being used, it is rewound by the system if possible, otherwise, the operator is prompted to reposition the device. If a Scratch File is not being used, the operator is prompted to reposition each input in sequence. Each fd input during PASS 1 is displayed and TET pauses. The user should reposition the fd if necessary and enter CONTINUE.

Specifying TASK as the first parameter results in building one Load Module for a task (or the main segment of a task with overlays), and outputting it to the specified fd. This segment contains all absolute code, local common positioned in the main segment, and all programs included in the segment during pass one.

Specifying RL or TCOM results in building one Load Module for a Reentrant Library or Task Common Segment.

Specifying OVLY results in an absolute overlay segment being output to the file. For each OVERLAY command entered during pass 1, a BUILD OVLY command must be entered in corresponding order during pass 2. Overlay segments can be built on one file or separate files, and are ready for loading at execution time by the main segment via Fetch Overlay (SVC 5) calls.

Specifying OS results in building a memory image load module of OS/16 MT2 onto the specified file or device, with or without overlays, depending upon the user specified system parameters.

If fd does not specify a Contiguous Disc File, and the OS is to be overlayed, the command is rejected. In order to load the Operating System with the Bootstrap Loader, the fd must be of the form:

VOLN:OS16XXXX.NNN

where:
- VOLN = Volume name of up to 4 characters
- OS16 = Mandatory first 4 characters of filename
- XXXX = Up to 4 optional characters
- NNN = Three hexadecimal digits as the extension (OS identifier)

VOLUME voln

This command is used to specify a default volume for TET commands. If omitted, the system default volume is used. It can be entered more than once, if desired, and takes effect immediately. voln is any ligitimate volume name of one to four characters.

This command can appear anywhere in the sequence, including prior to ESTABLISH. Whenever a File Descriptor (FILENAME.EXT) that does not specify a volume name is encountered, voln is used.

MAP [fd]

The MAP command requests TET to display the contents of the dictionary to the device specified by fd, defaulting to the file or device currently assigned to Logical Unit 3 if fd is not specified. The value of the resultant map is dependent upon when the command is issued. During pass 1, a list of undefined symbols is valuable in determining which programs are yet required. The MAP is most useful at the end of pass two when the dictionary is complete. The items are output in address order.

An explanation of each item in a TET map is listed below. Individual headings are not printed unless there is an item to be printed under that heading. A sample map is illustrated in Figure 5-2, 5-3, 5-4.

PARTN	Output in the header portion of the map, indicates the physical memory location of the task.
MEMORY CONSTANTS:	
UBOT	Start address of the task, RL or task common block being established.
BIAS	Bias taken for relocatable code
SIZE	Address of first location above the image to be loaded. For a task with overlays, this is the start of the overlay area.
UTOP	Address of first location in the GET Storage area if any.
CTOP	Minimum acceptable address for the last halfword location in the user's required memory partition.
PROGRAM LABELS:	An entry in this list is a 4 digit hexadecimal address followed by a 6 character label. Each entry represents a program label and its start location in the user's space.
ENTRY POINTS:	An entry in this list is a 4 digit hexadecimal address and its associated symbolic name. It is a list of all definition addresses in TET's dictionary.
UNDEFINED SYMBOLS:	This a list of all external references for which no definitions have been encountered.
COMMON BLOCKS:	This section lists the components of the local common area of the user's task. An entry is a 4 digit hexadecimal address field, followed by the local common symbolic name.
LIBRARY ENTRIES:	This is a list of all RESOLVEd references.
TASK COMMON BLOCK:	This section of the map can have only one entry (xxxx TSKCOM). It is present during processing if the TSKCOM command is give.
OVERLAY:	Each overlay of a task is listed on a separate page after the main segment has been mapped. The name of the overlay follows the heading, followed by all entry points and undefined symbols contained in the overlay segment (items 2, 3, 4, 5 in the map).
OVERLAY AREAS:	This section appears only on the final map (after the BUILD command) for an overlayed OS. The following sub-headings apply.
PROG	The program label of the program containing the overlay area.
START	The start address of the overlay area.
NEXT	The address of the halfword following the overlay area.
OVS	The number of overlays to use the area.
RAD	The Random Address in the Contiguous File at which the first overlay starts.
SECTS/OV	The number of sectors used for each overlay.

AMAP [fd]

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

REWIND [fd]

The REWIND command assigns fd to Logical Unit 1 and rewinds the file. If fd is not specified, Logical Unit 1 is rewound.

```
OS16MT2 TASK ESTABLISHER 00-00 LOAD MAP

DATE: 11/28/75     TIME: 16:52:13
TYPE: TA
JOB:  EXAMPLE
PARTN: 7000
FD:    WORK:ROOT.TSK


MEMORY CONSTANTS:

  7000 U3OT      7024 BIAS      78EA SIZE      7AC6 UTOP
  7DC6 CTOP


PROGRAM LABELS:

  7024 LTESTB    7126 TCOMPR    713E OVTEST


ENTRY POINTS:

  7024 LTESTB    7024 START2    702C MID2      7034 CONTB
  705E SR1B      707C SR2B      7082 SR3B      7088 OKMESS
  70EA FLMESS    7126 TC1       7134 CODER     7136 OVTEST
  733E PROUT     7358 FOJ       735C DECLAR    739E CHECKS
  73DA PASS      73E0 FAIL      7690 AOVA      7694 AOVB
  7698 AOVC      769C AOVD      76A0 AOVE      77D4 LOADA
  77DC STATA     77E0 LOADB     77E8 STATB     77EC LOADC
  77F4 STATC     77F8 LOADD     7800 STATD     7804 LOADE
  780C STATE     78EA ENDTES


UNDEFINED SYMBOLS:

  START4   MID5     SR4C     TWO      FOUR     DATA


COMMON BLOCKS:

  78EA //        78F2 RTCOMN


LIBRARY ENTRIES:

  AA0A CODEC     AA5A TESTCM    AAE0 CODED     AAF4 TESTRD
  AB46 CODEE


TASK COMMON BLOCK:

  D024 TSKCOM
```

Figure 5-2. TET/16 MAP EXAMPLE – Main Program

```
OS16MT2 TASK ESTABLISHER 00-00 LOAD MAP

DATE: 11/28/75     TIME: 10:52:13
TYPE: TA
JOB:  EXAMPLE

OVERLAY:  OVLY1


PROGRAM LABELS:

  78FA CODEA     792A SRAB


ENTRY POINTS:

  78FA CODEA     792A ENDA      792A SRAB
```

Figure 5-3. TET/16 MAP EXAMPLE – First Overlay

```
OS16MT2 TASK ESTABLISHER 00-00 LOAD MAP

DATE: 11/28/75     TIME: 10:52:13
TYPE: TA
JOB:  EXAMPLE

OVERLAY:  OVLY2


PROGRAM LABELS:

  78FA CODEB     7A90 SRAB


ENTRY POINTS:

  78FA CODEB     7A90 ENDB      7A90 SRAB
```

Figure 5-4. TET/16 MAP EXAMPLE – Second Overlay

WFILE [fd]

The WFILE command assigns fd to Logical Unit 2 and Writes a file mark to the file. If fd is not specified, a file mark is Written to Logical Unit 2. This command should not be used on disc files.

LOG [fd]

The LOG command causes all operator commands to be copied to the unit specified by fd. If fd is not specified, commands are copied to the device or file assigned to Logical Unit 3.

NOLOG

The NOLOG command halts the LOG command operation.

JOB aaaaaaaaaaa

This command allows the operator to title the TET output map with the characters aaaaaaaaaaa. Any characters are valid in this zero to twelve character field. This command is permitted at any point during the execution of TET.

PAUSE

This command causes TET to pause. To continue, enter CO on the console device.

END

Terminates operation of TET and returns control to the operating system.

OPERATING PROCEDURES

General Information

TET/16 may be run as an established task under OS/16 MT2 or as a non-established Background task. TET is supplied as a non-established object program. To run under OS/32, the 32-bit object of TET/16 supplied with the package (TET1632.OBJ) must be established as a task under OS/32.

TET is executed by the use of the LOAD, LFGR, or LDBG command and START command. Logical Unit assignments should be made prior to issuing the START command. When TET is started, the message "TET/16 01-00" is output to LU 7. Table 5-2 lists the TET Logical Units and their use.

TABLE 5-2. TET/16 LOGICAL UNIT ASSIGNMENTS

LOGICAL UNIT	DATA TYPE	USE	DEVICE EXAMPLES
1	Binary	Object-code input	HSPTR, Magnetic Tape, Disc, TTY
2	Binary	Image Load-Module	HSPTP, Magnetic Tape, Disc
3	ASCII	Memory map output, command logging	TTY, Line Printer, CRT
4	Binary	Scratch (optional) (Object code format)	Magnetic Tape, Disc
5	ASCII	Command input	Card Reader, Disc, TTY, CRT
7	ASCII	Error messages, warnings, prompts to operator	TTY, Line Printer, CRT

Logical Unit 5 and Logical Unit 7 must be assigned before execution. Logical Unit 1, Logical Unit 2 and Logical Unit 3 can be assigned by parameters in TET commands or prior to execution. If a scratch file is to be used, Logical Unit 4 must be assigned prior to execution to a binary device of record length 108 or greater. TET determines the existence of a scratch file by whether or not Logical Unit 4 is assigned. The scratch file is rewound at the beginning of pass 1. At the end of pass 1, a file mark is written to the scratch device and it is rewound for pass 2. If it is not a rewindable device, the operator is prompted to reposition the scratch. During pass 2, Logical Unit 1 is assigned to the scratch device and Logical Unit 4 is assigned to the null device. At the end of pass 2, Logical Unit 4 is reassigned to the scratch device, i.e., one scratch device may be used for successive TET/16 runs without operator intervention. Logical Unit 1 is reassigned to the last input device.

Specifying a File Descriptor in certain TET commands causes the file to be assigned to one of the TET Logical Units. Table 5-3 summarizes this action. This information is useful in defaulting File Descriptors to already assigned files.

TABLE 5-3. IMPLICIT TET/16 ASSIGNMENTS

COMMAND	LU ASSIGNED
INCLUDE	1
EDIT	1
RESOLVE	1
REWIND	1
BUILD	2
WFILE	2
MAP	3
AMAP	3
LOG	3

Various parts of the File Descriptor, or the entire File Descriptor can be omitted by the user for convenience. Typically if the fd is omitted, TET checks the Logical Unit that should be assigned for that function, and uses the assigned device until a reassignment occurs. Consider, as an example, the following sequence.

Assume the user has assigned LU1 to the PTRP: before starting TET.

```
ES      TASK
BIAS    C000
IN
IN
IN
IN      MAG1:

IN
IN      PTRP:

BU      TA, TETOUT
```

Finding no File Descriptor in the first INCLUDE command, TET checks LU 1 to see what is assigned. If nothing were assigned, TET would output the message 'LU 1 NOT ASSIGNED'. Since, in this case, the Paper Tape Unit is assigned, TET reads data from LU 1. This Input Unit is used for each INCLUDE command until IN MAG1: where TET closes LU 1 and then assigns the Magnetic Tape Unit to LU 1. Data is then read from Magnetic Tape until IN PTRP: which causes TET/16 to start inputting from the Paper Tape again.

If the fd specified is a Disc File, TET may default a predefined VOLN: to the File Descriptor. If the user does not specify the Volume Name and if the VOLUME command has been issued, TET substitutes the Volume Name specified in the VOLUME command. If the VOLUME command has not been issued the OS System Default Volume is used.

If no extension is specified, TET/16 uses:

- .OBJ for INCLUDE, EDIT, RESOLVE
- .TSK for BUILD TA, BUILD TC, BUILD OV, BUILD RL

Special Considerations for Libraries

A library is given a 36(24) byte UDL.

All object code programs to be included in a library must be output to one file or device. They are then included with one command during the library establishment, and resolved with one command during a task establishment.

Special Considerations for Task Common

A Task Common segment must be a single labelled common block definition of TSKCOM, or a block data subprogram of this common block. No other code is accepted.

A Task Common segment is given a 36(24) byte UDL.

Special Consideration for Tasks With Overlays

No overlay may contain an absolutely located code.

No overlay may reference another overlay in the same overlay area.

If an overlay defines Labelled Common which does not appear in the root nor in any other accessible overlay, it is positioned within that overlay.

Refer to: Guide to User Overlays, Chapter 6 of *OS/16 MT2 Programmer's Reference Manual*

Command Input Stream

Commands to TET can be entered in batch mode or interactively. In either case, certain logical considerations constrain the command sequence.

After starting TET, the command stream is read from LU 5. In batch mode, a REMOTE command should be used to prevent TET from pausing, unless an operator is present. To begin an establishment, the ESTABLISH command must precede all other commands except REMOTE, BATCH, INTERACTIVE, VOLUME, MAP, and AMAP, which can appear at any point in the stream.

TET execution is started by an ESTABLISH command. TET defines the segment on the first pass, and constructs the load module on the second pass. Before specifying any inclusion of task code, TET must know the partition start address and whether the task uses floating point arithmetic. The BIAS, TSKCOM, and the OPTIONS commands must be entered before any INCLUDE command. An OPTION CSEG command, if given, must always precede a TSKCOM command.

The task's program contents must be processed. INCLUDE and EDIT are used to select the input object format programs that are to be part of the main program segment. One INCLUDE statement must be used to bring in a single relocatable program or an entire file. Any number of INCLUDE statements may follow. If a single program is included, the input read operation stops at the end of that program, allowing the next inclusion from the same file or another file.

The EDIT statement reads the entire specified file, and marks for inclusion any programs that have program labels referenced by the code already included. If a Scratch File is present, the entire Edit File is copied to it, but only the required programs are used during the BUILD operation. EDIT statements can be repeated, allowing an EDIT to bring in programs referenced by programs included by a previous EDIT. The user can select programs from multiple files by varying the fd parameters of successive EDIT and INCLUDE statements. See Examples in the section entitled EXAMPLES OF TET OPERATION.

TET resolves all references from one included program to another in a task's included code, but references to a Reentrant Library must be resolved by the use of a RESOLVE command. TET/16 finds the ENTRY symbols that are the same as the referencing EXTRNs in the task. A message indicating that unresolved labels exist is output at the beginning of pass 2, in addition to a table in the MAP which lists these symbols.

One RESOLVE command should be issued after all INCLUDE and EDIT statements of a task including any overlay segments. An error message is printed if more than one RESOLVE command is entered. Only one library segment may be referenced by a task.

TET resolves Task Common references if the TSKCOM command is given, noting their occurrance in the task's LIB. Task Common is referenced by the symbol TSKCOM as the Labeled Common block name.

Overlays are specified and named by OVERLAY statements, and defined by INCLUDE and EDIT statements, after their main segment has been completely defined (by INCLUDE and EDIT statements). If the main segment and overlays together use GET STORAGE calls greater than X'300' bytes, a GET command must be given to reserve such space. Each overlay results in a separate Load Module. All INCLUDE and EDIT statements after an OVERLAY statement and before the next OVERLAY statement (or a BUILD statement) define that overlay. Space beyond the area where the task is loaded, for an expanding operation such as a symbol table, can also be reserved with a GET command.

After an ESTABLISH TA command within TET's first pass, the PRIORITY and TSW commands can also be entered.

The first pass ends and the second pass starts upon encountering a BUILD command. Any unresolved references remaining are reported by a message to the operator.

A BUILD command specifies creation of one Load Module for a task, an overlay, an RL, Task Common block, or an OS using the resolutions and specifications of the first pass. Each overlay module of a task requires a separate BUILD command. All programs included on pass one are read again, and built onto the Load Module as specified during pass one.

TABLE 5-4. RECOMMENDED OPERATOR COMMAND SEQUENCE

	COMMAND	OPTIONAL/REQUIRED
PASS 1	REMOTE	OPT
	LOG	OPT
	VOLUME	OPT
	ESTABLISH	REQD
	BIAS	REQD (except for OS)
	OPTIONS	OPT
	TSKCOM	REQD (if TSKCOM referenced)
	PRIORITY	OPT
	TSW	OPT
	INCLUDE	REQD
	EDIT	OPT
	GET	OPT
	OVERLAY	OPT
	RESOLVE	OPT
PASS 2	BUILD	REQD
	MAP/AMAP	OPT
	END	REQD

Compound Overlay Files

In the case of overlay segments, it is sometimes desirable to build segments on a single file as shown in Figure 5-5.

OVERLAY SEGMENT 1	OVERLAY SEGMENT 2	. . .	OVERLAY SEGMENT n

Figure 5-5. Single Overlay File

This particular file structure introduces special consideration when building on disc. A contiguous disc file must be used. TET attempts to allocate a file when the BUILD command specifies a file that does not already exist on disc. If the user wants to build this compound file, the default allocate function of TET may not be used because TET cannot anticipate how many overlays will be built to the same file.

Therefore, when building the compound file the user must allocate the file before entering TET: (note that this refers to disc only). The user may choose a file size large enough. To determine the exact length of the required file, lengths of the individual segments (expressed as the number of 256-byte blocks) must be summed to arrive at the approximate total image length. In addition, a single LIB sector for each segment to be contained in the file plus five sectors for work space is required by TET. Figure 5-6 illustrates by example:

Overlay 1	3500 hex (round up to nearest X'100' byte boundary)
Overlay 2	500 hex
Overlay 3	2200 hex
Overlay 4	1500 hex

Find the segment length expressed in sectors:

Overlay 1	3500/100 = 35	sectors
Overlay 2	500/100 = 5	sectors
Overlay 3	2200/100 = 22	sectors
Overlay 4	1500/100 = 15	sectors
	TOTAL = 77	sectors
	+ 4	sectors (LIBs)
	+ 5	sectors (Workspace)
	TOTAL = 86	sectors

Figure 5-6. Compound Overlay File Size Example

NOTE

When generating compound disc files, it is not always permissible to change output units during the process. Consider a task with 4 overlays, where overlay 1, overlay 2, and overlay 4 are directed to one fd, and overlay 3 is directed to another. Numbers 1 and 2 are built sequentially. Logical Unit 2 assignment is changed for number 3, and when reassigned for number 4, the Disc File is rewound. Number 4 over-writes those previously built on that file. It is therefore recommended that all overlays for one task be built on the same file, or each on a separate file.

EXAMPLES OF TET/16 OPERATION

Establish A Single Program Task

This command sequence establishes a task such as TET/16, which requires no linking of programs and no editing or resolving of external references.

```
ESTABLISH TA              (Establish a task)

BIAS A000                 (Partition start address A000)
GET A00                   (Reserve 2.5K memory above code)
INCLUDE PTRP:             (Input object code from Paper Tape Reader)

BUILD TA,MAG1:            (Build load module on Magnetic Tape)

MAP PR:                   (Produce map)

END
```

Establish A Simple Task From Selected Programs

This command sequence establishes one Load Module from a file that contains other programs not required by this task. Programs A, C and D are to be included in the task. Figure 5-7 is a description of the input and output files.

The required command sequence is as follows:

```
ESTABLISH TASK

BIAS 9000                 (Partition start address 9000)

INCLUDE FILEA, PROGA

INCLUDE ,PROGC

EDIT                      (Includes PROGRAM F, referenced by PROGRAM A, and PROGRAMS D
                           and H referenced by PROGRAM C.)

BUILD TASK                (As no fd is specified, and assuming that LU 2 is not assigned to a Contiguous
                           Disc File, this task is built in memory, where it requires a work area, the
                           total size of tasks A, C, D, F, and H.)

MAP PR:                   (Produces a map of the task).

END                       (Return control to the system)
```

Establish a Complex Task

To illustrate the capabilities of TET in establishing a more involved task, this command sequence shows establishment of a task with overlays, references to a Reentrant Library and Task Common, absolute-address and special space requirements. The main segment and each overlay is to be built on a separate file, and it is necessary to use two Disc Volumes because of space limitations. Disc is available for building the task and the command input is on a CSS File.

The task comprises three programs from FILE A and three from FILE C. Two overlays are necessary: one from FILE B, and the other from FILE C with two programs that it references. These files are shown in Figure 5-8.

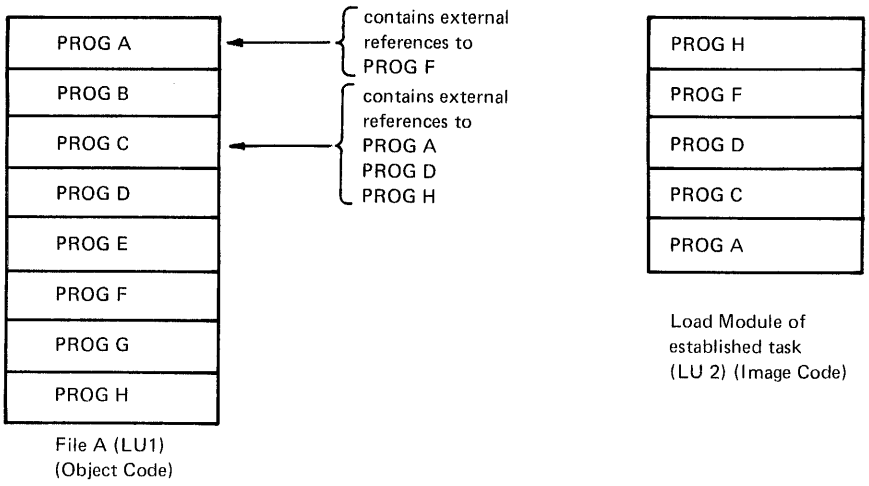


Figure 5-7. Simple Task Establishment

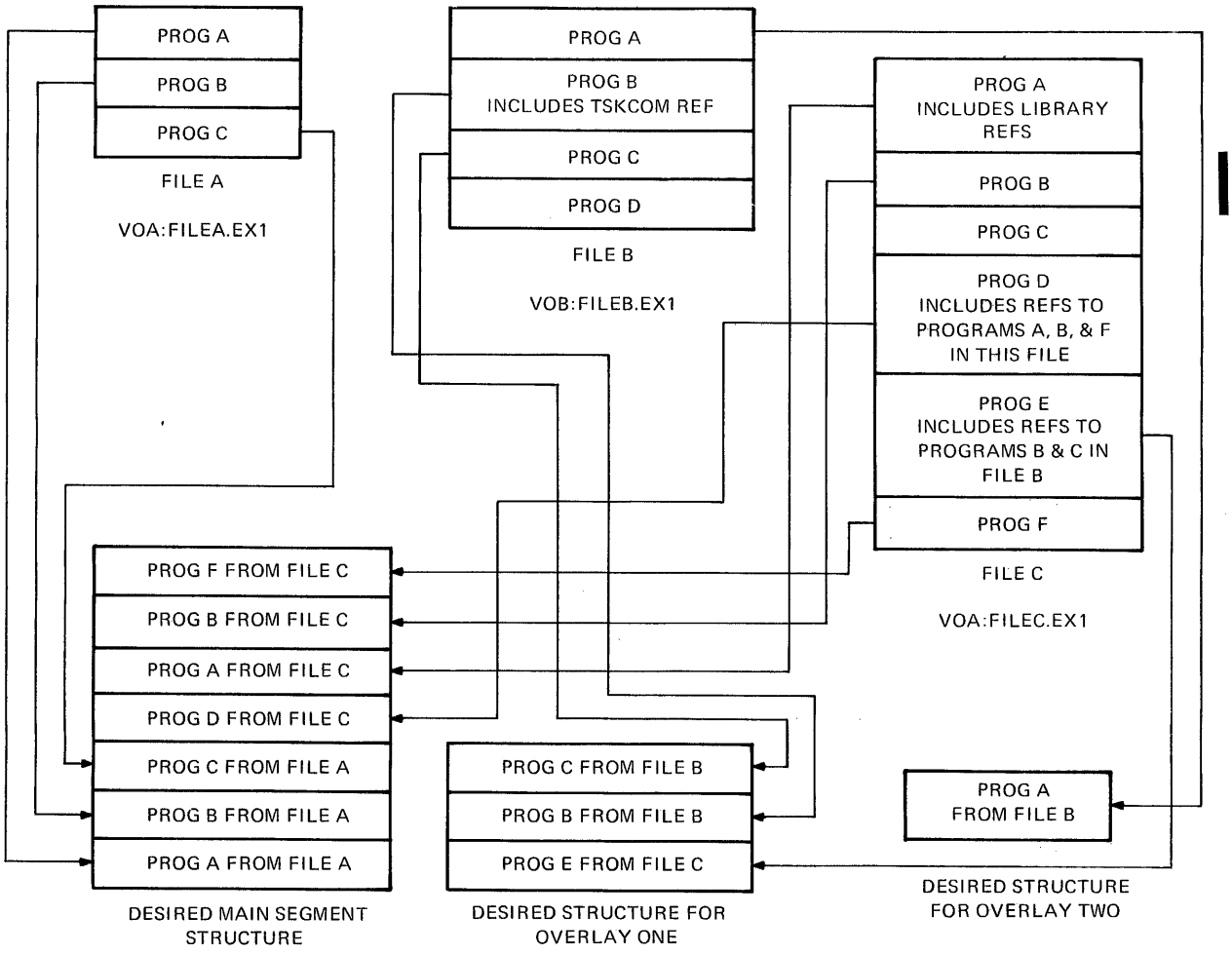


Figure 5-8. Graphic Description of Task With Two Overlays

The command sequence is:

REMOTE	(Specifies that TET/16 should not pause since input is controlled by CSS).
ESTAB TASK	
BIAS 8000,8200	(Partition start address 8000. Relocatable bias 8200 provides 512 bytes of absolute-address space before beginning relocatable code).
VOLUME VOA	(VOA is the Default Volume Descriptor)
TSKCOM E400	(Task Common partition address E400)
INCL FILEA.EX1	(Includes all of FILE A)
INCL FILEC.EX1,PROGD	(Includes PROGRAM D of FILE C)
REWIND FILEC.EX1	
EDIT	(Edits FILE C, which includes PROGRAMs A,B, and F. Default is to LU 1 which was last assigned to FILE C).
RESOLVE MAG1:,7200	(PROGRAM A in FILE C contains references to a reentrant library module which is on the Magnetic Tape Unit. The RESOLVE command reads the RL object module from Magnetic Tape and satisfies the references.)
PRIORITY 20,14	(Specifies that the task is to run at priority 20, unless raised at run time, to a priority no higher than 14).
GET 400	(Allows 1024 bytes for GET STORAGE calls in the main segment)
OVERLAY OVONE,NEW	(Terminates definition of the main segment; names and initiates definition of the first overlay)
INCL FILEC.EX1,PROGE	(Include first overlay)
EDIT VOB:FILEB.EX1	(Edits FILE B for programs referenced by PROGRAM E. Includes PROGRAMs B and C, from FILE B; PROGRAM C references TSKCOM, which is noted in the task's LIB).
OVERLAY OVTWO	(Terminates first overlay definition and initiates second)
INCLUDE VOB:FILEB.EX1,PROGA	(Include second overlay)
BUILD TASK,FILED.LMD	(This command terminates all specifications; pass two starts, and all INCLUDEd and EDITed programs will be copied from the scratch to the Load Module, which is being built on FILE D of VOA. The extension field can be any three characters, such as (LMD) Load Module)
BUILD OVLY,FILEG.LMO	(Builds overlay one of FILE G of Volume A; LMO = Load Module Overlay)
BUILD OVLY,VOG:FILEA.LMO	(Builds overlay two on FILE A of Volume G)
MAP	(Produces a map of the entire task)
END	(Returns control to the system)

Figure 5-9 illustrates a map of this task in memory. The main segment is built as shown, with the absolute-address area first, followed by the main inclusions, the overlay area large enough for the longest overlay, and the GET requested area. The system, at load time, places the CTOP indicator to the top of the partition.

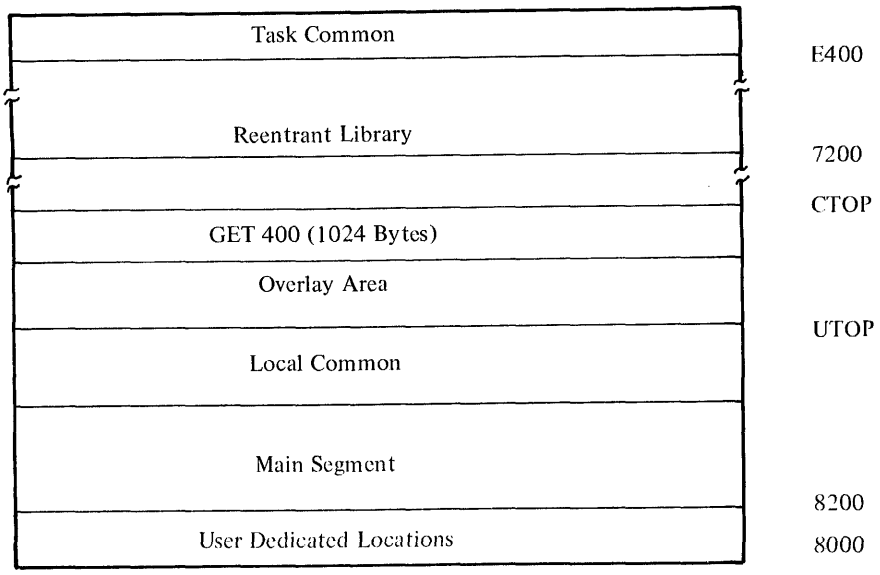


Figure 5-9. Memory Map of Overlay Task Establishment

Establish a Reentrant Library Segment

This command sequence establishes a Reentrant Library Segment, sharable by any tasks in the system which reference it.

This example describes the establishment of selected routines from the FORTRAN Run Time Library as a Reentrant Library Segment. The command sequence might be:

ESTAB RL	(Specifies that the establishment is to produce a Reentrant Library Load Module)
BIAS 7200	(Position start address 7200)
INCL MAG1:	(Include the FORTRAN run time routines)
BUILD RL,RELIBRY	(Ends pass one definition; builds library Load Module on file VOLN:RELIBRY.TSK)
MAP	(Specifies a map of completed module)
END	(Returns control to the system)

Establish a Task Common Segment

This command sequence establishes a Task Common Segment, sharable by any tasks in the system which reference it.

ESTAB TCOM	
BIAS E400	(Partition start address)
INCL PTRP:,*	(Include first program only, from Paper Tape Reader)
BU TCOM,MAG1:	(Build on Magnetic Tape)
JOB TCOM SEGMENT	(Title for map)
MAP	(Output map)
END	

Establish a Task With Multilevel Overlays

A task runs through three discrete phases before terminating, and the code to process each phase may be overlayed. Two of the phases share some common routines. The third calls two other routines which may also be overlayed as illustrated in Figure 5-10. A Memory Map of Multilevel Overlays is shown in Figure 5-11.

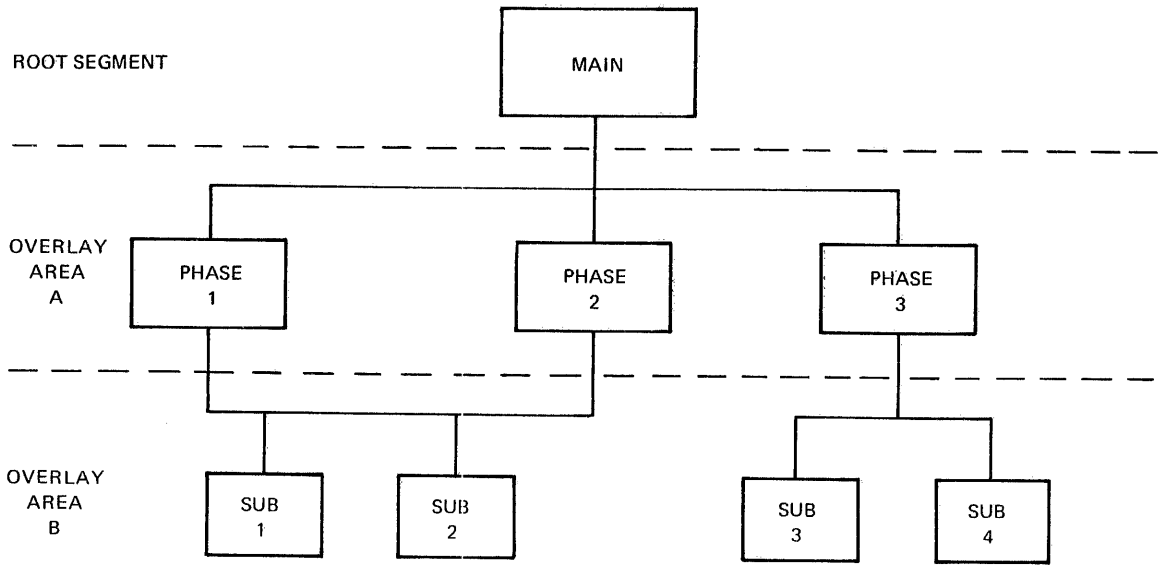


Figure 5-10. Multilevel Overlays

The sequence of commands is:

```
ESTAB TA
BIAS 8000                (Partition start address)
INCL MAIN.OBJ            (Main control program)
OV PHASE1,NEW            (First overlay area, overlay name PHASE1)
INCL PHASE1.OBJ          (Include overlay)
OV PHASE2                (Overlay name PHASE2)
INCL PHASE2.OBJ          (Include overlay)
OV PHASE3                (Overlay name PHASE3)
INCL PHASE3.OBJ          (Include overlay)
OV SUB1,NEW              (Second overlay area, overlay name SUB1)
INCL SUB1.OBJ            (Include overlay)
OV SUB2                  (Overlay name SUB2)
INCL SUB2.OBJ            (Include overlay)
OV SUB3                  (Overlay name SUB3)
INCL SUB3.OBJ            (Include overlay)
OV SUB4                  (Overlay name SUB4)
INCL SUB4.OBJ            (Include overlay)
BU TA,MAIN.TSK           (Build task load module)
BU OV,PHASE1.OVL         (Build first overlay for area A)
BU OV,PHASE2.OVL         (Build second overlay for area A)
```

```
BU OV,PHASE3.OVL        (Build third overlay for area A)
BU OV,SUB1.OVL          (Build first overlay for area B)
BU OV,SUB2.OVL          (Build second overlay for area B)
BU OV,SUB4.OVL          (Build fourth overlay for area B)
MAP                     (Map in address order)
AMAP                    (Map in alphabetic order)
EN
```

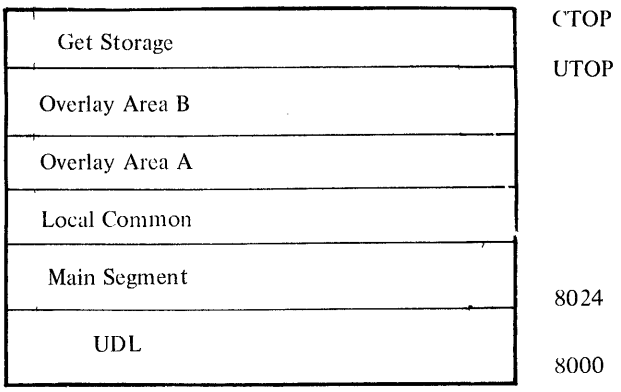


Figure 5-11. Memory Map of Multilevel Overlays

Establish OS/16 MT2

The following command sequence establishes the Operating System, building it on the Contiguous Disc File VOLN:OS16R00.001 in a form that can be bootstrapped into memory.

```
ESTAB OS
INCLUDE EXEC16.OBJ          (Include Executive as first module)
INCLUDE FMGR16.OBJ          (Include File Manager)
EDIT DLIB16.OBJ             (Edit Driver Library)
INCLUDE USERDRV.OBJ         (Include user-coded drivers)
INCLUDE CMDP16.OBJ          (Include Command Processor as last module)
BUILD OS,VOLN:OS16R00.001   (Build onto Contiguous Disc File)
MAP PR:
AMAP
END
```

CHAPTER 6
CAL/16

INTRODUCTION

The Common Assembler (CAL/16), 03-101, is a subset of the Common Assembler (CAL) 03-066. It is used on 16-Bit Processors and assembles source programs for 16-Bit Processors only. It is supplied with the OS/16 MT2 package in three forms, memory-based (CAL/16) and disc-based (CAL/16D and CAL/16DS).

Note that CAL/16 cannot be used to assemble the intermediate text output of FORTRAN V because the COMN and BDATA pseudo-ops are not processed by CAL/16.

SYSTEM REQUIREMENTS

The following are the minimum requirements to operate CAL/16:

- any 16-Bit Processor
- OS/16 MT2
- approximately 14KB memory over and above OS/16, but exclusive of symbol table space for CAL/16
- approximately 16.25KB memory over and above OS/16 for CAL/16D (15KB for CAL16DS)
- a source input device
- a source output device
- a binary output device

CAL/16 FEATURES

For detailed information on the assembler language, refer to the *Common Assembler Language User's Manual,* Publication Number 29-375.

CAL/16 contains the features of CAL with the exception of those features listed below.

1. The TARGT pseudo-op must specify 16. All assemblies are targeted for 16-Bit Processors.
2. All 32-bit instruction processing is deleted.
3. Contiguous or Indexed files may be used as the cross reference file. If an Indexed file is used, its record length should be $\geqslant 256$.
4. The following op-codes are deleted.

A	CHVR	ECS	M	RBT	SI	WDHS
ABLF	CI	EXHR	MR	RDCS	SPSW	WDRHS
AI	CL	GIPI	N	RDRHS	SR	WPDHS
AM	CLI	L	NHM	RPDHS	ST	X
AR	CLR	LA	NI	RPSW	STBHS	XHM
ATLF	CR	LBHS	NR	RTLF	TBT	XI
BDCS	CRC12	LBHSI	O	S	TI	XR
BESHS	CRC16	LHL	OHM	SBHSI	TLATE	
BNSHS	D	LHS	OI	SBT	TS	
C	DCHS	LPSWR	OR	SCP	UNC	
CBT	DR	LR	RBLF	SHM	WDCS	

5. The following pseudo-ops and their associated features are deleted.

Deleted pseudo-op	Suggested alternate
ABS	ORG
CAL	N/A
CNOP	N/A
DSF	DS
NOCAL	N/A
NORX3	N/A
SYM	N/A
SQUEZ	N/A
BDATA	N/A
COMN	N/A
DLIST	DC, DS
IFE	IFZ
IFO	IFNZ
MSG	N/A
WIDTH	N/A
NOSQZ	N/A
IF	IFNZ
PURE	N/A
IMPUR	ORG
OPT	N/A

6. The following constant types are deleted from the DC pseudo-op.

F – fullword decimal
E – single precision floating point
D – double precision floating point

OPERATING PROCEDURES

Either version of CAL/16, memory-based or disc-based, is loaded in object Format, or is established first using TET/16 and then loaded by the image loader. Logical unit assignments are made before the assembly is started. Assembly options are given either in the source or as arguments to the START command. The arguments supersede any similar pseudo-ops in the source.

Logical Unit Assignments

Logical Unit

01 Source Input Device – The source program to be assembled is read from this device on pass 1. On pass 2 this device is rewound and reread only if SCRAT and PPAUS are not selected.

02 Binary Object Output Device – the assembled object program is written to this device on the last pass. 108 byte records are output.

03 Assembly Listing Output Device – the assembly listing is printed on this device on the last pass.

04 Primary Source Copy (Scratch) Device, Output and Input (Bulk Storage) – the source program is written to this device on pass 1 if the SCRAT option is specified and read on pass 2. This device is rewound following pass 1. If Logical Unit 4 is assigned to a disc file, the logical record length of that file should be $\geqslant$ 80 bytes.

05 Symbol Table Cross Reference File Device (direct Physical Access File) – cross reference information is built on the file during the last pass and is printed after the last pass. This file is accessed randomly. Each cross reference record is 256 bytes long.

06 Symbol Table File – an indexed file of logical record length 256 bytes or a contiguous file (recommended size 256 sectors). Used by disc-based version only.

07 Source Library Input Device – source information is read from this device on pass 1. Source information is also read from this device during pass 2 if SCRAT is not specified. This device is rewound each time a COPY statement is read.

08 Symbol Table Work File – an indexed file of logical record length 256 bytes. Used by disc-based version only.

09 Scratch Device for Accumulation of Errors, Output and Input (Bulk Storage) – errors that are included in the error count along with the associated statement number where the error occurred are written to this device during the last pass if selected by an ERLST option and then read back at the end of the assembly. If Logical Unit 9 is assigned to a disc file, the logical record length of that file should be $\geqslant$ 36 bytes.

START Options

The options are specified as arguments to the OS START command. Any combination of spaces and/or commas may separate or follow external options in the START command. In addition, each external option may be abbreviated to a minimum of the characters underlined in the following table:

OPTION	OPTION VALUES
PPAUS	None
CROSS	None
NLIST	None
SQCHK	None
SCRAT	None
WIDTH	Width of print line
LCNT	Number of lines/page
ERLST	None
NLSTC	None
FSIZE	Symbol table file size
FREZE	None

Each option has the same meaning as the pseudo-op of the same name. Options or option values specified in the START command remain in effect for the entire assembly (or assemblies, if BATCH is in effect) regardless of any pseudo-op encountered in the source. For example, if NLIST is specified as an external option, CAL ignores any LIST or NLIST pseudo-ops.

Operation of Memory-Based CAL/16

CAL/16 operates in a similar manner to CAL.

Operation of Disc-Based CAL/16D and CAL/16DS

CAL/16D is designed to assemble programs of any size irrespective of available resident memory. It differs from CAL/16 in that part of the symbol table is built on disc if it cannot fit in available memory, and thus CAL/16D is only operational in a disc based system. The use of CAL/16D is basically the same as CAL/16, except two additional logical unit assignments are required (for symbol table work), and an additional starting option is provided.

Logical Unit 6 is assigned to either an Indexed file of 256 byte records, or a Contiguous file. For a Contiguous file the number of sectors allocated must be 256, or the number specified in the starting option FSIZE.

If the Contiguous file assigned contains less than 256 sectors (or less sectors than the value specified in FSIZE), an EOM message is output during the first pass of the assembly, and the task is paused. The user has the option of assigning a file of correct length and continuing the task.

Starting Option FSIZE is the size of the file assigned to LU 6; this value is used to initialize the file for building the symbol table. If FSIZE is not supplied, the default value is 256. The use of 256 records easily allows for assembling programs containing 2500 symbols. (The full Command Processor in OS/16 MT2 contains approximately 1800 symbols). However, by specifying FSIZE larger than 256, larger programs may be assembled.

Whenever FSIZE is supplied and a Contiguous file is assigned to LU 6, the number of sectors allocated must be greater than or equal to FSIZE. (See the LU6 description.)

CAL16/DS is a smaller version of CAL/16D and must be used if a processor has only a 32KB memory. File allocations must be considered carefully when executing in a 32KB environment. The following is suggested:

– use contiguous files for LU 6 and LU 8. Use CAL/16 option FSIZE=200

Example:
```
ALLO SYM,CO,200
ALLO MRG,CO,200
```

– do not use the CROSS option

– for sysgen assemblies use a contiguous file for the CUPOUT file. A 1000 sector file is sufficient for most sysgens.

Examples

The following three examples illustrate how the Common Assembler is loaded and the logical units are assigned to the devices or files containing the appropriate information.

Example 1 (Non-Disc system)

```
LDBG      MAG1:       Load CAL/16 from mag tape
ASSIGN    1,CR:       Assign the source module input to LU 1 (Card Reader)
ASSIGN    2,PTRP:     Assign the object module output to LU 2 (Paper Tape Punch)
ASSIGN    3,PR:       Assign the list device to LU 3 (Line Printer)
ASSIGN    4,MAG1:     Assign a scratch device to LU 4 (Mag Tape)
ASSIGN    7,PTRP:     Assign the source SYSGEN output of CUP to LU 7
                          (Paper Tape Reader)
```

Example 2 (Disc system using memory-based CAL/16)

```
LDBG      CAL16.OBJ          Load CAL/16 from a Disc File
ALLO      EXEC.OBJ,I         Allocate a file for the assembly's object output
ALLO      SCRAT,IN,80        Allocate scratch file
ALLO      CROSS,IN,256/2     Allocate cross-reference file
ASSIGN    1,EXEC.CAL         Assign the OS/16 source module to LU 1 (Disc File)
ASSIGN    2,EXEC.OBJ         Assign the object module output to LU 2 (Disc File)
ASSIGN    3,PR:              Assign the list device to LU 3 (Line Printer)
ASSIGN    4,SCRAT            Assign scratch file to LU 4
ASSIGN    5,CROSS            Assign cross-reference file to LU 5
ASSIGN    7,CUPOUT           Assign the course SYSGEN output of CUP to LU 7
                                (Disc File)
```

Example 3 (Disc system using disc-based CAL/16D)

```
LDBG      CAL16D             Load CAL/16D from a disc file
ALLO      EXEC16.OBJ,I       Allocate a file for the object output
ALLO      SYMTAB,CO,512      Allocate a file for symbol table
ALLO      SYMWORK,IN,256     Allocate a file for symbol table work
ASSIGN    1,EXEC16.CAL,SRO   Assign OS/16 source module to LU 1
ASSIGN    2,EXEC16.OBJ       Assign object module output to LU 2
ASSIGN    3,PR:              Assign the list device to LU 3
ASSIGN    6,SYMTAB           Assign file for symbol table to LU 6
ASSIGN    7,CUPOUT           Assign the source SYSGEN output of CUP to LU 7
ASSIGN    8,SYMWORK          Assign file for symbol table work to LU 8
```

The scratch unit (LU 4) should be assigned only if the scratch feature is being used. For example, scratch should not be used if the source modules are on disc.

Once the correct units have been assigned and the devices are ready, CAL/16 is started with the START command. For the previous Examples 1 and 2, one of the following commands can be given:

```
START                for non-scratch assemblies where source input is rewindable
START,PPAUS          for non-scratch assemblies where source input is not rewindable
START,SCRAT,CROSS    for scratch assemblies with cross-reference listed
```

For Example 3 (loading CAL/16D), one of the following commands can be given:

```
START,FSIZE = 512    for disc-based assemblies

START                for disc-based assemblies where symbol table file is indexed
                     or 256 sector contiguous file
```

Features in CAL/16D not in CAL/16DS

CAL/16DS is a somewhat smaller version of CAL/16D designed for use in 32KB systems. The following features have been removed.

- The ERLST, SQCHK and WIDTH options are not provided
- The SPACE psuedo-op is ignored
- Common-mode CAL mnemonics are not accepted
- Output from CAL MACRO is not accepted
- The LCNT, LSTM, NLSTM, and DCY psuedo-ops are not accepted

CHAPTER 7
DISC INTEGRITY CHECK

INTRODUCTION

The Disc Integrity Check Utility Program Number 03-080 provides a means of recovering open disc files following an Operating System crash. Recovery from a system crash requires reloading the Operating System. This utility is then used to restore the integrity of data on disc volumes that were dismounted without being marked off-line. (A system crash has the effect of dismounting on-line disc volumes). The program rebuilds the Bit Map and validates file pointers of indexed, contiguous, and chained files under OS/16 MT2, regardless of whether the files are supported by the Operating System. (Chained files are supported by OS/32 MT).

The Disc Integrity Check closes all files found to be assigned and validates all control information on the disc. This latter function is performed in case bad data was written to the volume during the system crash. Complete volume recovery is not always possible because bad data may have been written to the volume prior to the crash. The program output messages explain the status of individual files or the entire disc, and describe what actions have been taken or attempted.

CAUTION

IT IS IMPERATIVE TO RUN THE INTEGRITY CHECK UTILITY WHENEVER THE INTEGRITY OF A DISC IS IN QUESTION. FAILURE TO DO SO IMMEDIATELY CAN RESULT IN THE UNNECESSARY LOSS OF DATA AND FILES

Systems without direct-access devices need only restore the Operating System environment that existed prior to the system crash. No further action is required.

Systems with direct-access devices that did not have any direct-access files assigned at the time of the system crash can be recovered using the procedure for systems without direct-access devices. However, it is not always possible to tell if any files are assigned, (i.e., a program may have made an assignment using SVC 7). Therefore, it is recommended that the Disc Integrity Check be used on all systems configured with direct-access devices. Failure to execute this utility program after a system crash may leave direct-access volumes in a state where the volume cannot be marked on without protect.

SYSTEM REQUIREMENTS

For operation, the Disc Integrity Check requires:

16-Bit Processor
Operating System OS/16 MT2
8.50 KB of memory above the OS size, plus an optional buffer for Read Check Operation
A console device (CRT or TTY)
Any currently supported disc device.

PRINCIPLES OF OPERATION

When user tasks assign direct-access files, a File Control Block (FCB) is created in memory which is used by the file management routines in processing user requests to the file. The most current state of an assigned file is defined within the FCB.

The File Directory, located on each disc volume, may not have up-to-date information recorded for each file. In particular, the sector allocation Bit Map may show sectors as allocated which are not allocated, as far as the File Directory is concerned. The File Directory also shows the use of each file (i.e., unassigned, assigned) and the access privileges currently in effect for each file. Thus, after a system crash, some files may be shown as assigned in the File Directory with their FCBs removed from memory by the crash.

The Disc Integrity Check program closes all assigned files and validates the control information on the disc. The program START command assigns the Disc Volume to Logical Unit 1 (LU 1) via the Supervisor Call 7 (SVC 7) ASSIGN call and assigns the print device to LU 3. The attributes are fetched via the SVC 7 – FETCH ATTRIBUTES and the device code is checked to see if the device is a disc. Once it is established that the device is actually a disc, the maximum Logical Block Address (LBA) is calculated and the Volume Descriptor (VD) is read in from LBA zero via SVC 1. If there is an I/O error, the disc must be reinitialized, and the program terminates with output message (#4).

NOTE

All program output messages are listed by number in the last paragraph of this chapter.

If there is no I/O error, the volume name is checked for conformity with Operating System conventions, and all pointers are checked to see that they do not exceed the maximum LBA on the volume. If valid, the Bit Map final LBA is calculated and the Bit Map is zeroed. One sector is allocated in the Bit Map for the VD and the sectors to be occupied by the Bit Map are allocated. If the VD points to an OS image on the pack (old 32-bit packs only), the sectors it occupies are allocated. The VD is checked to see if a File Directory is present; if not, the program is finished.

For volumes with directories, the utility starts by reading the first directory block. The directory block is then allocated in the Bit Map. Directory blocks are processed in this manner until the end of the directory or until a bad directory link is found. If the latter condition occurs, the directory chain is closed off and a message (#6) is printed. Since the rest of the directory cannot be found, the files it defined are lost and must be re-allocated and rebuilt. Then the first directory block is re-read and checked for active file entries. Non-active entries are skipped. Active file entries have their file names validated by verifying that the name conforms to OS naming conventions. Files are checked for TYPE and to see that both the first and last LBA lie within the volume.

The Disc Integrity Check Utility Program verifies contiguous, chained, and indexed files. All three file types are supported by OS/32 MT. OS/16 MT2 supports only contiguous and indexed files.

Contiguous Files

Read/Write counts of contiguous files are checked by the program. If one or both counts are non-zero, message (#10) is printed with this file. The program then allocates the sectors occupied by the file.

Chained Files

Chained files are checked to see if the block size is zero, in which case the file is deleted and messages (#17) and (#9) are printed. The begin field (first logical block address) is checked to determine if it is zero. If zero, the Read/Write counts are set to zero and the number of logical records in the file are examined. If the number of logical records is non-zero, the file is deleted and message (#9) is printed. If the first logical block address is non-zero and there are no records in the file, both first logical block address (FLBA) and last logical block address (LLBA) are set to zero before the next directory entry is processed.

The number of blocks in the file are calculated when both the begin field and the number of logical records are non-zero. The chain is then followed through the link fields of each block until either the end (as specified in the directory) is reached, or the pointer runs off the volume. If the pointer runs off the volume, the file is deleted and message (#9) is printed.

If the number of blocks between FLBA and LLBA does not equal the calculated number of blocks, the file is deleted and message (#16) is generated followed by message (#9).

If the number of blocks between FLBA and LLBA equals the calculated number of blocks, the program starts at FLBA and goes through the chain again, the chain is followed through the link field of each block and the sectors the file occupies are allocated in the Bit Map. The last link, as recorded in the directory, is checked. If it points to another entry, the chain is broken and the message (#11) is printed.

Indexed Files

The data and index block size of indexed files are checked. If zero, the file is deleted and messages (#17) and (#9) are printed.

The begin field is tested for zero. If zero, the Read/Write counts are zeroed and the number of logical records are tested for zero. If the number of logical records is non-zero, the file is deleted and message (#9) is printed. If the number of logical records is zero and the begin field is non-zero, both FLBA and LLBA are set to zero before the next directory entry is processed.

The number of data pointers is calculated for a file with a non-zero begin field and one or more logical records. The index and data block pointers, contained in each index block, are tested to make certain that they point to valid sector addresses on the disc and are not overlapping other files. The backward pointer of the first index block is tested for zero. If non-zero, the file is deleted and messages (#11) and (#9) are printed. All other backward pointers are checked to make certain that they point to the previous index block or the file is deleted and messages (#11) and (#9) are printed.

If the forward pointer in the last index block (LLBA) is not zero, it is set to zero, the index block is rewritten and message (#11) printed. If the last index block contains non-zero pointers after the last calculated record, these pointers are zeroed and message (#18) is printed. If the last calculated data pointer is not contained in the last index block (LLBA), the file is deleted, messages (#16) and #9) are printed. Otherwise, all sectors occupied by the file are allocated in the Bit Map. If the file was assigned, the Read/Write counts are set to zero and message (#10) is printed.

Directory entries are processed until the end of the directory. All empty non-preallocated directory blocks are deleted. After all file entries are validated, sector zero of the disc is read once more. If the disc is marked off-line and VATR.ONM in the attributes field is set, Disc Integrity Check zeros this bit and rewrites the VD. LU 1 is closed and the program terminates.

OPERATING PROCEDURE

The following procedure is recommended for all systems configured with direct-access devices. Load Disc Integrity Check after reloading the Operating System. Execute the program once for each disc pack by passing the disc file descriptor as an argument to the program by use of the START command.

The file descriptor must be passed as a parameter in the START command; it may not be preassigned. The list device may be preassigned to LU3, or the file descriptor can be passed as a parameter to the program in the START command. Any disc supported by the Operating System can be used with the Disc Integrity Check program and any list device can be used to output messages. The program operates as an Executive Task; OPTION ETASK must be specified when loading the object program. Also the program can be TETed as an E-Task and loaded by the image loader. The CSEG option must never be specified.

Disc Integrity Check may be run with the disc marked off-line, or with the disc marked on-line without protect if no files are assigned.

If the Operating System is overlayed on the disc volume one cannot mark the disc off-line and run Disc Integrity Check.

1. Mark the disc off-line (or on-line with OS and without protect if it is the OS volume).
2. Use the START parameters to execute Disc Integrity Check.

The syntax of the start command is: START,disc fd [,print fd] [,option]

Examples:

```
T    .BG
OPT  ET
LD   DISCHECK
ST   ,DSC1:,CON:
ST   ,DSC1:,PR:                    Normal program with read check
ST   ,DSC1:,CON:,READCHECK         Normal program with read check
ST   ,DSC1:,CON:,NOREADCHECK       Normal program without read check
ST   ,DSC1:,CON:,CLOSE             Close files only
```

The Disc Integrity Check Utility Program provides the following options:

READCHECK: If READCHECK is specified, the program searches for bad sectors in the following manner. As many sectors as can be accommodated in a buffer between UTOP and CTOP are read. If non-zero status is returned, a sector-by-sector read is performed until the bad sectors are located. Any bad sectors are marked as allocated in the Bit Map and message (#15) is output. If zero status is returned, the next group of sectors is read. This process continues until the entire disc is checked.

READCHECK is performed differently if the program is loaded into a small partition where the buffer size is less than four sectors. In this situation, every nth sector in each cylinder is read, where n is chosen for the particular device, until all the sectors are checked.

Default is READCHECK

NOREADCHECK: If this option is specified, no READCHECK is performed. When the DISC INTEGRITY CHECK clears the Bit Map, all sectors previously flagged defective are freed. This option should never be specified if a disc is known to have bad sectors.

Default is READCHECK

CLOSE: When this option is specified, all active file entries in the Directory are checked for non-zero Read/Write counts. This means the file is still assigned.

Non-zero Read counts are set to zero and message (#10) is printed. Non-zero Write counts cause message (#19) to be printed; the counts are not reset.

CAUTION

SINCE NO ADDITIONAL VALIDITY TESTS ARE PERFORMED WITH THIS OPTION, IT SHOULD BE USED IF FILES WERE ASSIGNED FOR READ ONLY OR IF NO FILES WERE ASSIGNED.

PROGRAM OUTPUT

This Disc Integrity Check program produces twenty-two messages. One message informs the user the program is operating and the remaining are error messages. The following table classifies the error messages into four categories.

TYPE	MEANING	MESSAGE #
Warning	Informatory messages	10,11,14,15,18,19
File Errors	A file is deleted or data is lost	7,8,9,10,16,17
Directory Errors	Possibility of many files lost	6,22
Operator Errors	Operator intervention required	2,3,4,5,12,13,20,21

The following messages are output by the program:

1. DISCCHECK XX-YY

   This message is initially printed on the Console and informs the user that the program is operational, XX is the current revision level of Disc Integrity Check, YY is the update level within the revision.

2. FD-ERR

   This message is logged if an invalid file descriptor is issued in the START command.

3. FORM-ERR

   This message is printed when the options in the START command do not conform to specifications.

4. BAD PACK-REINITIALIZE

   This message is printed if the disc cannot be checked because of some I/O error. It is also printed if the disc is not Write enabled.

5. DEVICE NOT DISC

This message is logged if the first file descriptor in the START command is not a disc descriptor.

6. BAD DIRECTORY-CHAIN BROKEN

This message is printed if the pointer to the next directory block is not valid. The directory chain is closed off. This means that the directory entries further down the directory chain are now not accessible to any program, and the files that they defined are lost. To obtain a list of the valid files, use the OS FILES Command.

7. BAD FILENAME filename

This message is printed when a filename does not conform to the OS naming conventions. The file is then deleted.

8. INVALID FILE TYPE, FILE filename

This message is printed when the file type field in the directory is not contiguous, chained or indexed. The file is deleted.

9. FILE filename DELETED

This message is printed after a previously active directory entry is marked inactive. Causes for this message include:

An invalid directory pointer
An invalid file type (see message #8)
A bad file name (see message #7)
A contiguous file with the last LBA (LLBA) less than the first LBA (FLBA)
An invalid block size (see message #17)
An invalid chained or indexed file link
Calculated number of blocks not equal to actual number of blocks in a chained file (see message #16)

This message is output for an indexed file for the following reasons:

The index or data pointers are invalid

The calculated number of data blocks does not agree with the actual number of data blocks between FLBA and LLBA (see message #16)

The FLBA is zero but the number of logical records is non-zero

If the last data pointer is not contained in the last index block

When a backward pointer does not point to the previous index block (see message #11)

10. POTENTIAL LOST DATA ON FILE filename

This message is printed when a file is closed by the program.

11. CHAIN BROKEN ON FILE filename

This message is printed under the following circumstances:

When examining a chained file, the Last Logical Block (LLBA) currently recorded in the directory points to another LBA. This causes the chain to be closed off at the LLBA, and the message to be printed. Any blocks of data following the LLBA are lost.

When examining an indexed file, the last forward pointer of an index block is not zero. This causes the last forward pointer to be reset to zero, the first sector of the index block to be rewritten, and the message to be printed.

This message is also printed when the backward pointer of an index block does not point to the previous index block. The message is then followed by message (#9).

12. ASSIGN ERROR CODE XX

This message is logged if there is an error while trying to assign either LU 1 or LU 3 to their respective file descriptors (xx = returned SVC 7 status).

13. IO ERROR ssdd

    This message is logged if a non-zero status was received while trying to Write to LU 3. The program is then paused (ss = device independent status, dd = device dependent status).

14. IO ERROR ssdd LBA = nnnnnn

    This message is logged if a non-zero status is received while trying to Read or Write a sector on the disc (ss = device independent status, dd = device dependent status, nnnnnn = hexadecimal logical block address).

15. BAD SECTOR, LBA = nnnnnn

    This message is logged if a bad sector was found while doing a Read check. This sector is then marked as allocated in the Bit Map (nnnnnn = hexadecimal logical block address).

16. INCORRECT BLOCK COUNT ON FILE filename

    This message is printed when the calculated block count does not equal the actual number of data blocks for a chained or indexed file. This message is followed by message (#9).

17. INVALID BLOCKSIZE OF ZERO ON FILE filename

    This message is printed when the blocksize field in the directory is zero for a chained or indexed file. The file is deleted.

18. DATA POINTERS FOLLOWING LAST POINTER NOT ZERO, FILE filename

    This message is logged when any index file data block pointers following the last calculated pointer are non-zero. The pointers are then set to zero.

19. FILE filename ASSIGNED FOR WRITE, COUNTS NOT RESET

    This message is generated when non-zero Write counts are found during execution of the Close option. Disc Integrity has not been restored and the program has to be re-executed without the Close option.

20. ILLEGAL OPTION COMP

    This message is logged if OPTION COMP is in effect when the program is started.

21. ILLEGAL OPTION UT

    This message is logged if OPTION ET is not in effect when the program is started.

22. DIRECTORY INCORRECT - REBUILD WITH MARK ON

    There is an incorrect pointer in the volume descriptor or files have been allocated or deleted using OS/32, OS/16 MT2 R03 or an earlier revision of OS/16. The directory should be rebuilt with the NEW option of the MARK command, and Disc Integrity Check should be run again.

The following list contains examples of typical messages output by the program.

```
DISCCHECK XX-YY
BAD SECTOR, LBA = 6154
DATA POINTERS FOLLOWING LAST POINTER NOT ZERO, FILE AA.TSK
POTENTIAL LOST DATA ON FILE DLIB.OBJ
END OF TASK 0

DISCCHECK XX-YY
DEVICE NOT DISC
END OF TASK 1

DISCCHECK XX-YY
BAD PACK -- REINITIALIZE
END OF TASK 0

DISCCHECK XX-YY
INVALID FILE TYPE, FILE CUP.OUT
FILE CUP.OUT DELETED
POTENTIAL LOST DATA ON FILE TSTCHN.CHA
BAD FILENAME 4355525240202020.202020
FILE CURR@.DELETED
INCORRECT BLOCK COUNT ON FILE SCRT
FILE SCRT.DELETED
BAD DIRECTORY-CHAIN BROKEN
END OF TASK 0
```

CHAPTER 8
OS/16 BACKUP UTILITY

INTRODUCTION

The OS/16 Disc Backup Utility provides a fast method of saving files. The files may be transferred from disc to disc, or disc to magnetic tape or tape to disc. Either all files or selected files may be saved and restored. Optionally, the data on the backup device may be verified.

The tape format of OS/16 BACKUP is nearly compatible with DISC COMPRESS. Therefore, tapes created with DISC COMPRESS can be restored using BACKUP with the following exception:

OS/16 BACKUP does not handle multiple tape volumes created by COMPRESS. However, files can be selectively restored or verified provided that all requested files are contained on one tape, and that no specified file is continued on the next tape.

FEATURES

The Disc Backup provides the facilities to:

- transfer files directly from one disc to another; the output disc serves as a backup of the original.
- transfer files from an input disc to an intermediate magnetic tape device; the magnetic tapes are used as a backup.
- restore the data from the intermediate device to an output disc.
- verify data copied during the backup operation.
- verify data which has been copied during a previous execution of the program.
- selectively dump individual files from disc to disc or from disc to tape.
- selectively restore files from tape to disc.
- transfer files created under OS/32 to OS/16 media or vice versa.
- delete an already existent file on the output disc.
- enter additional starting arguments from a specified device or file.
- transfer all files created before a given date.
- transfer all files created after a given date.

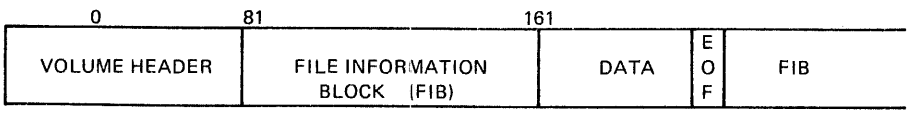
Disc to Disc Backup

When transferring files from one disc to another, BACKUP writes the files onto the destination disc in a contiguous manner as long as there are no bad sectors. This minimizes access time on the destination disc for indexed files, and maximizes the amount of contiguous free space on the destination disc.

BACKUP can copy the files onto an empty destination disc or on a disc already containing some files.

Disc to Magnetic Tape Backup

The output tape created by BACKUP when copying files onto magnetic tape is in the following format.



Volume Header — Contains the following fields:

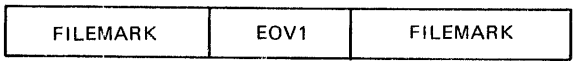
- Disc volume (volume name of input disc).
- Sequence number of tape, starting with number one.
- Number of blocks written on the preceding tape.
- Size of buffer used to transfer data.

File Information Block — The File Information Block provides information relative to the file. It precedes the data of each file.

Data — The disc block image of the data on the file.

EOF — End of file mark.

EOT Handling — An end of volume label is written at the end of each tape volume. The format of this label is:



This end of volume indicator allows Selective Restore to span multiple tapes. It is also used by BACKUP to ensure that all records written to a tape are restored properly.

NOTE

If a very large buffer size is specified in the Start command, the user must make certain that the tape has a sufficient length of trailer following the EOT marker or the tape may run off the reel in an attempt to write the last record.

SYSTEM REQUIREMENTS

The Disc Backup Utility requires:

- approximately 18KB of memory plus additional memory required for buffers.
- a console device.
- at least one currently supported disc device and an additional disc or magnetic tape.

BACKUP uses any additional memory available up to CTOP to expand its buffers. Program execution times are a function of memory, and decrease as memory increases.

OPERATING INSTRUCTIONS

The OS/16 BACKUP operates as an Executive Task (E-TASK).

All discs used by the Disc Backup Utility must be marked on-line. It is possible for the input disc to be marked on-line protected, but this is not required. If the input disc is on-line protected, users may read from, but not write to, any files on the volume. If the input disc is not protected, users may read from, and write to, all files on the volume. If Backup attempts to copy a file which is currently assigned with write privileges, a message is output indicating the file cannot be copied. If option SKIP is in effect, the program skips to the next file without pausing. If option SKIP is not in effect, the program pauses after logging the message. At this point, the condition may be corrected by closing the file. When the program is continued, it attempts to copy the same file.

The integrity of all files is assumed. To guarantee successful execution of the program, the output disc must be either initialized prior to compression or, if files are to be restored in selective mode, the disc must be in a valid state. Initialization ensures that any bad sectors on the disc are avoided during the program's operation and that all file entries are removed from the disc's directory. The integrity of a disc is ensured by executing the Disc Integrity Check Utility.

NOTE

BACKUP does not save temporary or spool files. All filenames are output to the list device as they are copied by BACKUP. In this way, the operator is provided with a log of the files contained on a given tape.

To execute BACKUP:

1. Load BACKUP as follows:

```
LD BACKUP
T .BG
OP ET
```

2. If an empty disc is used as an output device, it must be already initialized using the OS/16 INITIALIZE command.

3. Mark the disc used as input on-line, (optionally) protected using the OS MARK command, as follows:

```
MARK fd:,ON
MARK fd:,ON,PROTECT
```
or

4. Mark the disc used as output on-line using the OS MARK command, as follows:

```
MARK fd:,ON
```

5. The OS START command is used to start the Disc Backup program, and to provide parameters specifying the desired operations.

The following keywords are recognized as valid arguments in the START command and may be specified in any order, separated by commas:

IN=FD: (required) Where FD is the input device to the program. This device is assigned for Sharable Read Only to LU 1.

OUT=FD: (required) Where FD is the output device. The program assigns this device for Sharable Read/Write to LU 2.

LIST=FD: (optional in START command but a required assignment) Where FD is the list device for filenames and messages. The list device may be preassigned by the user to LU 7. If entered in the START command, the list device is assigned for Sharable Write Only to LU 7.

SIZE=n (optional) Where n is the buffer size in KB requested for Disc to Tape Operation. The default size is 12K. The n is a decimal number with optional decimal places (e.g., 16.50).

VERIFY (optional) Data on the input and output device is verified after all files have been copied. If the data does not verify, the non-verifying records from both files are output to the list device along with an error message.

DELETE (/NODATECHECK) (optional) Any file already existing on the destination disc whose date of creation is older than that of the corresponding file on the input device is deleted. If NODATECHECK is specified, the file on the destination disc is deleted regardless of the date of creation. The list of files copied indicates if a file was replaced.

COMMAND=fd — Where fd is the input device from which additional arguments are to be taken. Command may appear anywhere in the START argument list. After processing all the START arguments, additional arguments are read from the fd. The arguments are the same as in the START argument list and are processed until an END is encountered.

VO — (optional) Data on the input and output device is verified only. No copy operation is performed. Any records that do not verify are output to the list device.

ABORT — (optional) When this keyword is specified in the START command, the program terminates if non-ZERO status is returned following an I/O operation or when attempting to allocate or assign a file. If ABORT is not specified, the task PAUSEs. (Default = PAUSE)

SINCE= mmm/dd/yy (dd/mmm/yy) — (optional) Where (month) mmm is alphabetic, (day) dd and (year) yy are decimal. All files created on or after the date given are transferred. (All system files are transferred regardless of the date given.)

ARCHIVE= mmm/dd/yy (dd/mmm/yy) — (optional) Where mmm, dd and yy are as defined above. All files created on or before the date given are transferred. (All system files are transferred regardless of the date given.)

SKIP — (optional) When this option is specified in the START command, any files which cannot be successfully assigned on the input disc, or allocated and assigned on the output disc by BACKUP are not transferred. The files are identified in an error message; the program skips to the next file without pausing. If any files were skipped during the copy operation, a message is generated and verify is not performed. (Default = PAUSE)

SELECT=FD: — (optional) This option can be used to selectively copy or verify files from disc to disc, or to selectively transfer or verify files from disc to tape; and to restore or verify them from tape to disc. FD is the file descriptor from which filenames to be restored or verified can be entered. BACKUP assigns this FD to LU 5. The SELECT list is terminated by a */ or ./ record. As much memory as is available after allocation of buffers will be used for the SELECT list. A minimum of 40 entries will always be available.

The format used in specifying filenames is as follows:

filename (or) . ext (or) / accnt (or)

The filename, ext and accnt fields are optional. The accnt field may specify an account number, or the characters S or - only. If the accnt number field is omitted, system files are assumed (account number zero).

The filename and ext field may contain a - to indicate a match on any filename or extension. For example:

A.B/S

or

A.B — specifies that system file A.B is to be copied.

A.- — specifies that all system files with filename A, any extension, are to be copied.

-.B — specifies that all system files with any filename and extension B are to be copied.

Private user files may be backed up by specifying the account number. For example:

A.B/5 — specifies that file A.B of account 5 is to be copied.

-.-/5 — specifies that all files in account number 5 are to be copied.

or

-/5

Filenames are read until either the maximum number of files that can be selected in one operation has been reached, or an End of Block indicator (/* or ./) has been found. After all filenames have been entered, BACKUP starts the requested operation.

When files are to be selectively restored from magnetic tape to disc, the program may be started with a tape other than number 1. In this manner any tapes prior to the tape containing the first file to be restored or verified need not be passed. Any succeeding tapes, however, must be in sequence.

When started, Backup prints the message OS/16 BACKUP XX-YY and proceeds with the requested operation. If any errors or abnormal conditions occur, a message is logged on the System Console and printed on the list device and the appropriate action is taken. A list of all error messages and resulting actions is found at the end of this document. Upon successful completion, the following message is printed:

END OF TASK 0

EXAMPLES:

Examples of START command for DISC to DISC operation:

START ,IN=DSC1:,OUT=DSC2:,LIST=PR:	copy DSC1 to DSC2, no options
START ,IN=DSC2:,OUT=DSC1:,VERIFY,A,LIST=PR:	copy DSC2 to DSC1, verify, stop on errors
ASSIGN 7,PR: START ,OUT=DSC1:,IN=DSC2:,VE	preassign List Device copy DSC2 to DSC1, verify

Examples of START command for DISC to MAGNETIC TAPE and MAGNETIC TAPE to DISC operation:

START ,IN=DSC1:,OUT=MAG1:,LIST=PR:	copy disc to tape, no options
START ,IN=DSC1:,OUT=MAG1:,LIST=PR:, SIZ=4.5,VE,A	copy disc to tape size=4.5KB, Verify, stop on errors
START ,VE,IN=DSC1:,OUT=MAG1:,LIST=PR:	copy disc to tape, verify
START ,IN=MAG1:,OUT=DSC2:,LIST=PR:,VE	restore, verify
START ,IN=MAG1:,OUT=DSC2:,LIST=PR:,A,VE	restore, verify, abort
START ,IN=MAG1:,OUT=DSC2:,LIST=PR:	restore
START ,IN=MAG1:,OUT=DSC2:,LI=PR:,VO	verify files between MAG1: and DSC2: but do not copy files
START ,IN=DSC5:,OUT=MAG1:,L=PR:,SEL=CON:	selectively restore files from DSC5: to MAG1:, read filenames from CON:
START ,IN=DSC1:,OUT=MAG1:,L=PR:,VO,SEL=CON:	selectively verify without copy files between DSC1: and MAG1:, read filenames from CON:

MESSAGES OUTPUT BY THE PROGRAM

MESSAGE	MEANING	PROGRAM ACTION
1. OS/16 BACKUP XX-YY	The program is operational. XX is the program's revision level. YY is the update level within the revision.	continue
2. OPTION VERIFY	Program started verify routine.	continue
3. PLEASE MOUNT TAPE NUMBER XX	This message is generated if the end of a tape is reached before all files have been copied or verified; or at the start of the verify routine when the tape currently mounted is not the first tape.	pause
4. INVALID TAPE VOLUME XXXX, EXPECTING XXXX	For multi-volume tapes, when the currently mounted tape has not been created from the same input disc as the previous tape.	pause
5. TAPE OUT OF SEQUENCE, SEQU=XX	For multi-volume tape to disc operation where XX is the sequence number found on the volume label of the currently mounted tape; e.g., if the first tape mounted is not tape number one during a normal restore operation.	pause
6. INCORRECT NUMBER OF RECORDS TRANSFERRED	For multi-volume tape to disc operation, when the number of data blocks written on the previous tape during disc to tape operation does not equal the number of data blocks read during tape to disc operation.	abort

7.	MARK INPUT DISC ON	Input disc has not been marked ON.	pause
8.	MARK OUTPUT DISC ON	Output disc is OFFLINE.	pause
9.	FD-ERR	Invalid file descriptor in START command.	abort
10.	FORM-ERR	Syntax error in START command.	abort
11.	LU XX UNASSIGNED	Input, Output or List device have not been assigned.	pause
12.	INVALID DEVICE CODE	Device code is below 48, or magnetic tape is specified as both Input and Output device.	abort
13.	INPUT DISC CONTAINS NO FILES	No directory found on Input disc.	abort
14.	I/O ERROR LU=XX STATUS=YY ON FD: fn.ext	An I/O Error is encountered during an SVC 1 Read or Write operation on any Input or Output device or file. XX is the logical unit assigned at the time of the I/O, YY is the error status.	pause (abort if A option)
15.	INVALID FILE TYPE, FILE fn FILE fn NOT TRANSFERRED	File type not contiguous or indexed.	continue
16.	ASSIGN(DELETE)ERROR FILE fn:message	Bad status encountered while: a) trying to assign a device (START command). b) attempting to allocate, delete, or assign a file. "Message" specifies the type of error depending on returned SVC 7 status.	abort pause (abort if A option; continue if SKIP option)
17.	SYNTAX ERROR fn	Invalid syntax in filename for Selective Restore	continue
18.	INSUFFICIENT MEMORY	Not enough memory available. Reload the program into a larger segment and restart.	abort
19.	SELECTIVE RESTORE: MAXIMUM ENTRIES=XXXX	Mode is selective restore. XXXX is the maximum number of filenames that can be entered.	continue
20.	SELECTIVE VERIFY MAXIMUM ENTRIES=XXXX	Mode is selective verify only. XXXX is the maximum number of filenames that can be entered.	continue
21.	ENTER FILENAMES TO BE COPIED	Program request for filenames that are to be restored. If LU 5 is assigned to the console, a prompt is output.	wait for input on LU 5
22.	ENTER FILENAMES TO BE VERIFIED	Program request for filenames that are to be verified only. If LU 5 is assigned to the console, a prompt is output.	wait for input on LU 5
23.	SELECTED FILES EXCEED MAXIMUM	The maximum number of files allowed during Selective Restore/verify has been exceeded.	continue with requested operation
24.	SELECTED FILES NOT COPIED fn fn ⋮	This message is output after a Selective Restore operation if any of the specified files have not been found on the disc or tape. All filenames not processed are listed following this message.	continue
25.	SELECTED FILES NOT VERIFIED fn fn ⋮	This message is output after a Selective Verify Operation if any of the specified files have not been found on the disc or tape. The filenames follow this message.	continue
26.	SKIP IN EFFECT VERIFY IGNORED	Files were skipped during the copy operation. Verify cannot be performed.	End of Task

27.	NON-VERIFY:FILE vn LOGICAL UNIT X: RECORD NUMBER XXXX	Data in file fn does not verify.	continue to verify next record if chained or indexed file, if contiguous or indexed file, or verify next file if contiguous file
28.	DATE-ERR	Date given is invalid.	abort
29.	FILE REPLACED (mmm/dd/yy)	Informative message output after the name of any file that was deleted and replaced. If NODATECHECK is not specified, the date of creation of the replaced file is also output.	continue
30.	DEVICE NOT AVAILABLE FOR EXCLUSIVE USE: FD	Device fd cannot be assigned due to a conflict in access privileges.	pause
31.	SET DATE AND TIME	Informative message output if system supports clock, and date and time have not already been set.	pause

CHAPTER 9
OUTPUT SPOOLING

FUNCTIONAL DESCRIPTION

Output spooling allows more than one task to be assigned to a print device simultaneously. Data to be printed is written to disc files where it is then copied by the Spooler to the available printer on a first-in/first-out basis.

To use the output Spooler, the user assigns any logical units to be printed to the pseudo print device defined by the PSEUDO sysgen statement. As soon as the logical unit is closed, the Spooler automatically prints the results. Printing may be delayed because of a backlog to the printer.

There is no limit to the number of tasks or logical units that may be assigned to the pseudo print device. After the user makes a logical unit assignment, the following internally occurs:

- The operating system automatically intercepts all assignments to the pseudo printer and allocates an indexed file, called a Spool file, on the Spool volume.
- Subsequent write calls cause data to be written to this file and not to the printer. The Spooler supports both image and formatted writes.

When the LU assigned to the Spool file is closed, the file is placed into the Spooler's print queue. The print queue is maintained as a file on the spool volume. If there is an entry on the print queue, the output Spooler begins printing and stays active as long as there is something on the queue.

PRINT Command

The operator command PRINT can be used to invoke the Spooler to print a disc file.

If the user desires multiple copies of a file, data may be written to any disc file and a subsequent PRINT command may be issued to print multiple copies.

The user must ensure that sufficient disc space is available to accommodate output spooling. The user task is responsible for handling EOM status while writing to spool files within its own I/O error recovery routines.

A header page with the following information precedes the printing of each spool file:

user-id or filename
user account number
time of day
date

VOLUME Command and the SPOOL Sysgen Statement

The disc volume (spool volume) used for the queue file and the spool files themselves may be specified with the VOLUME command or the SPOOL sysgen statement. The queue file is automatically created when execution of the Spooler task begins, unless a queue file already resides on the spool volume. The Spooler will use a pre-existing queue file, thus permitting interruption of Spooler execution without loss of print files.

Only print files submitted to the Spooler while it is executing are queued and subsequently printed. Therefore, it is essential that the system console operator load and initiate execution of the Spooler before any users submit print files. If the Spooler is not executing, no action is taken on print files submitted with the PRINT command and an error message is displayed on the terminal indicating that the Spooler was not able to service the PRINT request. The PRINT command can be reentered after the Spooler begins executing.

Files submitted for printing by means of assignment to the pseudo device will not be automatically printed if the files are closed when the Spooler is not executing. Status is returned to the program closing the file, indicating that an error has occurred. The file can then be resubmitted for printing using the PRINT command when the Spooler is executing.

OPERATING INSTRUCTIONS

The Spooler is provided with the OS/16 MT2 Package. It must be loaded with the task-id SPOOLER. Example:

LOAD SPOOLER (i.e., LOAD SPOOLER,SPOOLER.TSK)

Starting the Spooler

The START command to the SPOOLER task indicates which print device is to be assigned exclusively for the spooling operation. The format of the START command is:

START,PR=fd,VOL=fd [,LETTER=fd][,M=n]

where:

PR	identifies the print device on which the print files should be listed.
VOL	identifies the spool volume specified in the most recent VOLUME command or the spool volume specified in the SPOOL sysgen statement if the name of the spool volume had not been modified with a VOLUME command.
LETTER	identifies the volume that contains the file LETTER.IN. This file contains information necessary for creating the large letters on each listing's heading. This file is supplied with OS/16. If LETTER is not specified, the large letters are not used on the listing headers; instead, the heading is printed using normal size letters.
M	specifies the number of message buffers that the Spooler should use to communicate with the command processor and file manager. The number of message buffers required depends upon the spooling activity at the specific installation. There should be a message buffer for each Spooler request (PRINT command or closing of a pseudo print device) that can occur simultaneously. At least two buffers should be used; the default is four.

The PR and VOL parameters are required; LETTER and M are optional. All of the keywords can be abbreviated; only the first letter has to be used, e.g., V=MT16:

Examples of the START sequence:

```
LO SPOOLER      load the spooler task
TA SPOOLER      set the current task
```

ST,PR=PR:,VOL=MT16:
where:
PR: is the print device.
MT16: is the spool volume.
Small letters should be used for list headings and four message buffers should be used.

ST,PR=PR:,VOL=MT16:,LETTER=MT16:
where:
PR: is the print device.
MT16: is the spool volume.
Large letters should be used for list headings and four message buffers should be used.

ST,PR=PR:,VOL=MT16:,LETTER=MT16:,M=2
where:
PR: is the print device.
MT16: is the spool volume.
Large letters should be used for list headings and two message buffers should be used.

NOTE

The Spooler assigns the print device for exclusive access (EWO) for the duration of its execution.

The file currently being printed may be aborted by passing the Spooler and closing the logical unit assigned to the input file (LU 1). Execution can then be resumed with a CONTINUE command.

Deleting a file queued for printing aborts printing of a file that is queued but not being printed. It is also possible to restart the printing of the file by:

- pausing the spooler task,
- rewinding the input file (LU 1), and
- resuming execution with the CONTINUE command.

ERROR MESSAGES

The errors reported by the Spooler are:

ILLEGAL OPTION UT

Cause: OPTION ET was not specified prior to starting the Spooler.
User Action: Restart the Spooler after specifying OPTION ET.

ILLEGAL OPTION COMP

Cause: OPTION COMP was specified prior to starting the Spooler.
User Action: Restart the Spooler without specifying OPTION COMP.

INSUFFICIENT STORAGE

Cause: The partition that the Spooler was loaded in was not large enough.
User Action: Load the Spooler in a larger partition. Use fewer message buffers on the list heading. Each message buffer requires an additional 78 bytes. Using large letters requires an additional 256 bytes.

ASSIGN ERROR ON {PRINT / LETTER / QUEUE} FILE

Cause: The print device, letter file, or Spool queue file could not be successfully assigned.
User Action: Enter the correct parameters on the START command and reexecute the Spooler.

START COMMAND ERROR,KEYWORD={ILLEGL / PR / VOL / LETTER / M}

Cause: The value entered for the keyword identified in the message was incorrect. If the word ILLEGL is displayed in the message, an invalid keyword was specified on the START command.

User Action: Correct the keyword in error, ensuring that it is not specified more than once in the START command, and reexecute the Spooler.

SPOOLER QUEUE OVERFLOW

Cause: A file was submitted to the Spooler for printing with a PRINT command or a file assigned to the pseudo print device was closed when there were no entries available in the queue.

User Action: Wait for the Spooler to print a few files, thereby creating available entries in its queue, before submitting more print files. The file that was being submitted when the error occurred must be resubmitted for printing with a PRINT command.

I/O ERROR ON voln:filename.ext

Cause: An I/O error occurred when attempting to read from or write to the file identified in the message.
User Action: Ensure that all of the parameters entered on the START command are correct. If they are not, correct the START command and reexecute the Spooler. If the file in error is the print device, the Spooler automatically pauses, awaiting correction of the problem. The printer may be out of paper or may have been taken off line for some reason during a write operation. Spooler execution can be resumed with a CONTINUE command after the error is corrected. Execution can also be aborted with a CANCEL command.

CONFIGURATION REQUIREMENTS

For operation, the Spooler requires:

16-bit Processor
Any OS supported disc device(s)
Any OS supported print device
3.25kb of memory above the OS size, plus an optional 0.25kb buffer if the large headings are desired
OS/16 MT2 system with at least one foreground partition

Example:

The following example sequence establishes the Spooler for operation:

```
*LD TET                  Load the task establisher
*TA .BG                  Set the current task
*AS 5,CON:;AS 7,CON:     Assign the console
*AL TS,IN;AS 4,TS        Allocate and assign scratch file
*ST                      Start TET
*.BG >ES TA              Establish as a task
*.BG >BI E000            Set the bias
*.BG >OPT ET             Make it an E-task
*.BG > GET300            Leave room for the letter and message buffers
*.BG >INCLUDE SPOOLER    Read Spooler object
*.BG >BUI TA,SPOOLER     Build the task
*.BG >MAP CON:           Take a map
*.BG >END                End the establishment
*.BG:END OF TASK 0
```

CHAPTER 10
USING UTILITY SOFTWARE

INTRODUCTION

A complete software system is composed of several parts. These include the Operating System, Language Processors such as CAL and FORTRAN, and Utility Programs.

This guide describes the most commonly used utility programs that are capable of running in an OS/16 MT2 environment. In some cases, there exists an overlap between the functions of one utility program and those of another. This presents the user with a choice of which utility program to use when confronted with a given problem. This guide attempts to make this choice somewhat more simple, by describing, in brief, each utility program and its intended uses. The utilities described in this chapter can be run in the Background or established using TET/16 and run in a Foreground environment.

The user is referred to the document listed with each utility description for a more detailed discussion of that utility.

COMPATIBILITY BETWEEN OS/16MT2 AND EXISTING UTILITIES

OS/16 MT2 is upwards compatible with earlier 16-bit operating systems for common SVC functions, with the exception of SVC 1 random I/O and SVC 5 fetch overlay calls. All utilities and user programs designed prior to OS/16 MT2 and all compiled FORTRAN programs must be run with 'OPTION COMPATIBLE' set, until such time as they are modified to use the new form of the SVC parameter blocks. See Appendix 11.

TEXT MANIPULATION UTILITIES

Text manipulation utilities include text editors, copiers, etc. These programs are ASCII-oriented, but some can handle binary text. The programs discussed in this section are: OS EDIT, Source Updater, and OS COPY.

OS EDIT AND OS/16 EDIT

OS EDIT, program number 03-063, is a general-purpose ASCII-oriented text editor. It is principally designed for interactive use. but has some facilities allowing it to be used in a batch environment if necessary. OS EDIT is line- and character-oriented. It has powerful facilities for pattern-matching and string-replacement. OS/16 EDIT, program number 03-169, is a disc based version of the editor.

OS EDIT may be used to edit any ASCII text file. It is not restricted to any particular text format; therefore it can be used to edit CAL or FORTRAN programs, or any other sort of text. Refer to *OS EDIT User's Manual,* Publication Number 29-373 or *OS/16 EDIT User's Manual,* Publication Number 29-637.

An example of OS EDIT's use is illustrated in Figure 10-1.

```
*TASK.BG
*LD OSEDIT                    ;*LOAD THE EDITOR
*ASSIGN 1, INPUT              ;*ASSIGN INPUT FILE
*ASSIGN 2, OUTPUT             ;*AND OUTPUT FILE
*ASSIGN 3, CON:               ;*ASSIGN LIST DEVICE
*ASSIGN 4, NULL:              ;*NO BATCH MODE
*ASSIGN 5, CON:               ;*ASSIGN COMMAND DEVICE
*ASSIGN 6, CON:               ;*ASSIGN ERROR MESSAGE DEVICE

*START
OS EDIT
> CR                          Create a file
> NOW IS THE TIME
> FOR ALL MEN
> TO COME TO THE AYD
> OF THE PARTY
> /*                          Terminate text input
> PR 2                        Print line 2
2 FOR ALL MEN
> 2.1 AND WOMEN               Insert a line
> PR 2,3                      Print lines 2 to 3
2 FOR ALL MEN
2.10 AND WOMEN
3 TO COME TO THE AYD
> CH /AYD/AID/                Change AYD to AID
> ML 1                        Move to top of buffer
> PL /ALL MEN/                Print line containing ALL MEN
```

```
2 FOR ALL MEN
>CH /MEN/GOOD MEN/          Change MEN to GOOD MEN
>PR                         Print all text
1 NOW IS THE TIME
2 FOR ALL GOOD MEN
2.10 AND WOMEN
3. TO COME TO THE AID
4 OF THE PARTY.
>EN                         Output edit buffer and end task
END OF TASK 0
*
```

Figure 10-1. OS EDIT Example

Source Updater

The Source Updater, 03-090, is a special-purpose text editor and updater designed specifically for use with CAL source files. It is principally batch-oriented and is designed so that it can be run unattended, although it can be used interactively if necessary.

Since the Source Updater is specifically CAL-oriented, it can make use of the sequence numbers in a CAL source file for efficient file search. In addition, commands are provided for verification and listing of source files.

Refer to the Source Updater Users Manual, Publication Number 29-630.

An example of this program's use is illustrated in Figure 10-2.

```
*LDBG SOURCEUP                  ;*LOAD SOURCE UPDATER
*TASK .BG
*ASSIGN 3,PR:                   ;*ASSIGN LIST DEVICE
*ASSIGN 5,CON:                  ;*ASSIGN COMMAND INPUT DEVICE
*START
SOURCE UPDATER
>UPDATE CR:,OLDFILE.CAL,NEWFILE.CAL
```

(This command causes the change deck in device CR: to be applied to the old source file in file OLDFILE.CAL, with the output going to NEWFILE.CAL. The change deck might include the following statements:)

```
INSERT  ABC01450                Insert following sequence ABC01450
LAB01   LHI    R15,X'203'
        STH    R15,ERRCODE
/*
DELETE  ABC01680, ABC01695      Delete some statements
MODIFY  ABC02300                Modify a statement
        LE     FR0,PI
SELECT                          Insert or modify as required
        ATL    R12,SYSQUE       ABC03420
        STH    R5,TEMP          ABC03530
        SER    FR0,FR2          ABC05565
LAB3    EQU    *                ABC06780
/*
ENDUP                           End of update

(Now control reverts to the command device)

EXCEPTION OLDFILE.CAL,NEWFILE.CAL
REWIND NEWFILE.CAL
COPY NEWFILE.CAL,NEWFILE.BAK
VERIFY NEWFILE.CAL,NEWFILE.BAK
END
```

Figure 10-2. Source Updater Example

The previous statements perform the following actions:

1. Generate a listing of all differences between the old and new files;
2. Rewind the new file and copy it to a backup file;
3. Compare both files to make sure they are the same;
4. End processing.

OS COPY

OS COPY, program number 03-056, is a utility program that provides file duplication capabilities. It is able to copy both ASCII and binary files of fixed-length records. When using cards or magnetic tape media, it can copy entire "volumes" of files with one command. (Such a "volume", not to be confused with the OS/16 direct-access volume, is a file containing a sequence of files separated by file marks.)

Refer to *OS COPY Program Description,* Publication Number 03-056A15. OS COPY is a batch-oriented program, but can be used interactively if necessary. An example of its use is illustrated in Figure 10-3.

```
*LOAD OSCOPY                    ;*LOAD COPIER
*TASK OSCOPY
*ASSIGN 1,MAG1:                 ;*ASSIGN INPUT DEVICE
*ASSIGN 2,MAG2:                 ;*AND OUTPUT DEVICE
*ASSIGN 3,PR:                   ;*LIST DEVICE
*ASSIGN 5,CON:                  ;*AND COMMAND INPUT DEVICE
*START
OS COPY
>FIXD                           Set for fixed-length records
>CPYB PROGA                     Copy program PROGA (binary)
>CPYB PROGB,,ALL                Copy program PROGB and all succeeding
                                programs; rewind input and output
>RWD 1
>RWD 2
>VERB PROGA                     Verify PROGA
>VERB PROGB,,ALL                Verify remainder of tape
>END                            End of task
END OF TASK 0
*
```

Figure 10-3. OSCOPY Example

Copying functions may be performed by several programs other than the OS COPY program. The OS/16 Library Loader can copy binary files (in object format only) on any medium; however, it does not contain verification capabilities. The OS Source Updater and OS EDIT can copy ASCII files on any medium; however, the OS Source Updater is restricted to 80 byte records, and the OS EDIT program has no verification capabilities.

LOADERS

Under the category of loaders come all those programs that process the output of language Processors so that this output can be executed, either at the current time or at some later time. Programs discussed in this section are the OS/16 Library Loader, TET/16, the OS/16 MT2 Direct Access Boot Loader and the Boot Puncher.

OS/16 Library Loader

The OS/16 Library Loader, program number 03-030, is a linking loader with library manipulation capabilities. It is able to build load modules for later execution under OS/16. It has a full complement of commands for building and modifying object-format subroutine libraries. In addition, it is able to copy object-format programs using any available media under OS/16.

Refer to *16-Bit Loader Description Manual,* Publication Number 28-321. As a load module builder, it has full linking and library-editing capabilities. It builds absolute load modules only, but can handle either absolute or relocatable input text. When building a load module, leave room for the UDL between the start of the partition and the start of the module. Examples of the library loader's use are illustrated in Figures 10-4 and 10-5.

```
*LOAD LIBLDR                    Load the Library Loader
*TASK LIBLDR
*ASSIGN 3,PR:                   Assign list device
*ASSIGN 5,CON:                  Command input
*ASSIGN 1,PROGRAM.OBJ           Program input
*ASSIGN 2,SUBR.OBJ              Subroutine to be linked
*ASSIGN 4,LIBRARY.OBJ           Subroutine Library to be edited
*ASSIGN 6,PROGRAM.OUT           Load module output
*START                          Start it
*LOADER-R06
>OUT 6                          Define load module output LU
>BI 8024                        Set the load module origin
>LOAD 1                         Now "load" the program
>LINK 2                         Link in the subroutine
>EDIT 4                         and edit the library
>XOUT                           Close the output
>MAP 3                          and write the load map to the printer
>END                            End of task
  END OF TASK 0
  *
```

Figure 10-4. OS/16 Library Loader Module Building Example

Figure 10-4 shows the use of the OS/16 Library Loader as a load module builder. Figure 10-5 shows its use for library manipulation.

```
*LOAD LIBLDR                    Load Library Loader
*TASK LIBLDR
*ASSIGN 5,CON:                  Assign command input
*ASSIGN 3,PR:                   and listing
*ASSIGN 1,OLDLIB.OBJ            Old master
*ASSIGN 2,SUB12.OBJ             New subroutine
*ASSIGN 4,NEWLIB.OBJ            and new master
*START                          and start
LOADER-R06
>DUPE 104 SUB12                 Dupe from beginning to new subroutine
>COPY 204 SUB12                 Now copy subroutine from change file
>FIND 1 SUB13                   Find the next program on the input file
>DUPE 104                       and dupe the rest of the file
>RW 4                           Rewind output
>TABLE 403                      Dump a table-of-contents to the printer
>END                            End of task
END OF TASK 0
*
```

Figure 10-5. OS/16 Library Loader Library Maintenance Example

TET/16

The OS/16 MT2 Task Establisher (TET/16) performs task building functions for OS/16 MT2. Unlike the OS/16 object loaders, which load programs in object format, the memory image loader loads programs in memory image format, with data in a Loader Information Block prefixed to the image file to pass information to the loader. TET/16 builds these image files. Refer to Chapter 5 of this manual.

TET/16 has facilities for handling overlays and Library Files, as well as those for handling ordinary task building. It can be used in both interactive and batch modes of operation. A typical example of the use of TET/16 to establish a FORTRAN task is illustrated in Figure 10-6.

```
*OP NOCOMP                    ;*ENSURE NOT IN 'COMPATIBLE' MODE
*LOAD TET                     ;*LOAD TET
*TASK TET
*ASSIGN 5,CON:                ;*ASSIGN COMMAND INPUT
*ASSIGN 7,CON:                ;*ASSIGN ERROR MESSAGE OUTPUT
*ASSIGN 4,TET.SCR             ;*ASSIGN SCRATCH FILE
*ASSIGN 1,PROGRAM.OBJ         ;*ASSIGN OBJECT FILE
*ASSIGN 2,PROGRAM.TSK         ;*ASSIGN TASK IMAGE OUTPUT FILE
*LOAD TET
*START                        ;*AND START IT
TET/16 00-00
>ESTABLISH TASK               Set TET to establish a task
>OP COMP                      Set option compatible
>BI 8500                      Set start of partition
>INCLUDE                      Include PROGRAM.OBJ (on LU 1)
>INCLUDE SUBR.OBJ             Link in subroutine on file SUBR.OBJ
>RESOLVE FTNRTL,E000          Resolve references to the Resident Library at E000
>EDIT SUBLIB.OBJ              Get further subroutines from file SUBLIB.OBJ
>PRIORITY 200,50              Set running and max priority for task
>GET 1400                     Give the task extra memory
>BUILD TASK                   Put the task image output on file PROGRAM.TSK
>MAP PR:                      Put a map of the established task on PR:
>END                          End of task
END OF TASK 0
*
```

Figure 10-6. FORTRAN Task Example

OS/16 Direct Access Boot Loader

The OS/16 Direct Access Boot Loader is a program that allows an established copy of OS/16 MT2 to be boot loaded from disc. Its use is fully described in Chapter 2 of this manual, Loading the OS.

OS/16 Boot Puncher

The OS/16 Boot Puncher, program number 03-108, is used to create a bootstrap tape of a program which may be loaded by the 50 Sequence. Its input is an object format module of the program. Output is one record containing a memory image of the program. This program is used to create bootstrap tapes of the OS/16 Direct Access Boot Loader. An example of its use is given in Figure 10-7. Refer to Appendix 13 for a description of boot puncher error messages.

```
*LDBG BPCH16                  LOAD BOOT PUNCHER INTO BACKGROUND
*TA .BG                       SET TASK TO BACKGROUND
*ST,BOOT16.OBJ,PTRP:          START IN=BOOT16.OBJ,OUT=PTRP:
OS/16 MT2 BOOTSTRAP PUNCHER 00-00
END OF TASK 0
```

Figure 10-7. OS/16 Boot Puncher Example

SYSTEM MAINTENANCE UTILITIES

Under this heading fall all utility programs whose principal function is to act as extensions of the operating system itself, and to perform tasks closely related with the operating system.

OS/16 Configuration Utility Program (CUP/16)

This program is used to perform tailored System Generation for the OS/16 MT2 Operating System.

The operation of this program is fully discussed in the *OS/16 MT2 System Planning and Configuration Manual.*

Disc Integrity Check

OS/16 maintains directories for each Disc Volume on the disc itself. In case of a system crash or system reload when Disc Files are open, these directories may be left in a faulty state. This condition can result in the inability to delete certain files, to assign certain files, or to release disc space which is no longer needed.

Disc Integrity Check, Program Number 03-080, is designed to examine an off-line Disc Volume and to correct any error conditions resulting from a system crash. It provides warning messages when a possible loss of data on files may have occurred as a result of the crash. It is capable of deleting any Disc Files that have erroneous directory entries or physical record chains that make them unusable. Refer to *Disc Integrity Check Description,* Chapter 7. An example of its use is illustrated in Figure 10-8.

```
*LD DISCHECK                              ;*LOAD DISC CHECK IN BACKGROUND
*MARK DSC1:,OFF                           ;*MARK THE DISC OFFLINE
*TASK .BG
*OPTIONS ET                               ;*MAKE IT AN E-TASK
*START ,DSC1:,CON:                        ;*START IT
POTENTIAL LOST DATA ON FILE FRED.XYZ
POTENTIAL LOST DATA ON FILE IRVING.PXT
END OF TASK 0
*
```

Figure 10-8. Disc Integrity Check Example

BACKUP

BACKUP/16, program number 03-239, may be used to create a backup of a Disc Volume. It may be used to compact sector allocation into a contiguous area on a Disc Volume, thus maximizing the largest contiguous free space on the volume. Refer to Chapter 8. Figure 10-9 illustrates the use of BACKUP to dump the contents of a Disc Volume to Magnetic Tape and to restore it to the same volume.

```
*TA .BG
*LD BACKUP                                LOAD BACKUP
*OP ET                                    MAKE IT AN E–TASK
*MA DSC1:,ON,PROT                         MARK DISC ON WITH PROTECT
*ST, IN=DSC1: ,OUT=MAG1: ,LIST=PR: ,VERIFY
END OF TASK 0
```

To restore:

```
*TA .BG
*OP ET                                    MAKE IT AN E–TASK
*LD BACKUP                                LOAD BACKUP
*MA DSC1:,OFF                             MARK OFF DISC
*IN DSC1:,PACK,RE                         INITIALIZE PACK
*MA DSC1:, ON
*ST, IN=MAG1: ,OUT=DSC1: ,LIST=PR: ,VERIFY
END OF TASK 0
```

Figure 10-9. BACKUP Example

PROGRAM MAINTENANCE UTILITIES

AIDS/16

AIDS/16 is an interactive debugging utility program designed to assist the user in debugging tasks in an OS/16 MT2 environment. It has powerful facilities for monitoring the execution of a program, either in real-time or in interpretive mode, and allows program modifications to be easily made at run time. Do not run more than one program with AIDS/16 at any one time. AIDS/16 may be used only in the background on extended memory systems. Refer to *AIDS/16 User's Guide*, Publication Number B29-571. An example is illustrated in Figure 10-10.

```
*TASK .BG                          ;*LOAD TASK TO BE DEBUGGED
*LD BUGGY, 8000                   ;*LOAD OS AIDS IN UPPER MEMORY
*LD AIDS,A100                     ;*ASSIGN COMMAND INPUT DEVICE
*ASSIGN 5,CON:                    ;*START AIDS/16
*START A100
AIDS/16 00-00
>BI   8000                        Set relocatable bias
>OA   1302                        Open location 1302 relocatable
1302R C8300045 LHI 3,45
>RH   C850                        Change the contents of that halfword
>IX   470                         Insert Breakpoint, display
>IX   1350                        Insert Breakpoint, display
>GO   44                          Start program
  BRK: 0470R                      First breakpoint executed
>OX   .A                          Examine general register 10
  GP(A) 0100
>GO                               Resume from breakpoint
  BRK: 1350R                      Second breakpoint executed
>IP   2040                        "Protect" all 2040 relocatable
>GO                               Resume execution
  PRO: 108AR 2040 425A
  BRK: 0470R                      Breakpoint executed
>ZX                               Remove breakpoint and protection
>ZP
>GO                               Resume program unconditionally
END OF TASK 0                     Termination of program
*
```

Figure 10-10. AIDS/16 Example

LANGUAGE PROCESSORS

The following language Processors are available under OS/16 MT2:

CAL	03-066
CAL/16	03-101
memory-based	03-101F01
disc-based	03-101F02
CAL MACRO	03-084
FORTRAN V	03-060
FORTRAN IV	03-054
BASIC	03-055
BASIC LEVEL II	03-105

CAL and CAL/16

CAL, The Common Assembly Language Assembler, provides the capability to assemble a program for both the 32-bit and the 16-bit line of Processors under OS/16. It has a full complement of user features, including conditional assembly, page control, arithmetic expressions, and support of COMMON. It contains over 70 pseudo-ops. Refer to *Common Assembler Language User's Manual,* Publication Number 29-375.

CAL/16 is a subset of CAL. It is designed for smaller memory environments and supports programs for the 16-bit machines only. Operation of CAL and CAL/16 are the same, except that CAL must be run with OPTION COMP. Refer to Chapter 6 of the manual for further information on CAL/16. Figure 10-11 illustrates the use of OS/16.

```
*TASK .BG                          Set task to background
*LD CAL16                          Load CAL/16
*ASSIGN 1,CR:                      Assign source input to card reader
*ASSIGN 2,OBJFILE                  Assign output object file
*ASSIGN 3,PR:                      Assign list output to the printer
*ASSIGN 4,SCRAT                    Assign scratch file
*ASSIGN 7,COPYFILE                 Assign source copy file
*START                             Start CAL
 CAL/16 00-00
 END OF TASK 0
```

Figure 10-11. CAL/16 Example

CAL MACRO

CAL MACRO Processor, program number 03-084, allows the user to write a series of Macro definitions and to invoke these macro definitions, causing an expansion into CAL source statements. CAL MACRO output may then be input to CAL for assembly. Macro definitions may be placed in a library or included in the input stream along with references to these definitions. Refer to *CAL MACRO Processor Reference Manual,* Publication Number 29-408. Figure 10-12 illustrates an assembly with a macro preprocessing step:

```
*TASK .BG                    LOAD CAL MACRO PROC INTO BACKGROUND
*LD CALMAC                   ASSIGN OUTPUT TO INTERMEDIATE FILE
*AS 1,EXPSRC.CAL             ASSIGN LIST DEVICE
*AS 3,PR:                    ASSIGN SOURCE INPUT TO CARD READER
*AS 6,CR:                    ASSIGN MACRO LIBRARY
*AS 7,MACLIB                 START PREPROCESSOR
*START
END OF TASK 0
RW EXPSRC.CAL,1              REPOSITION INTERMEDIATE FILE
*LD CAL16                    LOAD CAL/16
*AS 2,OBJFILE                ASSIGN OUTPUT
*START                       START ASSEMBLY (USE NO SCRATCH)
CAL/16 00-00
END OF TASK 0
```

Figure 10-12. CAL MACRO Example

FORTRAN V

The FORTRAN systems available under OS/16 are FORTRAN V and Extended FORTRAN IV. FORTRAN V outputs intermediate source that must be assembled by CAL, and established by TET before execution. Extended FORTRAN IV outputs object code that need only be established. It is designed for use in small memory systems. The FORTRAN V compiler and corresponding object programs must be run with OPTION COMP.

Figure 10-13 illustrates a sample FORTRAN V operating sequence:

```
*LD FORTRANV                 Load the FORTRAN compiler into the background
*TASK .BG
*OP COMP
*ASSIGN 1,CR:                Source Input
*ASSIGN 2,TEXT               Intermediate text output
*ASSIGN 3,PR:                Source Listing
*ASSIGN 7,PR:                Error Listing
*START                       Start Compilation
 END OF TASK 0               Compilation complete with no errors

                             (The message END OF TASK 1 is output if compila-
                             tion errors were detected)
```

Figure 10-13. FORTRAN V Example

At this time the file TEXT must be assembled by CAL as described in the previous section. Assume the object program is output on file OBJECT.

The OS/16 MT2 Task Establisher (TET/16) may be used to prepare the object program for execution. For the purposes of this example, assume a main FORTRAN program with two user subroutines in 16-bit object format on file OBJECT with a file mark written after the last user subprogram. Assume also that the FORTRAN V Level 1 Run Time Library is resident on a Direct-Access File named RTL.OBJ.

Once TET/16 is started, the first commands entered are:

```
ESTABLISH TASK
OPTIONS FLOAT,COMP
BIAS  xxxx
```

The system then responds with a prompt, whereupon the user enters:

```
INCLUDE OBJECT.
```

This causes TET/16 to read the main FORTRAN object module and link the two user subroutines; this operation is terminated by the detection of the filemark. The system again responds with a prompt, whereupon the user should enter:

```
EDIT RTL.OBJ
```

TET/16 makes one pass over the Run Time Library object module library, loading only those routines as required by the user program. Once this operation is complete, the system responds with a prompt and the user should enter:

BUILD TASK,fd

where fd is a file to contain the established task. If at this point, additional routines are still required to complete the FORTRAN task, TET/16 outputs the message:

UNDEFD SYMBOLS

The user should then take a map (see below) to determine the cause of the error. In general, once a FORTRAN program and all user subroutines are loaded and the Run Time Library is edited, no routines remain undefined.

Once the task is built, the system responds with the prompt and the user requests a memory map of the task with the command:

MAP fd

where fd is the list device. This map defines the entry points of all subprograms and gives the minimum partition size required for execution of this task.

EXTENDED FORTRAN IV

This compiler generates object code rather than intermediate source output. Tasks may be established in the same manner as the FORTRAN V example above. Operating instructions are found in the *Extended FORTRAN IV User's Guide,* Publication Number 29-336.

BASIC LEVEL II

BASIC LEVEL II is an exhanced interpreter designed to run under OS/16 or OS/32. Refer to the *Basic Level II Reference Manual,* Publication Number 29-488.

For information on running the BASIC LEVEL II Interpreter on an OS/16 MT2 system configured with no Command Processor Module, refer to the *System Planning and Configuration Guide,* Chapter 2.

CHAPTER 11
SYSTEM LIBRARIES

It is prudent to back up the information shipped in the OS/16 MT2 package so that system or operator error cannot destroy the only copy of the information. The following sections describe backup procedures for magnetic tape and disc media.

NOTES

1. These backup procedures are performed while running under the OS/16 MT2 Starter System just loaded. OS/16 MT2 Starter Systems are backed up as well as the components necessary to configure the OS/16MT2 system that is to be the result of this entire Startup procedure.

2. If a backup device called for in these procedures is not available in the configuration, another magnetic device (Disc, Cassette, etc.) can be substituted. For example, if a second Magnetic Tape is unavailable, a cassette can be substituted (MAG2: becomes CAS1:).

UNPACKAGING THE MAGNETIC TAPE PACKAGE ONTO A DISC

- Load STARTER 3 as described on Page 2-1
- Ready a Disc, insuring that Hardware Protect is turned off.
- Follow the procedure below:
  - Enter the following OS/16 commands:

```
INIT DSC1:,MT16          Initialize the Pack
MA DSC1:,ON              Mark Disc On
V MT16                   Set Default Volume
SET PAR .S/E800
FF MAG1:                 Skip Past File Mark
LD MAG1:                 Load Library Loader
BF MAG1:                 Reposition to front on Library Loader
FF MAG1:
AS 5,CON:                Assign Library Loader Units
AS 1,MAG1:
```

  - At this point use the TABLE command of the Library Loader to confirm that the object program on the tape agrees with the list in the Packaging Information document, Appendix 3. Verify from LOADER to RELLDR. To reposition to the start of the Library Loader enter:

```
PA
BF MAG1:
BF MAG1:
BF MAG1:
FF MAG1:
```

  - Allocate disc files for all package items. Use the disc file names in the Packaging Information Document, Appendix 5.

```
Example:  ALLO LIBLDR.OBJ,I,108
          ALLO EXEC16.CAL,I,80
          ALLO CUP1632.OBJ,I,126
          ALLO LETTER.IN,CO,6
```

– Use the Library Loader to copy object files from mag tape to disc.

```
Example:  AS 2,LIBLDR.OBJ          ASSIGN SOME FILES
          AS 3,DLIB16.OBJ
          AS 4,DLIB16EX.OBJ
          AS 6,CUP.OBJ
          ST                       START LIBRARY LOADER
          COPY 102                 COPY LIBRARY LOADER
          DUPE 103                 COPY DRIVER LIBRARY
          DUPE 104 CUP16           COPY EXTENDED DRIVER LIBRARY
          COPY 106                 COPY CUP16
          PA                       PAUSE LIBRARY LOADER
```

– The user may now assign other object files (TET.OBJ, etc.) and enter CO to reenter the Library Loader and continue copying with the COPY command up to the next file mark on tape.

Use OS COPY to copy the remaining files to disc:

```
EN                      End Library Loader
LD COPY                 Load OS copy from Disc
AS 2,LETTER.IN
ST
CPYB ,256,,1,6          Copy Output Spooler Table
END
AS 2,CUP1632.OBJ        Copy CUP1632 (Repeat for TET1632)
ST
CPYB ,126
END
```

– To again verify that you have the right tape contents use the OS copy FNDA command to locate the remaining modules. Verify with the Packaging Information Document, Appendix 3.

– Copy these modules.

```
AS 2,PCB16.CAL
ST
CPYA ,,ALL              Copy PCB
END
AS 2,EXEC16.CAL
ST
CPYA                    Copy Exec Source (repeat for remaining files)
  .
  .
  .
```

– To display the contents of the disc after unpackaging, enter FI. Starter 3 may now be used to perform a sysgen from disc.

MAGNETIC TAPE SYSTEM BACKUP

All the modules in the MT package are shipped on one Magnetic Tape, making this volume inconvenient to use as a working volume to perform SYSGENs and to access the utilities, since the tape must be searched from the beginning each time any module is to be accessed. Therefore, the backup procedure recommended is to produce several working volumes and leave the original volume as the backup.

The order of the modules on a Magnetic Tape is found in the Packaging Information Document.

It is recommended that 8 working volumes be created as follows, once a Starter system is loaded.

OS Tape

Mount the OS/16 MT2 tape on MAG1: and the output tape on MAG2:. Use the following commands:

```
*RW MAG1:                          Position OS/16 MT2 package tape at load point
*FF  MAG1:;FF MAG1:;FF MAG1:       Position to Library Loader
*TA   .BG
*LD  MAG1:                         Load Library Loader
*AS  1,MAG1:                       Make assignments
*AS  2,MAG2:
*AS  3,PR:
*AS  5,CON:
*ST                                Start Library Loader
 FI   1 OSCOPY                     Position to OS COPY
 END                               Exit from Library Loader
*LD  MAG1:                         Load OS COPY
*RW  MAG1:                         Position to REL Loader
*OPT COMP                          Set OPTION compatable
*ST                                Start
 CPYB    .1000                     Copy REL Loader to output
 CPYB                              Copy starter 1 to output
 CPYB                              Copy starter 3 to output
 END
```

Now the output tape contains a REL Loader, STARTER1 and STARTER3. To load the OS from this tape use the appropriate steps outlined in Chapter 2.

Object Library Tape

Leaving the input tape at its current position, mount a new output tape on MAG2:

```
*LD      MAG1:                     Load Library Loader
*RW      MAG1:                     Position to Library Loader
*FF      MAG1:
*FF      MAG1:
*FF      MAG1:
*ST

 DU      102                       Duplicate Library Loader, driver libraries on output
 DU      102 CUP16                 tape up to CUP object module
```

Utility Tape

Mount a new output tape on MAG2:

```
BF 1;BF 1                          Position input to Library Loader
FF 1

CO  102                            Copy Library Loader
FI   1  CUP16                      Position to CUP
DU  102                            Copy rest of utilities
```

To load any utility from this tape, mount tape on MAG1: for example:

```
*LD MAG1:                          Load Library Loader
*AS 1,MAG1:;AS 5,CON:
*ST
 FI  1 (utility name)              Use TABLE command of Library Loader to list utility names
 END                               Exit Library Loader
*LD MAG1:                          Load desired utility
```

Parameter and Control Block Tape

Mount a new output tape on MAG2: and enter the following to the Library Loader:

```
 BF 1                          Find source updater
 FF 1
 FI  1 (Source Updater)        See tape Table of Contents for Source Updater program label
 END                           Exit
*LD MAG1:                      Load it
*FF MAG1:                      Position to source modules
*FF MAG1:
*FF MAG1:
*ST
COPY MAG1:,MAG2:               Copy Parameters & Control Blocks
```

Source Library Tape

Mount a new output tape on MAG2:

```
COPY MAG1:,MAG2:               Copy Executive, - filemark --
COPY                           Copy File Manager, - filemark --
COPY                           Copy Command Processor, - filemark
COPY                           Copy Driver Library, -- filemark --
END                            Exit
WF MAG2:                       Write 2nd filemark as end of volume
```

CSS Package Tape

Mount a new output tape on MAG2:

```
ST
COPY MAG1:,MAG2:               Copy CSS package
END                            Exit
```

Boot Loader Tape

Mount a new output tape on MAG2:

```
*RW   MAG1:
*FF   MAG1:
*FF   MAG1:
*FF   MAG1:
*LD   MAG1:                    Load Library Loader
*AS   5,CON:
*AS   1,MAG1:
*ST
 FI   1 (Boot Puncher)         Position to Boot Puncher
 END                           Exit
*LD   MAG1:                    Load the Boot Puncher
*ST   ,MAG1:,MAG2:             Output Boot Loader to tape
```

BUILDING OVERLAYED DISC SYSTEM USING MAGNETIC TAPE PACKAGE

The magnetic tape and cassette packages include two starter systems. Starter 1 has no disc support. Starter 3 is non-overlayed disc-supporting system; non-overlayed so that it can be built to and loaded from a non-disc device, and disc-supporting so that it can be used to build an overlayed disc-supported system directly onto the disc using CAL/16D and TET/16.

For details of the system generation process, refer to the *System Planning and Configuration Guide.*

DISC SYSTEM BACKUP

Since each program or library resides in a separate file (see disc table of contents for file identification), the OS/16 MT2 disc package is well suited to be a working volume. A backup copy of the disc contents is easily created with the BACKUP Utility. In addition to the original shipped OS/16 MT2 disc package, any important disc volumes should be periodically backed up in the following fashion:

```
*MARK DSC1:,ON,PROT                          Mark volume on write protected
*LD MT16:BACKUP                              Load BACKUP
*OP ET                                       Make it an E-Task
*ST,IN=DSC1:,OUT=MAG1:,LIST=PR:,VERIFY       Save the disc on Magnetic Tape
```

Alternatively, the back up can be made directly to a second disc if available.

```
*MARK DSC1:,ON,PROT                          Mark volume on write protected
*LD MT16:BACKUP                              Load BACKUP
*OP ET                                       Make it an E-task
*INIT DSC2:,PACK,RE                          Initialize second disc pack
*MA DSC2:,ON                                 Mark it on-line
*ST,IN=DSC1:,OUT=DSC2:,LIST=PR:,VERIFY       Save the disc
```

The Starter 3 system on the disc is a non-overlayed disc-supported system in object code format. It is advisable to copy this system out to a magnetic tape or cassette, followed by BACKUP and Disc Integrity Check, to be held as a "backup OS" in case the disc is destroyed. This is particularly important for users with only one disc drive.

To create the tape, mount it on MAG1: and enter:

```
*LD  MT16:LIBLDR.OBJ                         Load Library Loader
*AS  1,MT16:OS16MT3.OBJ
*AS  2,MAG1:
*AS  4,MT16:BACKUP.OBJ
*AS  5,CON
*AS  6,MT16:DISCHECK.OBJ
*ST
CO   102                                     Output Starter 3
WF     2
CO   402                                     Output Disc Compress
WF     2
CO   602                                     Output Disc Check
WF     2
WF     2
END
```

To restore a disc pack from the dumped volume, load the saved STARTER 3 system from the backup tape, position to the start of disc compress, and enter:

```
*LD   MAG1:                                  Load BACKUP
*OP   ET
*INIT DSC1:,PACK,RE                          Initialize disc pack
*MA   DSC1:,ON                               Mark it On-line
*ST   ,IN=MAG1:,OUT=DSC1:, LIST=PR:,VERIFY   Restore the disc
```

DISC SYSTEM MAINTENANCE

This section illustrates the procedures involved in building and maintaining a disc-based program library under OS/16 MT2 by using the standard program library supplied with OS/16 as an example.

Building a Library

To build a library of programs on disc, one must have a Magnetic Tape, or Cassette containing all programs one wishes to build on the disc. The disc must be formatted and initialized (if not done so already), after formatting, enter:

```
*INIT DSC1:,PACK,READ          Initialize, volume=PACK, read check is performed
```

Once initialized, the disc must be marked On-line and the default volume set:

```
*MARK DSC1:, ON                Mark the disc On-line
*VOL PACK                      Set the Default Volume
```

Now the necessary files must be allocated:

```
*ALLO STARTER1.OBJ,IN
*ALLO STARTER3.OBJ,IN
*ALLO LIBLDR.OBJ,IN
*ALLO DLIB16.OBJ,IN
*ALLO CUP.OBJ,IN
*ALLO TET.OBJ,IN
*ALLO DISCHECK.OBJ,IN
*ALLO BACKUP.OBJ,IN
*ALLO SOURCEUP.OBJ,IN
*ALLO CAL16.OBJ,IN
*ALLO CAL16D.OBJ,IN
*ALLO EDIT.OBJ,IN
*ALLO COPY.OBJ,IN
*ALLO AIDS.OBJ,IN
*ALLO BOOTPNCH.OBJ,IN
*ALLO BOOTLOAD.OBJ,IN
*ALLO MIORTL.OBJ,IN
*ALLO LETTER.IN,CO,6
```

Copy the object programs using the OS/16 Library Loader:

```
*RW MAG1:                  Position tape to beginning
*FF  MAG1:                 Position to Starter1
*FF  MAG1:                 Position to Starter 3
*FF  MAG1:                 Position to Library Loader
*LD MAG1:                  Load the Loader
*TA.BG                     Set the task
*AS  1,MAG1:               Assign the input device
*AS  2,STARTER1.OBJ
*AS  3,STARTER3.OBJ
*AS  5,CON:                Assign the console device
*RW MAG1:                  Position tape back to beginning
*FF  MAG1:                 Position to Starter 1
*ST                        Start the loader
*.BG:>CO     102           Copy Starter 1
*.BG:>PA                   Pause the Loader
*FF  MAG1:                 Position to Starter 3
*CO
*.BG:>CO 103               Copy Starter 3
*.BG:>PA                   Pause the loader
*FF  MAG1:                 Position to Library Loader
*CL  2,3
*AS  2,LIBLDR.OBJ          Assign Logical Units
*AS  3,DLIB16.OBJ          to all of the files
*AS  4,CUP.OBJ
*AS  6,TET.OBJ
*AS  7,DISCHECK.OBJ
*AS  8,BACKUP.OBJ
*AS  9,SOURCEUP.OBJ
*CO                        Continue the loader
*.BG:>CO 102               Copy Loader
*.BG:>DU 103 CUP16         Dupe Driver Library
*.BG:>CO 104               Copy Cup
*.BG:>CO 106               Copy TET
*.BG:>CO 107               Copy Disc Integrity Check
*.BG:>CO 108               Copy BACKUP
*.BG:>CO 109               Copy Source Updater
*.BG:>PA                   Pause the loader
*CL  2,3,4,6,7,8,9
*AS  2,EDIT.OBJ            Assign the rest of
*AS  3,COPY.OBJ            the files
*AS  4,AIDS.OBJ
*AS  6,CAL16.OBJ
*AS  7,CAL16D.OBJ
*AS  8,BOOTPNCH.OBJ
*AS  9,BOOTLOAD.OBJ
```

```
*CO                                  Continue the loader
*.BG:>CO 102                         Copy OS Edit
*.BG:>CO 103                         Copy OS Copy
*.BG:>CO 104                         Copy OS Aids
*.BG:>CO 106                         Copy CAL/16
*.BG:>CO 107                         Copy CAL/16D
*.BG:>CO 108                         Copy Boot Puncher
*.BG:>CO 109                         Copy Boot Loader
*.BG:>PA
*CL 8,9
*AS 8,SPOOLER.OBJ
*AS 9,MIORTL.OBJ
*CO
*.BG:>CO 108                         Copy Output Spooler
*.BG:>DU 109      RELLDR             Copy Mini I/O RTL
*.BG:>END                            Terminate the loader
*CLOSE ALL                           Close all the files
```

Now copy the output spooler table using OS copy:

```
*LD COPY                             Load OS Copy Utility
*OPT COMP                            Set Option Compatible
*FF MAG1:                            Position to start of table
*AS 1,MAG1:                          Assign input tape
*AS 2,LETTER.IN                      Assign output file
*AS 5,CON:                           Assign command input device
*ST                                  Start OS Copy
*.BG>CPYB,256,,1,6                   Copy 6 256-byte records
*.BG>END                             Terminate OS Copy
*OPT NOCOMP                          Reset compatibility option
*CLOSE ALL                           Close all logical units
```

Now copy the source modules using OS source updater:

```
*LD SOURCEUP                         Load the source updater from disc
*FF MAG1:;FF MAG1:                   Position to the start of the source
*AS 3,CON:                           Assign list
*AS 5,CON:                           and command devices to the console
*ST                                  Start the updater
*.BG:>CO MAG1:,PCB16.CAL             Copy the PCB
*.BG:>CO MAG1:,EXEC16.CAL            Copy the EXEC
*.BG:>CO MAG1:,FMGR16.CAL            Copy the FMGR
*.BG:>CO MAG1:,CMDP16.CAL            Copy the CMDP
*.BG:>CO MAG1:,DLIB16.CAL            Copy the Driver Library
*.BG:>CO MAG1:,HLOC.CSS              Copy the High Level Operator Command Package
*.BG:>END                            End the updater
```

Now the disc is complete. To list the files on the disc to the Line Printer, enter:

```
*FI,PR:
```

Updating a Library

Object Programs

If an updated version of a program has been received, the disc may be updated as follows:

```
*LD LIBLDR                           Load the Library Loader into the Background
*AS 1,PTRP:                          Assign the input device
*DEL fn.OBJ                          fn-filename of program to be updated
*AL fn.OBJ,IN                        Reallocate file
*AS2,fn.OBJ                          Assign the output file
*AS5,CON:                            Assign the console device
*ST                                  Start the loader
*.BG:>CO 102                         Copy the updated program to the file
*.BG:>END                            End the loader
*CLOSE ALL                           Close the LU's
```

The new object program is now on the disc.

If an updated version of a source program has been received, the disc may be updated as follows:

```
*LD SOURCEUP                Load the source updater into the background
*DEL fn.CAL                 Delete the old file
*AS 3,CON:                  Assign the list
*AS 5,CON:                  and command devices to the console
*ST                         Start the updater
*.BG:>COPY MAG1:,fn.CAL     Copy the new source program
*.BG:>END                   End the updater
```

The new source program is now on the disc.

Once the disc has been set up with all the proper files, it is a good idea to Write protect all the files. To do this enter:

```
REP fd,FF00                 where fd=filename.ext
example:
REP LIBLDR.OBJ,FF00         Reprotect FF=write key,00=read key
```

Now the file may only be assigned with Read only access (SRO or ERO):

```
AS 1,LIBLDR.OBJ,SRO         Assign the loader to LU1 for Shared Read Only
```

This command will be accepted but:

```
AS 1,LIBLDR.OBJ
```

will be rejected with ASGN-ERR PROT.

To Write enable the file again enter:

```
REP LIBLDR.OBJ,0            Reprotect with ZERO keys
```

CHAPTER 12
FILES AND THE OPERATOR

All Direct-Access Devices supported by OS/16 MT2 may be accessed through the OS/16 MT2 File Manager, which provides a substantial and powerful set of volume and file management services. While a Direct-Access Device is marked Off-line, it is referred to by the device mnemonic associated with the device at system generation time. Example – DSC1:

Data on a Direct-Access Device is maintained as files on a named logical volume. Each volume contains all the information necessary to process the data on that volume. When a Direct-Access Device is marked On-line; the name of the volume mounted on that device is associated with, and used to refer to the device. Example – PACK:

MARK COMMAND

In order to ensure that the system maintains the correct volume information, the MARK command must be used to mount and dismount disc packs. When a new pack is mounted, the MARK ON command reads the volume name into the system volume table. Before dismounting a disc pack, it should be MARKed OFF to insure that no files on that volume are assigned. Use a DISPLAY LU command and the CLOSE command to close all LU's assigned to files for each task on that volume prior to issuing the MARK OFF.

OS OVERLAY FILE – OVERLAYED SYSTEMS

When the OS is boot loaded from disc, the Boot Loader saves the name of the file used to load the system. This file contains not only the resident portion of the OS, but also the OS overlays. When the OS is first loaded, the disc containing the OS file is marked On-line as the OS volume with write protection, as if the operator had entered:

```
*MARK DSC1:,ON,OS,PROTECT
```

When the OS volume is marked OFF, OS must be specified as an operand to the MARK command also:

```
*MARK DSC1:,OFF,OS
```

When the OS volume is marked OFF, operator commands are no longer accepted. The OS or a different OS must be boot loaded from disc.

The current OS fd may be displayed by entering the VOLUME command.

DISC INITIALIZATION

In order to prepare a disc pack for use by OS/16 MT2, the pack must be formatted by the Common Disc Formatter program.

See either the Common Disc Formatter Program Description, Part Number 06-173M95A15 for 2.5, 10, and 40 MB discs or the Common MSM Disc Formatter Program Description, Part Number 06-201M95A15 for 67 and 256 MB discs.

Once the disc pack has been formatted, it must be initialized by the OS/16 INITIALIZE command. In order to initialize a disc volume, the device it is mounted on must be Off-line. To initialize a new pack for OS/16 MT2 use enter:

```
*INIT DSC1:,PACK,READCHECK
```

This command places the name PACK in the Volume Descriptor of the pack on device DSC1:, constructs an empty directory and an allocation bit map which indicates all sectors unallocated except for: sector 0 (Volume Descriptor), the sectors required to contain the bit map, one cylinder for the empty directory (3 tracks on a 40MB Disc, 1 track on a 67MB, 256MB, Floppy Disc), and any sectors marked as defective during formatting. Since the READCHECK option performs a read check operation on each sector, this operation takes approximately three minutes on a 2.5MB disc and correspondingly long on any other disc.

If no defective sectors were detected in the formatting process, the initialization process may be shortened by not specifying the READCHECK option. The command would then be:

```
*INIT DSC1:,PACK
```

To rename a disc volume, mark the device On-line and enter:

```
*REN PACK:,NEWP:
```

To save an image of the currently loaded OS suitable for boot loading, mark the desired device On-line and enter:

```
*SAVE PACK:OS16MT2.000
```

NOTE

This command functions properly in a non-overlayed system only.

ASSIGNMENT AND ALLOCATION

OS/16 MT2 maintains control of assigned files through File Control Blocks (FCBs). FCBs are built in system space from the top of physical memory down. Any time an ASSIGN command is rejected with an ASGN-ERR BUFF message the amount of system space remaining is insufficient to build the required FCB. To gain space to allow the command, other assigned files must be closed or SET PARTITION command must be entered to increase the size of the .SYS partition:

```
*SET PART .SYS/D800          D800 = new start of system space
```

Keys and Access Privileges

The following examples illustrate the use of protection keys and access privileges for Direct-Access Files.

```
*ALLO PACK:FILE1,CO,100,AABB     Allocate a contiguous file with protection keys
*ALLO PACK:FILE2,IN,80           and an unprotected indexed file
*AS 1,PACK:FILE1,ERO,AABB        Assign file 1 for Exclusive Read
*AS2,PACK:FILE1,EWO,AA00         and for Exclusive Write
*AS3,PACK:FILE1,,AABB            Attempt to assign file for SRW (default) – rejected since it is already
                                 assigned exclusively.

ASGN-ERR PRIV

*CL 2                            Deassign for Exclusive Write
*AS 2,PACK:FILE1,SWO             Attempt to assign for Shared Write
ASGN-ERR PROT                    Rejected – invalid keys
*AS 2,PACK:FILE2                 File 2 assigned for SRW
```

Write Protected Disc

Write protection for discs is provided on a software basis. This allows the data to be read from the disc volume but prevents any attempt to write on the disc. Write protection should not be presumed to take the place of backup procedures since hardware failure can destroy data on a disc volume. To Write protect the data on a disc volume, enter the following command after mounting the pack:

```
*MARK DSC1:,ON,PROTECT
```

If the OS file is contained on a volume to be Write protected enter:

```
*MARK DSC1:,ON,OS,PROTECT
```

This command may be entered regardless of the state of the hardware write protect feature of the disc. If the disc device is left in hardware write protect mode, any attempts at MARK ON without PROTECT will instead MARK ON with PROTECT. This action ensures that the OS will never attempt to write on a hardware protected volume. Once a disc volume has been Write protected, all assignments for access privileges other than Shared Read Only (SRO) and all rename, reprotect, deletion and allocation attempts are rejected. Assignments for Shared Read Write (SRW) are granted Shared Read Only access.

DISC INTEGRITY CHECKING

The Disc Integrity Check Utility 03-080 is designed to restore a direct access volume to a usable state after a system crash or software failure. The most essential functions performed are:

- discs that were on-line at the time of the crash are marked off.
- the files that were assigned at the time of the crash are closed.
- the link fields of Indexed files that were being processed at the time of the crash are restored so that the data retained is usable by the system.

The first function is essential because unless the discs are marked off, they cannot be marked on without protect.

To restore a disc volume mounted on DSC1:, load the Disc Integrity Check and enter:

```
*TA name          . Set Current task to proper partition
*OP ET            Make it an E-Task
*ST, DSC1:,PR:    Start program-disc device=DSC1:, error list=PR:
```

Refer to Chapter 7 of this manual for a more detailed description of Disc Integrity Check.

CHAPTER 13
GUIDE TO WRITING AND USING CSS FILES

INTRODUCTION

OS/16's Command Substitution System (CSS) is a versatile facility that can be used for anything from simple batch-stream control to the development of highly complex macro commands. It has many of the characteristics of a programming language, and, as such, requires more than the usual degree of study to learn thoroughly. However, its structure allows simple functions to be performed without knowing everything there is to know about CSS. This chapter is intended to guide the user from the beginning stages through the more advanced stages of CSS use.

It is assumed that the user is familiar with the functions of the OS/16 non-CSS operator commands, or at least with their most common forms. If not, these may be found in Chapter 3 of this manual.

BASIC QUESTIONS

What is a CSS File?

A CSS file is a series of OS/16 MT2 operator commands. These commands exist in some machine-readable form, such as on cards, or disc, or Magnetic Tape. When the file is called, the commands contained in it are executed in order. When the special command $EXIT is executed, CSS execution is terminated from this file.

How is a CSS File Used?

The file is installed in a position to be read by the machine; i.e., the card deck is placed into the Card Reader hopper, the Magnetic Tape is mounted, or the necessary steps are performed for the particular medium the CSS file is on. If the CSS file is on disc, no installation is necessary. The device being used, whether disc, Card Reader, or other device, must be On-line.

The operator calls the CSS file by typing in the name of the device or disc file. This is typed in the same way as the operator would type in a command. For example, if the file is on cards, and the Card Reader is named CR2, the operator types in:

```
CR2:
```

and the file is executed. The colon (:) tells the system that the CSS file is on a non-disc device.

Can One CSS File Call Another?

With OS/16, a CSS file is like the console operator. Therefore, one CSS file can call another, just as the operator would. This may only continue to a certain depth. When too many CSS files are active, the Command Processor's Buffers fill up, and at this time, a new CSS file may not be called. The maximum nesting depth is set up at SYSGEN (SYStem GENeration) time, and can vary from one system to another. If a CSS file tries to call another one, and exceeds this depth, the system logs:

```
CSS-ERR LVL
```

and CSS processing ceases.

When one CSS file calls another, the first file is still active. When the second file finishes processing, the first one continues from the previous location. Therefore, a CSS file does not directly branch to another, but calls it like a subroutine.

What Commands Can Be Executed From CSS?

Every operator command can be executed from a CSS file. In addition, a number of special CSS commands can be executed, such as $EXIT, mentioned previously. Special CSS commands start with a dollar sign, so it is easy to identify them.

USING CSS FOR BATCH CONTROL

The simplest use of CSS is to control a batch job. This allows the user to put together the sequence of operator commands needed to run a job, together with the data for the job, and to enter the entire job stream to OS/16 through CSS. This saves much work for the console operator, and the job control statements can often be reused.

Job Control Decks

A typical job control deck is illustrated in Figure 13-1.

```
TASK .BG
LDBG CAL16                      ;*DO ASSEMBLY
ASSIGN 1,CR:
ASSIGN 2,MAG:
ASSIGN 3,PR:
ASSIGN 4,SCRATCH
START   ,SCRAT                  ;*START ASSEMBLER
. . .
(data)
. . .
REWIND MAG:
LDBG LIBLDR                     ;*LOAD LIBRARY LOADER
ASSIGN 5,CR:                    ;*ASSIGN COMMAND INPUT TO THIS FILE
ASSIGN 6,PROGRAM.OBJ
ASSIGN 7,SUBROUT.OBJ
START                           ;*START LOADER
OUT 6
LOAD 2
BIAS 6000
LINK 7
MAP 3
XOUT
END
CLOSE ALL                       ;*LOAD MODULE IS NOW BUILT
LBDG PROGRAM.OBJ                ;*LOAD NEW LOAD MODULE
START                           ;*RUN PROGRAM
$EXIT                           ;*END OF JOB
```

Figure 13-1. Typical CSS Job Control Deck

The first statements load the CAL/16 assembler, assign devices and files to its Logical Units, and start the assembly process; following these statements are the source statements of the program to be assembled. There are more command statements to load and assign devices to the Library Loader; the START command starts the Loader. The statements from OUT 6 to END are part of the Library Loader's command language. The CSS file loads the load module that the Library Loader produced, and runs the program. Finally, the $EXIT command terminates CSS processing.

Device-Independent Job Control Decks

The example given previously assumes that the CSS file is in a device named CR:. It would not work properly if the device name were different, for example, MAG2:. This is because the CSS statements specifically ASSIGN Logical Unit 1 of CAL and Logical Unit 5 of the Library Loader to a device named CR:, and these Logical Units (source input for CAL, command input for the loader) refer to the job control deck itself.

The program source and the Library Loader commands (OUT 6 through END) are embedded in the job control deck. If this job control deck were entered from device MAG2:, all CSS commands would be processed properly, up to the first source statement of the program being assembled. CAL reads the source, however, from device CR:, since this is the assignment of Logical Unit 1.

The situation can be a problem if there are two or more identical devices of the same kind in the system. Assume there are two card readers, named CR: and CR2:. The example job control deck works in one but not the other.

To create a device-independent job control deck, use the characters @0 whenever referring to the CSS file itself. @0 means "this device or file." The example is changed as follows:

```
LDBG CAL16
ASSIGN 1,@0
ASSIGN 2, MAG:
. . . .
ASSIGN 5, @0
. . . .
```

Separation of Jobs

When any error occurs in processing a CSS file, CSS processing stops, and control is returned to the console operator. This is desirable in many circumstances. It is not desirable when batch jobs are being run, however. If there are several sets of job control statements in the CSS file, for example, in the Card Reader, the faulty termination of one job should not cause all jobs to be aborted.

The CSS commands $JOB and $TERMJOB are used to isolate errors to a single job. Each job control deck starts with a $JOB statement and ends with a $TERMJOB statement. If this is done, an error in a given job simply causes the CSS Processor to skip all commands until a $TERMJOB or $JOB is found, and to resume normal CSS processing.

A typical batch stream, consisting of several jobs, is illustrated in Figure 13-2.

```
$JOB
  .
  .                  1st JOB CONTROL SEQUENCE
  .
$TERMJOB
$JOB
  .
  .                  2nd JOB CONTROL SEQUENCE
  .
$TERMJOB
$EXIT                *END OF BATCH STREAM
```

NOTE

It is not permissible to nest Jobs.

Figure 13-2. Typical Batched CSS Stream

Program Pauses and Other Interactions

CSS processing should not continue while a program called by CSS is running. In any job control deck, the job steps are sequential. When a background program is started, it runs to completion before any more CSS statements are executed. Otherwise, the CSS Processor might try to perform device assignments for the second job step before the first one was finished.

When a background program executes a PAUSE, CSS processing does not resume. Instead, the PAUSE message is routed to the console operator, who is responsible for correcting the error and issuing a CONTINUE command. CSS assumes that the program is still active, whether or not it has paused. If the program goes to End of Task, or if it is cancelled by the operator, CSS processing resumes.

This procedure assumes that programs only PAUSE for one reason:

> An error, or some abnormal condition, has occurred, and the program is unable to take corrective action by itself. Operator intervention is required.

Programs that PAUSE arbitrarily, or CAL assemblies with PAUSE or PPAUS statements in their source decks, cannot be properly handled under CSS batch control.

USING CSS TO AVOID REPETITIOUS ACTIONS

CSS is not used only for batch control. It can also be used to avoid lengthy, repetitious operator type-ins. Assume that at a given installation all CAL assemblies are done with a standard set of Logical Unit assignments. These assignments are made into a CSS file, and that CSS file is put on disc and used before each CAL assembly. For example, assume a disc file named CALASINE contained the following:

```
TASK        .BG
CLOSE       ALL
ASSIGN      1,CR:
ASSIGN      2,MAG1:
ASSIGN      3,PR:
ASSIGN      4,CALSCRAT.TMP
ASSIGN      5,CROSSREF.TMP
ASSIGN      6,SYMDUMP.TMP
ASSIGN      7,SORCELIB.CAL
ASSIGN      8,SQUEEZE.TMP
ASSIGN      9,ERRLIST.TMP
$EXIT
```

The operator places the source in device CR: and an object output tape on device MAG1: and types:

```
LDBG CAL
CALASINE
START
```

and the assembly proceeds. The LDBG and START commands can also be put on the CSS file.

This procedure not only simplifies the operator's task, but also reduces the possibility of error. For this reason, CSS is always used, if possible, to enter patch information to OS/16 after system start-up. Assume the installation has a list of patches that are necessary to fix a bug in the interim before a new revision of OS/16, or a list of patches to a user-written driver developed at the installation. These patches are prepared on a card deck, with verification information, as follows:

```
BIAS 0
MODIFY 1FE0,220,80A,4300,1ED4
EXAMINE 1FE0,4
$EXIT
```

Immediately upon system start-up, this deck is put in the Card Reader, and the command CR: entered from the console. The CSS file is read, and the patches are made to the system without possibility of typing errors. The EXAMINE command produces a listing of the patches on the console device. The operator verifies this listing before proceeding.

USING CSS TO BUILD COMPLEX COMMANDS

Although the previously described CSS uses are very important, the most sophisticated and powerful use of CSS is as a system macro command language, to build complex commands. The set of commands provided in the OS/16 Command Processor is sufficient to perform any function, but at a heavy cost in type ins. Complex commands like "COMPILE/ ASSEMBLE/LOAD and GO" are not provided. This is because such commands can be built using CSS.

The High Level Operator Command Package, supplied by INTERDATA, contains some useful and powerful complex commands built with CSS. These commands, however, probably will not satisfy everyone's needs. The following sections describe how to create complex commands tailored to the specific needs of any installation or application.

Passing Arguments to CSS Files

The CAL assignment example given above does not work if some of the devices vary from assembly to assembly. When executed, file CALASINE always assigns Logical Unit 1 to CR:, 2 to MAG1:, etc. To build a proper ASSEMBLE command, it is necessary to pass variable data (command "arguments") to a CSS file. This is done by using the character sequence @n.

Whenever the sequence @n (where n is any decimal number) is applied in a CSS command, the CSS processor uses the n'th argument in the CSS calling statement. This is best shown by example. Assume CSS file ASINE123 contains the following commands:

```
ASSIGN 1,@1
ASSIGN 2,@2
ASSIGN 3,@3
$EXIT
```

and the operator types in:

```
ASINE123 CR:,PR:,MAG1:
```

The CSS file is executed as though it contained:

```
ASSIGN 1,CR:
ASSIGN 2,PR:
ASSIGN 3,MAG1:
$EXIT
```

The first argument of the calling statement is CR:, and every occurrence of the sequence @1 in the CSS file is replaced with CR:. An argument is always terminated with a comma or end-of-line. Therefore, the second argument is PR: and the third argument is MAG1:. Now the previous example, CALASINE, can be turned into a generalized ASSEMBLE command as illustrated in Figure 13-3.

```
TASK        .BG
LDBG        CAL
CLOSE       ALL
ASSIGN      1,@1                ;*VARIABLE INPUT
ASSIGN      2,@2                ;*VARIABLE OUTPUT
ASSIGN      3,@3                ;*VARIABLE LISTING
ASSIGN      4,CALSCRAT.TMP
ASSIGN      5,CROSSREF.TMP
ASSIGN      6,SYMDUMP.TMP
ASSIGN      7,@4                ;*VARIABLE SOURCE LIBRARY
ASSIGN      8,SQUEEZE.TMP
ASSIGN      9,ERRLIST.TMP
START
$EXIT
```

Figure 13-3. ASSEMBLE Command Example

This file may be called as follows:

```
ASSEMBLE CR:,MAG4:,PR:,LIBRARY.CAL
```

The first argument is the input device, the second is the output device, the third is the listing device, and the fourth argument is the source library device.

Testing Arguments for Existence

Assume the operator wants to assemble a program without using a source library device? If the following is typed in:

```
ASSEMBLE CR:,MAG4:,PR:
```

the fourth argument is not present. A missing argument is considered to be a "null string", that is, a sequence of no characters. Therefore, the command ASSIGN 7,@4 is executed as:

```
ASSIGN 7,
```

This is an illegal command. The CSS file aborts, and the assembly does not take place.

It is possible, however, to test whether an argument exists or not. This is done with one of the special CSS commands $IFNULL and $IFNNULL. The $IFNULL command tests to see whether its argument is null. The $IFNNULL command tests to see whether its argument is not null. If the tested condition is true, the CSS Processor continues to execute statements. If the tested condition is not true, the CSS processor skips all statements until a $ENDC (end conditional) or a $ELSE command is found. Therefore, the command ASSIGN 7,@4 can be replaced with the following:

```
$IFNNULL @4                 ;*IS THERE A SOURCE LIBRARY?
ASSIGN 7,@4                 ;*IF SO, ASSIGN IT TO L.U. 7
$ENDC                       ;*END OF CONDITIONAL
```

This facility can also be used to set up default assignments. For example, assume the list device is normally the printer (PR:). The following sequence can be put into the ASSEMBLE CSS file:

```
$IFNULL @3                  ;*IS THERE A LIST DEVICE SPECIFIED?
ASSIGN 3,PR:                ;*IF NOT, USE PR:
$ELSE                       ;*IF THERE IS,
ASSIGN 3,@3                 ;*ASSIGN IT TO L.U. 3
$ENDC
```

If the operator typed in:

```
ASSEMBLE MAG2:,MAG3:,,MAG4:
```

The listing is assigned to PR:. An argument is null if it consists only of blanks; therefore, the ASSIGN 3,PR: statement is executed.

Testing Files for Existence

It is often desirable to allocate files from a CSS file. The development of the ASSEMBLE file requires a number of temporary files to assign for scratch purposes, etc., which causes a dilemma.

- If the scratch file is allocated, and it already exists, the ALLOCATE command is rejected and the CSS file aborts.
- If the scratch file is not allocated, and it does not exist, the ASSIGN command is in error, and the CSS file aborts.

In order to solve this problem a facility is provided to enable the CSS file to test the existence of certain files. This is done with the $IFX (if file exists) and $IFNX (if file does not exist) commands. These special commands work in the same way as the $IFNULL and $IFNNULL commands mentioned previously. That is, if the tested condition is true, CSS continues processing, but if the tested condition is false, CSS skips until a $ENDC or $ELSE command is found.

For example:

```
$IFNX CALSCRAT.TMP              ;*DOES CALSCRAT.TMP EXIST?
ALLOCATE CALSCRAT.TMP,IN,86     ;*NO,ALLOCATE IT
$ENDC                           ;*END OF CONDITIONAL
```

This sequence checks to see if the scratch file already exists, and allocates it if it does not.

The same facility can be used to test for the existence of a file passed to the CSS file as an argument. For example, see Figure 13-4.

```
TASK .BG
LDBG EDIT                       ;*LOAD THE EDITOR
$IFNX @1                        ;*IF THE FILE DOESN'T EXIST YET
ASSIGN 1,CON:                   ;*THEN INPUT FROM CONSOLE
$ELSE                           ;*BUT IF THE FILE DOES EXIST
ASSIGN 1,@1                     ;*THEN USE IT FOR INPUT
$ENDC
ALLOCATE EDIT.TMP,IN,80         ;*NOW ALLOCATE AN OUTPUT FILE
ASSIGN 2,EDIT.TMP,ERW           ;*ASSIGN FOR EXCLUSIVE READ-WRITE
ASSIGN 5,CON:                   ;*ASSIGN COMMAND INPUT
START                           ;*START THE EDITOR
CLOSE 1,2,5                     ;*CLOSE FILES
$IFX @1                         ;*IF THE CONSOLE WASN'T USED
RENAME @1,EDIT.OLD              ;*THEN RENAME THE INPUT FILE
$ENDC
RENAME EDIT.TMP,@1              ;*NOW RENAME THE OUTPUT FILE
$EXIT                           ;*AND RETURN TO THE CONSOLE
```

Figure 13-4. EDIT Command Example

This file needs some explanation. It is called as follows:

```
EDIT filename
```

If filename is the name of an existing file, that file is edited. The output file is named EDIT.TMP; but after the editing is done, the output file is given the name of the input file, and the input file is renamed EDIT.OLD. If filename does not yet exist, however, the input device is assumed to be the console, and the file created by the editor is given the name filename. This CSS file only works if @1 refers to a Direct-Access File, since the RENAME command as given above is not applicable to Non-Direct Access Devices. Now the EDIT file can be put on disc and used just as though it were a normal OS/16 command.

Return Codes and Error Handling

When a program, called by CSS, detects an error and terminates abnormally, CSS takes special action. A job control deck that performs a compilation, assembly, creation of a load module, loading and execution of a program, does not continue the process, if the first step (the compilation) terminated in error.

CSS must be informed of the erroneous end of task. This is done through the Return Code mechanism, but the program must cooperate. Each program, on terminating, returns a Return Code that may be used to show why the program terminated. This return code is a 16-Bit number that is set up by the program's author in the SVC 3 call that terminates the program. (For details on SVC 3, see the *OS/16 MT2 Programmer's Reference Manual).*

A return code of 0 means the program terminated properly, with no errors. A non-zero return code means the program terminated abnormally. The return code of 255 is used by the system if the task was cancelled by the console operator.

CSS files test the return code with a set of special CSS commands. These are:

$IFE (equal)	$IFNE (not equal)
$IFG (greater)	$IFNG (not greater)
$IFL (less)	$IFNL (not less)

These commands are used as follows:

$IFE 14	If Return = 14
$IFG 0	If Return Code > 0
$IFL 255	If Return Code < 255
$IFNE 1	If Return Code ≠ 1
$IFNG @3	If Return Code ≤ the third argument
$IFNL 2840	If Return Code ≥ 2840

If the tested condition is true, CSS continues to execute commands; if the condition is false, however, CSS skips all statements until a $ENDC or $ELSE is found.

The special command $SKIP causes CSS to unconditionally skip to the next $TERMJOB command, if inside a job control deck. This has the same effect as any error detected in a CSS statement. Thus:

```
LDBG CAL                    ;*LOAD THE ASSEMBLER
START                       ;*START IT
$IFNE 0                     ;*ANY ERRORS?
$SKIP                       ;*IF YES, SKIP FURTHER PROCESSING
$ENDC                       ;*OTHERWISE, CONTINUE
```

This is the simplest way to handle errors. If the program has multiple error return codes, each with a distinct meaning, the CSS file may inform the operator as to the nature of the problem, as follows:

```
LDBG PROGRAM                ;*LOAD THE PROGRAM
START                       ;*START IT
$IFNE 0                     ;*IF ANY ERRORS
$IFNE 1                     ;*RC = 2 MEANS FATAL ERROR
$SKIP                       ;*SO SKIP
$ENDC                       ;*BUT RC = 1 MEANS NON-FATAL ERROR
$COPY
*NON-FATAL ERROR DETECTED
$NOCOPY
$ENDC                       ;*CONTINUE AFTER LOGGING MESSAGE
```

Note the nesting of the $IFNE. . . .$ENDC sequences. Nesting of any $IF tests is permitted, to a level of 255.

A CSS file not only tests the Return Code, it may set it as well. This is possible because CSS is only active when the controlled program is inactive, and therefore the Return Code is not used asynchronously. The Return Code is set using the commands:

```
SET CODE n
```

which has the effect of setting the Return Code to n. This may be any number from 0 to 9999. The Return Code is changed whenever a program is run.

This ability of CSS to set the Return Code may be used by one part of a CSS file to signal another, providing there are no START commands between setting and testing the Return Code.

Sending Error Messages to the Console

The special CSS commands $COPY and $NOCOPY allow the CSS file to send messages to the system log device. Normally, CSS should be in $NOCOPY mode. Under these conditions, CSS commands are executed but are not printed out at the console or on the log device. If the CSS command $COPY is executed, however, a copy of all CSS statements executed up to and including the next $NOCOPY is printed at the console or on the log device, or both. (The choice of routing is set up by the system console operator with the SET LOG command, and CSS should not attempt to overide this choice.)

Since comments, as well as commands, are printed out, the easiest way to send a message is:

```
$COPY
*THIS IS A MESSAGE FOR THE OPERATOR
$NOCOPY
```

CREATING CSS FILES ON DISC

CSS files can be created on disc in one of two ways: first, the OS EDIT or Source Updater utility programs can be used in their usual fashion, and second, the BUILD and ENDB commands can be used to create a CSS file directly from the system console.

The BUILD command is used as follows:

```
BUILD filename.ext
```

This command causes an Indexed file of the name filename.ext to be allocated if there is not one already on the disc, and then puts the console into a "data entry" mode, to write data to that file. Every line typed at the console from that time until an ENDB command is found is written to the file being built.

This data may include any text at all, including commands, @ sequences, etc. The Command Processor does not attempt to execute any commands contained in the text, nor does it attempt to expand @ sequences. The only thing that cannot be written to a file being built is a line with the characters ENDB in the first four character positions.

When an ENDB command is read, the console reverts to its normal "command" mode.

To build a CSS file named FRED.CSS, the following is entered:

```
BUILD FRED.CSS
*THIS IS A CSS FILE NAMED @0
$EXIT
ENDB
```

This CSS file is now ready for execution. It may be called as follows:

```
FRED.CSS
```

or simply:

```
FRED
```

since the CSS Processor uses the default extension .CSS if the console operator does not type in any specific extension.

ADVANCED CONSIDERATIONS

Aborting All CSS Files

The $EXIT command causes the current CSS file to return to its caller. The command $CLEAR causes all currently active CSS files to be abruptly terminated, and returns control directly to the console operator. This command is used when a massive error is detected, and continuation is inappropriate.

Building Task Control Files

Although it is possible to embed task control commands within a CSS file, as shown previously, certain problems may occur while using this technique. In the example:

```
LDBG LIBLDR                ;*LOAD THE LIBRARY LOADER
TASK .BG
ASSIGN 5,@0                ;*ASSIGN TASK CONTROL TO THIS FILE
START                      ;*START LOADER
OUT 6
LOAD 1
LINK 2
EDIT 2
XOUT
MAP 3
END
$EXIT
```

the statements between START and $EXIT are not CSS commands, but are commands to the Library Loader. The START command activates the loader, which then reads and executes the commands from OUT 6 to END. Since these commands are not CSS commands and are not processed by CSS, but by the loader, parameter substitution via @n operands may not be used. This is because the loader does not contain parameter-substitution logic. There is no mechanism to apply parameter-substitution logic, even if it were desired.

This example does not work if the CSS file is on disc, since the ASSIGN 5, @0 command positions the file at its first logical record, and the loader gets LDBG LIBLDR as its first command.

If the loader is cancelled or abnormally terminates before it exhausts its full set of commands, the CSS Processor reads the next Library Loader command, as a CSS command, and attempts to execute it. In most cases, such a command would not be executable, and CSS processing would terminate; however, sometimes, utility program commands are similar to OS/16 commands, and the CSS Processor is able to execute one or more of them, often with undesired results.

In order to prevent these problems, the BUILD and ENDB commands, or the special CSS commands $BUILD and $ENDB, can be used. The following file has the same effect as the previous one, but provides better error control:

```
LDBG LIBLDR                ;*LOAD THE LOADER
TASK .BG
ALLOCATE LOADER.TMP,IN,10
BUILD LOADER.TMP           ;*BUILD LOADER CONTROL FILE
OUT 6
LOAD 1
LINK 2
EDIT 2
XOUT
MAP 3
END
ENDB                       ;*END OF LOADER CONTROL FILE
ASSIGN 5,LOADER.TMP,ERW    ;*ASSIGN CONTROL FILE
START                      ;*START THE LOADER
CLOSE 5                    ;*CLOSE CONTROL FILE
DELETE LOADER.TMP          ;*DELETE CONTROL FILE
$EXIT
```

This sequence builds a separate control file called LOADER.TMP and assigns it, rather than the CSS file itself, to the loader command input Logical Unit. The loader control file is then deleted from the disc before the CSS file terminates.

The $BUILD and $ENDB commands allow parameter substitution. Using BUILD. . .ENDB, all @ signs in the data being written to the file being built are ignored, as described previously. If $BUILD. . .$ENDB are used instead of BUILD. . .ENDB, parameter substitution takes place.

Using Standard File Extensions

In a disc-based system, standard file extensions can be used to relieve the console operator's task, and to allow CSS files to perform more sophisticated functions. The concatenation facility of CSS allows the use of standard extensions in a simple manner.

Standard file extensions that are useful in FORTRAN compilation, CAL assembly, are:

.FTN	FORTRAN source
.CAL	CAL source
.OBJ	object format
.LST	listing format
.ERR	error listing file
.LDM	load module format

A file COMPILE.CSS can be created to control compilation as illustrated in Figure 13-5.

```
*COMPILE CSS FILE
LDBG FORTRAN                ;*LOAD THE COMPILER
ASSIGN 1,@1.FTN,SRO         ;*ASSIGN FORTRAN INPUT
ASSIGN 2,@1.CAL,ERW         ;*ASSIGN CAL OUTPUT
ASSIGN 3,@1.LST,EWO         ;*ASSIGN LISTING (APPEND)
ASSIGN 7,@1.ERR             ;*ASSIGN ERROR MESSAGE FILE
START                       ;*START COMPILER
CLOSE ALL                   ;*CLOSE ALL FILES
$EXIT                       ;*AND EXIT
```

Figure 13-5. COMPILE Control Example

This file is called as follows:

```
COMPILE program-name
```

The COMPILE file assumes the FORTRAN source is on a file called program-name.FTN and puts the output, listing, and error messages onto files named program-name.CAL, program-name.LST and program-name.ERR respectively. This file assumes that all the named files already exist. The ASSEMBLE file can be created similarly (see Figure 13-6).

```
*ASSEMBLE CSS FILE
LDBG CAL16                  ;*LOAD ASSEMBLER
ASSIGN 1,@1.CAL,SRO         ;*ASSIGN CAL INPUT
ASSIGN 2,@1.OBJ             ;*ASSIGN OBJECT OUTPUT
ASSIGN 3,@1.LST,EWO         ;*ASSIGN LISTING (APPEND)
ALLOCATE @1.CRF,IN,256      ;*ALLOCATE CROSSREF SCRATCH
ASSIGN 5,@1.CRF             ;*ASSIGN SCROSS REFERENCE
ALLOCATE @1.SCR,IN,86
ASSIGN 4,@1.SCR
START ,SCRAT,CROSS
CLOSE ALL                   ;*CLOSE ALL LOGICAL UNITS
DELETE @1.CRF               ;*DELETE ALL SCRATCH FILES
DELETE @1.SCR
$EXIT                       ;*NOW EXIT
```

Figure 13-6. Assembly with Standard Extensions Example

This CSS file assigns all Logical Units and allocates and deletes scratch files as appropriate. It is called just like the COMPILE file, and assumes the files program-name.CAL, program-name.OBJ and program-name.LST already exist.

The library loader control file is illustrated in Figure 13-7.

```
*LIBLOAD CSS FILE
LDBG LIBLDR                 ;*LOAD LIBRARY LOADER
TASK .BG
ASSIGN 1,@1.OBJ,SRO         ;*ASSIGN OBJECT INPUT
ALLOCATE @1.LDM,IN          ;*ALLOCATE LOAD MODULE OUTPUT
ASSIGN 2,@1.LDM             ;*AND ASSIGN IT
ASSIGN 3,@1.LST,EWO         ;*ASSIGN LIST FILE (APPEND)
ASSIGN 4,FTNRTL.OBJ,SRO     ;*ASSIGN FORTRAN LIBRARY
ALLOCATE @1.LDC,IN,10
BUILD @1.LDC                ;*BUILD LOADER CONTROL FILE
OUT 2
LOAD 1
EDIT 4
MAP 3
XOUT
END
ENDB
ASSIGN 5,@1.LDC             ;*ASSIGN CONTROL FILE
START                       ;*START LOADER
CLOSE ALL                   ;*CLOSE ALL LOGICAL UNITS
DELETE @1.LDC               ;*DELETE LOADER CONTROL FILE
$EXIT                       ;*AND EXIT
```

Figure 13-7. Library Loader Example

This file creates a load module from the object file, by linking the FORTRAN Run-Time Library (RTL) subroutines.

Now a file can be built that coordinates the activities of all these files, as illustrated in Figure 13-8.

```
*EXECUTE CSS FILE
TASK .BG
ALLOCATE @1.CAL,IN,80          ;*ALLOCATE ALL NEW FILES
ALLOCATE @1.OBJ,IN
ALLOCATE @1.LDM,IN
ALLOCATE @1.LST,IN,120
ALLOCATE @1.ERR,IN,120
COMPILE @1                     ;*COMPILE THE PROGRAM
ASSEMBLE @1                    ;*ASSEMBLE IT
LIBLOAD @1                     ;*BUILD A LOAD MODULE
LDBG @1.LDM                    ;*LOAD THE LOAD MODULE
START                          ;*AND RUN THE PROGRAM
$EXIT
```

Figure 13-8. Compile, Assemble, Load and Execute Example

This EXECUTE file takes the program through all steps from compilation to execution. It also puts on the listing file the FORTRAN listing, assembly listing, and load module map. But it makes the error of assuming that the CAL, OBJ, LDM, LST and ERR files do not exist at the time EXECUTE is called. Furthermore, it does not take into account the possibility that the program may not be a FORTRAN program, or may already exist in CAL or OBJ form.

A FINAL EXAMPLE

Figure 13-9 contains a version of the EXECUTE file that takes the most basic version of the program that exists, and manipulates it as needed to execute it. If there is already a load module, it is simply executed. If there is an object file but no load module, a load module is created and then executed. If there is no load module or object, but there is a CAL file, it is assembled, and a load module is built and executed. Finally, if the most basic version is FORTRAN, the entire procedure described above is performed.

This example illustrates several important features including the use of indentation to make the file easy to read, and complete error-handling facilities, which include the processing of the FORTRAN return Code of 1, which means "one or more errors detected by compiler". (This return code is only returned by FORTRAN V Level 1 R01 and later.)

In every case there is a test for program cancellation, because any program running at any time may be cancelled by the console operator, and CSS files should take this into account.

The CSS EXECUTE file, and the files that is calls, COMPILE, ASSEMBLE and LIBLOAD, all allocate their own files and delete them when finished. This promotes good use of the disc. The use of the program name as part of the filenames, even for scratch files, tends to avoid filename conflicts.

With the use of this package, given a program in FORTRAN format on disc called, for example, FRED.FTN, the operator can type in:

```
EXECUTE FRED
```

and a long and complex sequence of events will take place, culminating in the execution of program FRED. No additional typeins by the operator are necessary. When the execution is finished, the CAL, object, load module, and listing files are available on disc, as well as the original FORTRAN file.

This example shows how a complex command with advanced facilities is created using CSS. Other than that, it does not fill the specific needs of any application. To promote clarity, no facilities for processing FORTRAN subprograms are included. The user is encouraged to use the techniques shown in the development of this CSS file, even though the file itself is not used. An important aspect of CSS is that the user does not have to rely only on the commands the Operating System designers have put into the Command Processor, but may design CSS commands to satisfy a particular set of needs.

```
*EXECUTE CSS FILE
TASK .BG
$IFNX @1.LDM                                    ;*IF NO LOAD MODULE
    $IFNX @1.OBJ                                ;*THEN IF NO OBJECT
        $IFNX @1.CAL                            ;*THEN IF NO CAL SOURCE
            $IFNX @1.FTN                        ;*THEN IF NO FORTRAN SOURCE
                $COPY                           ;*THEN ERROR
                *NO SUCH FILE IN FORTRAN FORMAT
                $NOCOPY
                $SKIP                           ;*SO EXIT
            $ENDC                               ;*ELSE (THERE IS A FORTRAN FILE)
            $IFX @1.LST                         ;*IF THERE IS A LISTING FILE
                DELETE @1.LST                   ;*THEN DELETE IT
            $ENDC
            ALLOCATE @1.LST,IN,120              ;*NOW ALLOCATE A LISTING FILE
            $IFX @1.ERR                         ;*IF THERE IS AN ERROR FILE
                DELETE @1.ERR                   ;*THEN DELETE IT
            $ENDC
            ALLOCATE @1.ERR,IN,120              ;*ALLOCATE ERROR FILE
            ALLOCATE @1.CAL,IN,80               ;*ALLOCATE CAL OUTPUT FILE
            COMPILE @1                          ;*NOW COMPILE
            $IFE 1                              ;*IF RETURN CODE IS 1
                $COPY                           ;*THE COMPILER FOUND ERRORS
                *ERROR IN COMPILATION
                $NOCOPY
                $SKIP                           ;*SO EXIT
            $ENDC                               ;*ELSE (NO COMPILE ERRORS)
            DELETE @1.ERR                       ;*DELETE THE ERROR FILE
            $IFNE 0                             ;*IF RETURN CODE ISN'T 0
                $COPY                           ;*FORTRAN WAS CANCELLED
                *COMPILATION ABORTED
                $NOCOPY
                DELETE @1.CAL                   ;*SO THE CAL FILE IS NO GOOD
                DELETE @1.LST                   ;*AND THE LIST FILE IS USELESS
                $SKIP                           ;*DELETE THEM AND EXIT
            $ENDC
        $ENDC                                   ;*NOW THERE IS CAL BUT NO OBJ
        ALLOCATE @1.OBJ,IN                      ;*SO ALLOCATE OBJECT FILE
        ASSEMBLE @1                             ;*AND DO ASSEMBLY
        $IFNE 0                                 ;*IF RETURN CODE ISN'T 0
            $COPY                               ;*CAL WAS CANCELLED
            *ASSEMBLY ABORTED
            $NOCOPY
            DELETE @1.OBJ                       ;*SO DELETE THE OBJECT FILE
            $SKIP                               ;*AND EXIT
        $ENDC
    $ENDC                                       ;*NOW THERE IS OBJ BUT NO LDM
    ALLOCATE @1.LDM,IN                          ;*SO ALLOCATE LOAD MODULE FILE
    LIBLOAD @1                                  ;*AND CREATE LOAD MODULE
    $IFNE 0                                     ;*IF RETURN CODE ISN'T 0
        $COPY                                   ;*THE LOADER WAS CANCELLED
        *LIBRARY LOAD ABORTED
        $NOCOPY
        DELETE @1.LDM                           ;*SO DELETE LOAD MODULE FILE
        $SKIP                                   ;*AND EXIT
    $ENDC
$ENDC                                           ;*AT LAST WE HAVE A LOAD MODULE
LDBG @1.LDM                                     ;*LOAD IT
START                                           ;*START IT
$EXIT                                           ;*FINISHED!
```

Figure 13-9. Complex CSS Example

CHAPTER 14
HIGH LEVEL OPERATOR COMMAND PACKAGE

INTRODUCTION

The High Level Operator Command Package allows the user to perform complex operator functions with a single operator command. For example, the command FORTCLG compiles, establishes, loads, and executes a FORTRAN program with no operator intervention.

The High Level Operator Command Package consists of a set of Command Substitution System (CSS) procedures. These procedures are designed to extend the basic commands included in the OS/16 MT2 Command Processor. Calls for the most common functions consist of simply a verb and a single operand. This chapter is designed as an operator's guide in using the package.

SYSTEM REQUIREMENTS

The following are the required resources for operation of the High Level Operator Command Package.

- OS/16 MT2
- A console device
- A disc device
- A printer
- The utility programs listed in the Operational Data section.

A disc is required to use this package. Disc files are assumed for most data, particularly for intermediate and output data. A default set of task establishment parameters is used in each procedure which produces an established task. This default set of parameters may be overridden by parameters specified by the user at the time the procedure is called.

Some additional assumptions are:

1. The background partition is available to the CSS procedure.
2. The background partition is large enough to load the appropriate utility program or language Processor.
3. All referenced utilities and language Processors are present in object form on the default system volume under the filenames specified in the Operational Data section.
4. The following types of files have the indicated extensions:

FORTRAN source	.FTN
CAL source	.CAL
CAL MACRO source	.MAC
Object	.OBJ
Task Image	.TSK
Task Logical Unit assignment files	.ASN
Task establishment command input	.TET
Help files	.HLP

5. The console device mnemonic is CON:.
6. All listing output is directed to device name PR:.
7. The default volume is set to the disc which contains the CSS package and the disc must be marked on unprotected.

CSS procedures output error messages, and clean up after themselves, deleting temporary files, etc. A Help file is available for each procedure and provides information about using the procedure. As an example, if the user is having difficulty using the EDIT procedure, typing

HELP EDIT

causes the file EDIT.HLP, which contains tutorial information, to be output to the console.

The Operational Data section contains references to relevant documentation for each procedure.

COMMANDS

This section contains a description of each procedure provided in this package. In reading the individual procedure descriptions, pay particular attention to the terms fn and fd. The term fn refers to a Disc File name, specifically excluding any extension field. A volume name, however, may optionally be included in fn if different from the Default System Volume. The term fd, unless otherwise noted, refers to any device name or an entire Disc File Descriptor which can contain an Extension Field.

Procedure names must be typed exactly as specified. The following characters are used to describe Command Syntax and should not be typed when the actual Command is entered:

Square brackets [xxx] — Used to indicate an optional argument. An argument enclosed in square brackets may be specified or omitted. If omitted and followed by other arguments, its preceding comma must be typed.

Braces {a / b} — Used to indicate a choice between two alternatives which are listed vertically within the braces.

FORTRAN Compile, Load and Go

Name: FORTCLG

Purpose: Perform a FORTRAN IV compile, task establishment, load and start.

Format: FORTCLG fn

Function: A FORTRAN main program and any number of subroutines contained in file fn.FTN are compiled, established, loaded, and started.

Operational Notes: File FTNRTL.OBJ is the assumed filename of the object format FORTRAN Run Time Library. If any errors are detected during the FORTRAN compilation, the process is halted after all programs have been compiled with no new files remaining. If file fn.TET exists it is used as TET input for the task establishment; otherwise, a default establishment is performed as described in the Task Establishment Defaults section. File fn.OBJ is created and deleted in the course of the process and file fn.TSK remains as a result of the process. Listing output is directed to the printer. If file fn.ASN exists, it is called to make Logical Unit (LU) assignments for the resulting task, otherwise DEFAULT.ASN is called to make LU assignments. The FORTRAN $BATCH option must be used when compiling a main program and a number of subroutines.

Messages:

***FORTCLG: MISSING PARAMETER (FN)
-- Filename specifying source file to be compiled was omitted.

***FORTCLG: File fn.FTN NON-EXISTENT
– Source file to be compiled cannot be found.

***FORTCLG: ERROR DETECTED DURING COMPILE
-- Syntax errors were detected by the FORTRAN Compiler. No new files are generated and the procedure stops.

***FORTCLG: ERRORS DETECTED DURING TASK ESTABLISHMENT
– TET ended abnormally. Refer to the TET Command log listed on the printer for nature of error.

Example: FORTCLG GRAPH

File GRAPH.FTN is compiled, established, and loaded. If file GRAPH.TET exists, it is used during the establishment, otherwise a default establishment is performed; if file GRAPH.ASN exists, it is called to make LU assignments, otherwise DEFAULT.ASN is called. The task is then started.

FORTRAN Compile

Name: FORT

Purpose: Perform a FORTRAN IV compile, and task establishment.

Format: FORT fn

Function: Compile, and establish a FORTRAN main program and any number of FORTRAN subroutines contained in file fn.FTN.

Operational Notes: File FTNRTL.OBJ is the assumed filename of the object format FORTRAN Run Time Library. If any errors are detected during the FORTRAN compilation, the process is halted after all programs have been compiled, with no new files remaining. If file fn.TET exists, it is used as Task Establisher (TET) input for the task establishment, otherwise a default establishment is performed as described in the Task Establishment Defaults section. File fn.OBJ is created and deleted in the course of the process; file fn.TSK is the result of the process. Listing output is directed to the printer. The FORTRAN $BATCH option must be used when compiling a main program and a number of subroutines.

Messages:

***FORT: MISSING PARAMETER (FN)
– Filename specifying source file to be compiled was omitted.

***FORT: FILE fn.FTN NON-EXISTENT
– Source file to be compiled cannot be found.

***FORT: ERRORS DETECTED DURING COMPILE
– Syntax errors were detected by the FORTRAN Compiler. No new files are generated and the procedure stops.

***FORT: ERRORS DETECTED DURING TASK ESTABLISHMENT
– TET ended abnormally. Refer to the TET Command log listed on the printer for nature of error.

Example: FORT ELIZER

File ELIZER.FTN is compiled, and established, resulting in file ELIZER.TSK. If file ELIZER.TET exists, it is used as TET input, otherwise a default establishment takes place.

CAL Assembly

Name: CAL

Purpose: Perform a CAL assembly.

Format: CAL fn [,CAL options]

Function: File fn.CAL is assembled by CAL/16D.

Operational Notes: Up to 12 starting options can be specified to be passed to CAL. The CAL source program contained in file fn.CAL is assembled with the resulting object being placed in file fn.OBJ.

The assembly listing is directed to the printer. CAL source library input, LU 7, is assigned to file PCB16.CAL, if it exists.

If any assembly errors are detected by CAL, processing is halted after the assembly with no new files remaining. Under normal operation, file fn.OBJ remains as the result of the process. For optimum efficiency, the CAL SCRAT option should not be specified.

Messages:

***CAL: MISSING PARAMETER (FN)
– Filename specifying source file to be assembled was omitted.

***CAL: FILE fn.CAL NON-EXISTENT
– Source file to be assembled cannot be found.

***CAL: ERRORS DETECTED DURING ASSEMBLY
-- Syntax errors were detected by the CAL Assembler. No new files are generated.

Example: CAL OP,CROSS

File OP.CAL is assembled resulting in file OP.OBJ; a cross reference is produced on the listing output.

CAL Assembly, Load and Go

Name: CALCLG

Purpose: Perform a CAL/16D assembly, load and execution.

Format: CALCLG fn [,CAL options]

Function: File fn.CAL is assembled, loaded into the background, and started. Object file fn.OBJ is produced.

Operational Notes: Up to 12 starting options can be specified to be passed to CAL. The CAL source program contained in file fn.CAL is assembled, loaded, and started. If file fn.ASN exists, it is called as a CSS file to make Logical Unit assignments for the task before starting it, otherwise DEFAULT.ASN is called.

The assembly listing is directed to the printer. CAL's source library input unit (LU7) is assigned to file PCB 16. CAL if existent. If any assembly errors are detected by CAL, processing halts after the assembly with no new files remaining.

Under normal operation file fn.OBJ remains as a result of the process. For optimum efficiency, the CAL SCRAT option should not be specified.

Messages: ***CALCLG: MISSING PARAMETER (FN)
– Filename specifying source file to be assembled was omitted.

***CALCLG: FILE fn.CAL NON-EXISTENT
– Source file to be assembled cannot be found.

***CALCLG: ERRORS DETECTED DURING ASSEMBLY
– Syntax errors were detected by the CAL Assembler. No new files are generated.

Example: CALCLG PER,ERLST,CROSS

File PER.CAL is assembled, with an error listing and a cross reference, and loaded. If file PER.ASN exists, it is called to make LU assignments, otherwise DEFAULT.ASN is called to make the assignments. The task is then started. File PER.OBJ remains after completion.

Macro Expansion and Assembly

Name: MAC

Purpose: Perform a CAL MACRO expansion and assembly.

Format: MAC fn [,CAL options]

Function: File fn.MAC is expanded and assembled.

Operational Notes: File fn.MAC is expanded using the CAL MACRO Processor. Logical Unit assignments to various macro libraries are made by calling CSS procedure MACROLIB.ASN which, unless modified to suit the needs of a particular installation, assigns LU 7 to the standard system macro library (filename MACROLIB.MAC).

Up to 12 starting options can be specified to be passed to CAL/16D for the subsequent assembly. CAL's source library input LU 7 is assigned to file PCB16.CAL if it exists.

Intermediate file fn.CAL is created and deleted during the course of this CSS procedure, and file fn.OBJ is produced as a result of the procedure. All listing output is directed to the printer. For optimum efficiency the CAL SCRAT option should not be specified.

Messages: ***MAC: MISSING PARAMETER (FN)
– Filename specifying source file to be expanded and assembled was omitted.

***MAC: FILE fn.MAC NON-EXISTENT
– Source file to be expanded and assembled cannot be found.

***MAC: ERRORS DETECTED DURING MACRO EXPANSION
– Errors were detected by the Macro Processor. Other than the listing provided, no new files are generated and the procedure stops.

Example: MAC LUM,CROSS

File LUM.MAC is expanded using macro libraries assigned by MACROLIB.ASN, assembled with cross reference listing, resulting in file LUM.OBJ.

Macro Expansion, Assembly, Load and Go

Name: MACCLG

Purpose: Perform a CAL MACRO expansion, assembly, load and start.

Format: MACCLG fn [,CAL options]

Function: File fn.MAC is expanded, assembled, loaded into the background, and started. Object file fn.OBJ is produced.

Operational Notes: File fn.MAC is expanded using the CAL MACRO Processor. Logical Unit assignments to macro libraries are accomplished by calling CSS procedure MACROLIB.ASN which, unless modified to suit the needs of a particular installation, assigns LU 7 to the standard system macro library (filename MACROLIB.MAC).

Up to 12 starting options can be specified to be passed to CAL/16D to effect the subsequent assembly. CAL's source library input LU is assigned to file PCB16.CAL if it exists. If file fn.ASN exists, it is called as a CSS file to make Logical Unit assignments for the task before starting it, otherwise DEFAULT.ASN is called.

Intermediate file fn.CAL is created and deleted during the course of the procedure and file fn.OBJ remains as a result of the procedure. All listing output is directed to the printer. For optimum efficiency, the CAL SCRAT option should not be specified.

Messages: ***MACCLG: MISSING PARAMETER (FN)
– Filename specifying source file to be expanded and assembled was omitted.

***MACCLG: FILE fn.MAC NON-EXISTENT
– Source file to be expanded and assembled cannot be found.

***MACCLG: ERRORS DETECTED DURING MACRO EXPANSION
– Errors were detected by the Macro Processor. No new files are generated and the procedure stops.

***MACCLG: ERRORS DETECTED DURING ASSEMBLY
– Syntax errors were detected by the CAL Assembler. No new files are generated and the procedure stops.

Example: MACCLG FILE12

File FILE12.MAC is expanded using macro libraries assigned by MACROLIB.ASN, assembled into FILE12.OBJ, loaded, and started.

Edit a File

Name: EDIT

Purpose: Edit a file.

Format: EDIT fd_1 [,fd_2] [,Vol]

Function:

1. If fd_2 is omitted:

   OS EDIT is loaded into the background. Command input to the editor is taken from the console. Fd_1 is edited with the resulting file replacing the existing fd_1. Should OS EDIT terminate abnormally, the existing fd_1 is left unchanged. If fd_1 is a new file a message to this effect is output and it is allocated (Indexed file, LRECL = 80).

2. If fd_2 is specified:

   As above except that the file to be edited is taken from fd_1 with the resultant file replacing fd_2. Fd_1 is unchanged.

Operational Notes: Fd_2 must specify a disc file with a logical record length of 80 bytes. If fd_2 is omitted, fd_1 must specify a disc file with a logical record length of 80 bytes; if fd_2 is specified, fd_1 may specify any device or file. Neither fd_1 or fd_2 may contain a volume descriptor. Vol indicates the volume on which the file to be edited resides. If omitted, the default system volume is assumed.

Since no files are changed if OS EDIT terminates abnormally, an editing session can be aborted by pressing the Break key, and cancelling the background partition.

By specifying fd_2 as in (2) above, an unedited backup file is retained (fd_1).

Messages:

***EDIT: MISSING PARAMETER (FD1)
– File Descriptor specifying file to be edited was omitted.

***EDIT: ABORT, NO NEW FILES GENERATED
– OS EDIT ended with a non-Zero Return Code, no files have been altered nor new files created.

***EDIT: NEW FILE fd
– A new file results from this editing session and is being allocated.

Examples:

EDIT TEST.CAL

File TEST.CAL is edited, with the edited version replacing the original version.

EDIT OLD.FTN,NEW.FTN

File OLD.FTN is edited. The edited version of OLD.FTN becomes NEW.FTN. File OLD.FTN is retained in its original form.

Establish a Task

Name: ESTAB

Purpose: Establish an object program as a Task, Reentrant Library, or Overlay.

Format: ESTAB fn [,id, bias]

Function: Object program fn.OBJ is established with the Task Establisher (TET). If file fn.TET exists, it is used as task establishment command input for TET. If file fn.TET does not exist, a default establishment is performed. In this case id specifies a one to four (one to three for extended memory) character identification for the resultant task image, and bias specifies the partition bias.

Operational Notes: If fn.TET exists, id and bias are ignored. The object programs to be established are specified in the INCLUDE statements contained in file fn.TET, and the image output file is specified in the BUILD statement contained in file fn.TET.

If fn.TET does not exist, the result is file id bias .TSK (the id and bias parameters are concatenated to form the filename). If id and bias are not specified, the task is biased at the background partition and placed on file fn.TSK.

Messages: ***ESTAB: MISSING PARAMETER (FN)
– fn was not specified.

***ESTAB: FILE fn.OBJ NON-EXISTENT
– The object file to be established cannot be found.

***ESTAB: MISSING PARAMETER (ID)
– Bias was specified, but id was omitted.

***ESTAB: MISSING PARAMETER (BIAS)
– id was specified, but bias was omitted.

Examples: If SUBPROG.TET exists,

ESTAB SUBPROG

establishes SUBPROG.OBJ according to the TET input contained in SUBPROG.TET.

If SUBPROG.TET does not exist,

ESTAB SUBPROG,SUBP,9000

establishes SUBPROG.OBJ as a TASK with all task attributes set to TET defaults, and biased at 9000. The task is placed in file SUBP9000.TSK.

If SUBPROG.TET does not exist,

ESTAB SUBPROG

establishes SUBPROG.OBJ as a TASK with all task attributes set to TET defaults, and biased at the background partition. The task is placed in file SUBPROG.TSK.

Copy an ASCII File

Name: COPYA

Purpose: Copy an ASCII file.

Format: COPYA fd_1,fd_2 [,**label] [,r] [,f] [,l]

Function: ASCII file fd_1 is copied to fd_2; fd_2 is allocated if necessary.

Operational Notes: Optional modifiers may be specified as follows:

** label – specifies an ASCII label to be searched for on fd_1 before copying.

r – specifies the record length of the file being copied. DEFAULT = 80.

f – specifies the starting record where copying is to begin. DEFAULT = first record.

l – specifies the record number where copying is to end. DEFAULT = last record (EOF/EOM).

File fd_2 is allocated, if necessary, as an Indexed file with a logical record length of 80 bytes or r, if specified.

Copying starts at record 1 of fd_2. If file fd_1 is smaller than file fd_2 the latter is possibly left with erroneous trailing records. In such a case the user is advised to delete file fd_2 before the COPYA procedure is executed and allow fd_2 to be allocated, thus ensuring that there are no trailing records. In using COPYA with Paper Tape, Card Readers, or Console-like devices, the end of the input file should be denoted by a record containing the characters '/*' in columns 1 and 2. Mag tape and Cassette files should use a filemark to denote EOF. Disc files contain a built-in EOF indication (Contiguous Disc files support filemarks if required).

Messages:

***COPYA: MISSING PARAMETER (FD1)
– File Descriptor specifying file to be copied FROM was omitted.

***COPYA: MISSING PARAMETER (FD2)
– File Descriptor specifying file to be copied TO was omitted.

***COPYA: FILE fd NON-EXISTENT
– File to be copied from cannot be found.

***COPYA: NEW FILE fd_2
– fd_2 is a new file and is being allocated.

Examples:

COPYA CR:,NEWFILE.FTN

Copy ASCII file from card reader to NEWFILE.FTN.

COPYA REF.LST,PR:,**TCB,132

Copy from file REF.LST to printer PR: starting at ASCII label **TCB with 132 byte records.

Copy a Binary File

Name: COPYB

Purpose: Copy a binary file.

Format: COPYB fd_1,fd_2 [,label] [,r] [,f] [,l]

Function: Binary file fd_1 is copied to fd_2. Fd_2 is allocated if necessary.

Operational Notes: Optional modifiers may be specified as follows:

label – specifies a binary label to be searched for on fd_1, before copying is to begin.

r – specifies record length of the file being copied. DEFAULT = 108.

f – specifies the starting record number where copying is to begin. DEFAULT = 1.

l – specifies the record number where copying is to end. DEFAULT = last record (EOF/EOM).

File fd_2 is allocated, if necessary, as an Indexed file with a logical record length of 108 bytes or r, if specified.

Copying starts at record 1 of fd_2. If file fd_1 is smaller than file fd_2, the latter is possibly left with erroneous trailing records. In such a case the user is advised to delete file fd_2 before the COPYB procedure is executed and allow fd_2 to be allocated, thus ensuring that there are no erroneous trailing records.

In using COPYB with files that do not have an EOF indication (Paper Tape), operator intervention is necessary to complete the operation. In such a case COPYB terminates with an 84xx I/O-error. At that time the user should terminate by typing TA .BG;CA which properly terminates the copy operation.

Messages:

***COPYB: MISSING PARAMETER (FD1)
– File descriptor specifying file to be copied FROM was omitted.

***COPYB: MISSING PARAMETER (FD2)
– File descriptor specifying file to be copied TO was omitted.

***COPYB: FILE fd_1 NON-EXISTENT
– File to be copied from cannot be found.

***COPYB: NEW FILE fd_2
– fd_2 is a new file and is being allocated.

Examples:

```
COPYB RES.OBJ,CAS1:,ERR
```

Copy object file from file RES.OBJ to Cassette CAS1: starting at object label ERR.

```
COPYB PTRP:,CAL.OBJ
I/O-ERR:8413
*TA.BG;CA
```

Copy object file from Paper Tape Reader(PTRP:) to Disc File CAL.OBJ.

Copy a Task

Name: COPYT

Purpose: Copy a file containing an established Task, Reentrant Library, or Overlay (i.e., TET output).

Format: COPYT fd_1, fd_2

Function: Image file fd_1 is copied to file fd_2; fd_2 is allocated (as a 256 byte record length Indexed file) if necessary.

Operational Notes: Copying starts at record 1 of fd_2. If file fd_1 is smaller than file fd_2, the latter is left with extraneous trailing records. In such a case, the user is advised to delete file fd_2 before the COPYT procedure is allocated thus ensuring that there are no erroneous trailing records.

In using COPYT with files that do not have an EOF indication (e.g. Paper Tape), operator intervention is necessary to complete the operation. In such a case COPYT terminates with an 84xx I/O-error. At that time the user should terminate by typing TA .BG;CA which properly terminates the copy operation. COPY may not be used to copy the OS file itself.

Messages:

***COPYT: MISSING PARAMETER (FD1)
– File descriptor specifying file to be copied FROM was omitted.

***COPYT: MISSING PARAMETER (FD2)
– File descriptor specifying file to be copied TO was omitted.

***COPYT: FILE fd_1 NON-EXISTENT
– File to be copied from cannot be found.

***COPYT: NEW FILE fd_2
– fd_2 is a new file and is being allocated.

Examples: COPYT TASK1.TSK,PTRP:

Copy image file TASK1.TSK to Paper Tape Reader/Punch PTRP:

COPYT MAG1:,MAG2:

Copy image file on Mag Tape MAG1: to Mag Tape MAG2:

Load and Execute a Task

Name: RUN

Purpose: Load and start a task.

Format: RUN fn

Function: File fn.TSK is loaded into the background partition. If file fn.TSK does not exist, file fn.OBJ is loaded. If file fn.ASN exists, it is called as a CSS procedure to make Logical Unit assignments for the task, otherwise DEFAULT.ASN is called to make default LU assignments. The task is then started.

Operational Notes: This procedure can be used to run the task image file resulting from a FORT call, the task image file that remains as a result of a FORTCLG, the object file resulting from a CAL or MAC, or the object file that remains as a result of a CALCLG or MACCLG call. See the description of DEFAULT.ASN for a sample of the form file fn.ASN might take.

Messages: ***RUN: MISSING PARAMETER (FN)
– File descriptor specifying program to be loaded was omitted.

***RUN: FILE fn.OBJ NON-EXISTENT
– Neither fn.TSK or fn.OBJ exist.

Example: After a FORTRAN compile and task establishment with:

```
FORT BARGRAPH
```

The user can load and start the resulting task image file with:

```
RUN BARGRAPH
```

Generate a New Operating System

Name: SYSGEN

Purpose: Generate an OS/16MT2 operating system.

Format: SYSGEN osid [,fd] [,NLIST] [,EX]

Function: The configuration statements from fd are used to configure an OS/16MT2 system. The resultant system is placed on file OS16osid, where osid has the format XXXX.ext.

Operational Notes: The Configuration Utility Program (CUP/16) is run using fd for configuration input. The default fd is CUPIosid. Output is put on file CUPOosid, and a listing of the CUP statements is output to the printer. Configuration statements are prepared as described in Chapter 4 of the *OS/16MT2 System Planning and Configuration Guide,* Publication Number 29-431.

osid must be of the form xxxx.nnn, where xxxx is up to four alphanumeric characters, and nnn is three hexadecimal characters.

NLIST is a CAL/16 option

EX specifies that the SYSGEN is for an extended memory processor.

CAL/16D is used to assemble the three source modules. File CUPOosid is used for CAL copy input. Files EXEC16.CAL, FMGR16.CAL, CMDP16.CAL are used as the system module source input. EXECosid, FMGRosid, CMDPosid are produced as the result of the assemblies.

TET/16 is used to establish the three object modules produced by the assembler along with the proper driver library file (DLIB16.OBJ or DLIB16EX.OBJ). The established system is output to file OS16osid. The system is built whether or not there are assembly errors.

Messages: ***SYSGEN: MISSING PARAMETER (OSID)
– Parameter specifying osid was omitted.

***SYSGEN: FILE fd NON-EXISTENT
– Configuration statement fd cannot be found.

Examples: SYSGEN MT20.000,CR:

A system is generated from configuration statements read from the card reader. The output of CUP is placed on file CUPOMT20.000. The outputs of CAL/16 are placed on files EXECMT20.000, FMGRMT20.000, CMDPMT20.000. The system is produced on file OS16MT20.000.

SYSGEN TEMP.123

A system is generated from configuration statements read from file CUPITEMP.123. The output of CUP is placed on file CUPOTEMP.123. The outputs of CAL/16 are placed on files EXECTEMP.123, FMGRTEMP.123, CMDPTEMP.123. The system is produced on file OS16TEMP.123.

Assign Default Logical Units

Name: DEFAULT.ASN

Purpose: Make default assignments for a task.

Format: DEFAULT.ASN (no arguments)

Function: DEFAULT.ASN is called by any CSS procedure which loads and starts a task to make default Logical Unit assignments for the task. Such Logical Unit assignments are used when no user provided task assignment file exists.

Operational Notes: Unless changed by the user to reflect the needs of particular installation, the contents of DEFAULT.ASN are:

```
ASSIGN  1,CON:
ASSIGN  3,PR:
ASSIGN  5,CON:
ASSIGN  6,PR:
$EXIT
```

INSTALLATION

The following procedure is used to build the OS/16MT2 High Level Operator Command (HLOC) package on disc.

Begin by MARKing the disc on-line, and setting the disc's volume name as the default volume.

```
MARK Disc  name,ON
VOLUME Volume Name
```

The individual procedures are built by calling file HLOC which contains the package. Enter the file name from the system console:

```
HLOC
```

After generating the procedures, HLOC checks for the presence of the required support programs on the DISC and outputs a list of those not present. Such programs should be built before the calling CSS procedure is used. The next section summarizes utility requirements.

Finally, files DEFAULT.ASN and MACROLIB.ASN should be examined for compatibility with the needs of the particular installation and modified if necessary (see the descriptions of procedures DEFAULT.ASN, and MAC or MACCLG respectively).

NOTE

The package requires an OS/16MT2 configured for a minimum of 2 levels of CSS nesting and a minimum command buffer length of 40 bytes. A command buffer length of 72 is necessary to call a HELP file.

OPERATIONAL DATA

Task Establishment Defaults

The following task establishment parameters are used when a default task establishment is selected by the user.

Establish as a TASK

OPTIONS:	User task Floating point Not memory resident Arithmetic Fault Pause SVC 6 pause Compatible Double Floating Point
GET:	X'400' bytes (the amount needed by a FORTRAN program)
BIAS:	The task is biased to run in the background partition if no bias is specified in the command.
PRIORITY:	Initial priority = 128, maximum priority = 128
TSW:	Status = 0000 start address = that address contained in the object code or UBOT+X'84' if not available.

Space is reserved for User Dedicated Locations (e.g., REL code starts at UBOT+X'84').

If the establishment is a result of the FORT or FORTCLG procedures, an EDIT of file FTNRTL.OBJ is performed.

If the above establishment defaults are unsuitable to a particular installation they can be changed by altering (via the EDIT Procedure call) the default establishment parameters contained in the ESTAB.CSS, FORT.CSS, and FORTCLG.CSS segments of file HLOC.CSS.

Utility Programs

The following utility programs and libraries are required by the package, and must be on the disc under the filenames listed below.

Program No.	Name	Filename	Used By
03-054	Extended Fortran IV Compiler	FORT4.OBJ	FORT,FORTCLG
		FORT4.OVL	(Required for the overlaid compiler only)
07-040	Fortran RTL	FTNRTL.OBJ	FORT,FORTCLG
03-101F02	CAL/16D Assembler	CAL16D.OBJ	CAL,CALCLG,MAC,MACCLG
03-101F03	CAL/16DS Assembler	CAL16DS.OBJ	SYSGEN
03-084	CAL Macro Processor	CALMACRO.OBJ	MAC,MACCLG
03-097	CUP/16	CUP.OBJ	SYSGEN
07-091	OS/16 MT2 Driver Library	DLIB16.OBJ	SYSGEN
03-087	OS/16 MT2 Task Establisher (TET/16)	TET.OBJ	ESTAB,FORT,FORTCLG,SYSGEN
03-056	OS COPY	COPY.OBJ	COPYA, COPYB, COPYT
03-063	OS EDIT	EDIT.OBJ	EDIT

Source Libraries

Program No.	Name	Filename	Used By
07-090	OS/16 MT2 Parameters and Control Blocks[1]	PCB16.CAL	CAL, CALCLG, MAC, MACCLG
07-068	System MACRO Library[2]	MACROLIB.MAC	MAC,MACCLG
07-087	OS/16 MT2 Executive	EXEC16.CAL	SYSGEN
07-088	OS/16 MT2 File Manager	FMGR16.CAL	SYSGEN
07-089	OS/16 MT2 Command Processor	CMDP16.CAL	SYSGEN

NOTES

1. Presence of PCB16.CAL is optional. See description of CAL.CSS procedure.
2. Presence of MACROLIB.MAC is the default case. See description of MAC.CSS procedure.

APPENDIX 1

OPERATOR COMMAND SUMMARY

ALLOCATE	fd {CONTIGUOUS,fsize [,keys] / INDEX [,lrecl [/bsize] [/isize] [,keys]]} defaults: lrecl = 108, bsize = 1, isize = 1, keys = 0000
ASSIGN	lu,fd [, [access priv.] [,keys]] access priv. = SRO, ERO, SWO, EWO, SRW, SREW, ERSW, or ERW defaults: access priv. = SRW, keys = 0000
BIAS	[{adrs / *}] default: adrs=0
BFILE	fd [,lu]
BRECORD	fd [,lu]
BUILD	fd
CANCEL	
CLEAR	fd
CLOSE	{lu [,lu] ... / ALL}
CONTINUE	
DELETE	fd [,fd] ...
DISPLAY DEVICES	[,fd]
DISPLAY FILES	, [voln:] [filename] [.ext] [,fd]
DISPLAY LU	[,fd]
DISPLAY MAP	[,fd]
DISPLAY PARAMETERS	[,fd]
DISPLAY REGISTERS	[,fd]
DISPLAY TIME	[,fd]

ENDB

EXAMINE adrs [{,n /adrs}]

FFILE [fd] [,lu]

FILES [voln:][filename [.ext][,fd] default volume name is system volume

FRECORD [fd] [,lu]

INITIALIZE fd,voln [,READCHECK]

LDBG fd [,bias]

LFGR partition, taskid [,fd] [, bias] [,CSEG]

LOAD {taskid .BG} [,fd] [,*]

MARK fd, {ON [{,PROTECT ,OS ,ROLL} ...] [,OS] / OFF}

MODIFY adrs, [data] , [data] ...
default data is zero

OPTIONS opt [,opt] ...
opt= AFC,AFP,RES,NONRES,UT,ET,ROLL,
NOROLL,FLOAT,NOFLOAT,DFLOAT,
NODFLOAT,SVCP,SVCC,COMP, or NOCOMP

PAUSE

PRINT [fd[,n][,D][,T]]

RENAME oldfd,newfd

REPROTECT fd,keys

REWIND [fd] [,lu]

RW [fd] [,lu]

SAVE fd

SEND (up to 64 character string)

SET CODE n

SET LOG	[fd [,COPY]]
SET PARTITION	name/start [,name/start] ....
SET PRIORITY	n
SET TIME	[mm/dd/yy] [,hh:mm:ss] or [dd/mm/yy] [,hh:mm:ss]
START	[adrs] [,args to prog]
TASK	[taskid]
VOLUME	[voln] or [voln/SPL]
WFILE	fd [,lu]
XDELETE	fd[,fd] ....
$BUILD	fd
$CLEAR	
$COPY	
$ELSE	
$ENDB	
$ENDC	
$EXIT	
$IFE	n
$IFG	n
$IFL	n
$IFNE	n
$IFNG	n
$IFNL	n
$IFNNULL	@n
$IFNULL	@n

$IFNX	fd
$IFX	fd
$JOB	
$NOCOPY	
$SKIP	
$TERMJOB	

APPENDIX 2

COMMAND ERROR RESPONSE SUMMARY

Response to a command error is a message in the general format:

XXXX-ERR YYYY ZZZZ #= N

XXXX is the general error class descriptor.

YYYY is an error descriptor which further defines the error.

ZZZZ is an informational field which specifies an entity helpful in determining the cause of the error.

N is the number of the command for multiple commands on a line; if N=1, this field is omitted

CLASS	TYPE	INFORMATION	MEANING
ALLO	PRIV	FD=fd	Allocate failed because volume is write protected.
ALLO	VOL	FD=fd	Allocate failed because specified volume not found in system.
ALLO	NAME	FD=fd	Allocate failed because file name currently exists on specified volume.
ALLO	SIZE	FD=fd	Allocate failed because specified size is too large.
ALLO	TYPE	FD=fd	Allocation failed because specified volume is a device instead of a direct access volume.
ASGN	LU	LU=lu	Assignment necessary for command failed because of illegal LU specification.
ASGN	VOL	FD=fd	Assignment required by command failed because specified volume or device not found in system.
ASGN	NAME	FD=fd	Assignment required by command failed because specified file name not found on specified volume.
ASGN	PROT	FD=fd	Assignment required by command failed because specified protection keys do not match keys associated with specified fd.
ASGN	PRIV	FD=fd	Assignment required by command failed because requested access privileges could not be granted.
ASGN	BUFF	none	Assignment required by command failed because of insufficient system space to contain buffer.
ASGN	ASGN	FD=fd	Assignment required by command failed because specified logical unit is already assigned.
ASGN	CONN	FD=fd	Assignment required by command failed because specified fd is an SVC 6 connectable device.
ASGN	EOM	FD-fd	Assignment required by command failed because of End of Medium I/O error. If error occurs on disc device, check size of disc (2.5MB, 10MB, etc.) was specified correctly when the system was SYSGENd.

CLASS	TYPE	INFORMATION	MEANING
ASGN	RECV	FD=fd	Assignment failed because recoverable I/O error encountered on device referenced by fd.
ASGN	FUNC	FD=fd	Assignment required by command failed because fd specifies a file of illegal type (e.g.; a chained file, created under OS/32, or an indexed file on a system SYSGENd without an INDEX cup statement).
ASGN	SIZE	FD=fd	Assignment failed because there is no room on the disc for an index and a data block.
ASGN	TYPE	FD=fd	Assign failed because specified fd is an off-line device.
CLOS	LU	LU=lu	Close required by command failed because LU specified is illegal.
CLOS	DU	FD=fd	Close required by command failed because device referenced by fd is unavailable to system. This results from I/O error encountered during Close.
CLOS	UNRV	FD=fd	Close required by command failed because unrecoverable I/O error encountered on device referenced by fd.
CLOS	RECV	FD=fd	Close required by command failed because recoverable I/O error encountered on device referenced by fd.
CSS	LVL	none	CSS call would cause CSS level to be greater than SYSGENed value or number of nested $IFs greater than 255.
CSS	BUFF	none	CSS parameter substitution caused a command line to exceed SYSGENed command length.
DELE	PROT	FD=fd	Delete failed because fd specified has non-zero protection keys.
DELE	VOL	FD=fd	Delete failed because specified volume not online to system.
DELE	NAME	FD=fd	Delete failed because specified file name not found on specified volume.
DELE	ASGN	FD=fd	Delete failed because specified fd currently assigned.
DELE	FD	FD=fd	Delete failed because of invalid syntax of fd.
DELE	DU	FD=fd	Delete failed because of device unavailable I/O error encountered on device referenced by fd.
DELE	UNRV	FD=fd	Delete failed because of unrecoverable I/O error encountered on device referenced.
DELE	RECV	FD=fd	Delete failed because recoverable error encountered on device referenced.
DELE	EOM	FD=fd	Delete failed because of End of Medium I/O error. If error occurs on disc device, check size of disc (2.5MB, 10MB etc) was specified correctly when the system was SYSGENd.
DELE	FUNC	FD=fd	Delete failed because fd specifies a file of illegal type (e.g.; a chained file, created under OS/32, or an indexed file on a system SYSGENd without an INDEX cup statement).

CLASS	TYPE	INFORMATION	MEANING
DELE	TYPE	FD=fd	Delete failed because fd specifies a device.
DELE	PRIV	FD=fd	Delete failed because volume which contains the specified fd is write protected.
DUPL	none	none	Specified direct access device contains a volume name which is identical to the name of another volume or device.
FORM	none	POS=text	Invalid command syntax. Usually caused by invalid delimiters. The information field indicates the position in the command where the syntax error was detected.
FUNC	SEQ	none	Command entered out of sequence. See particular command definition for restrictions.
FUNC	ARG	none	Not enough memory between UTOP and CTOP to contain the specified arguments to program.
FUNC	TASK	none	Command requires a currently selected task and there is none.
IO	DU	FD=fd	Device unavailable error encountered on specified fd.
IO	IOFN	FD=fd	Illegal function error encountered on specified fd.
IO	EOM	FD=fd	End of Medium error encountered on specified fd.
IO	EOF	FD=fd	End of file encountered on specified fd.
IO	UNRV	FD=fd	Unrecoverable error encountered on specified fd.
IO	RECV	FD=fd	Recoverable error encountered on specified fd.
IO	LU	LU=lu	Illegal or unassigned Logical Unit specified by lu.
LOAD	DCHN	none	Invalid REF-DEF chain encountered during load.
LOAD	CKSM	none	Checksum error encountered during load.
LOAD	SEQ	none	Object record out of sequence encountered during load.
LOAD	OBJ	none	Invalid object format is encountered during load.
LOAD	MEM	none	Insufficient memory in background partition to contain program.
LOAD	PRES	TSK=taskid	Specified TASK ID already present in system.
LOAD	NAME	TSK=taskid	Specified TASK ID is illegal.
LOAD	SEG	none	Specified task requires Task Common or Library partition that does not exist or is in incorrect 32kb module.
LOAD	LIB	none	Invalid format item in Loader Information Block.
LOAD	LU	none	Load LU is invalid or unassigned or Load LU attributes are invalid.
LOAD	PFUL	none	Partition required by specified task not vacant and cannot be rolled out.
LOAD	NOFL	none	Floating Pt not supported by sysgen.

CLASS	TYPE	INFORMATION	MEANING
LOAD	NPRT	none	Partition required by specified task not in system.
LOAD	EOM	none	Load failed because of End of Medium I/O error. If error on disc device, check size of disc (2.5MB, 10MB etc) was specified correctly when the system was SYSGENd.
LOGQ	none	none	System log queue has been overflowed. Log message has been lost.
MNEM	none	none	Command not recognized as command or CSS call.
OVLY	none	none	System overlay volume marked off-line.
PACK	none	none	Too few good sectors to allocate the allocation map and directory. Pack should be reformatted.
PARM	MNEM	POS=text	Mnemonic parameter was not recognized. Parameter in error specified by text.
PARM	VAL	POS=text	Parameter denoted by text, specifies an invalid value.
PARM	TASK	POS=text	Parameter denoted by text, contains invalid syntax for task ID.
PARM	FD	POS=text	Parameter denoted by text, contains invalid syntax for a File Descriptor.
PARM	PRQD	POS=text	A required parameter following the field denoted by text is missing.
RENM	NAME	FD=fd	New name already exists.
RENM	FD	FD=fd	Rename failed because of syntax error in specified fd.
RENM	DU	FD=fd	Rename failed because of device unavailable I/O error encountered on device referenced by fd.
RENM	UNRV	FD=fd	Rename failed because of unrecoverable error I/O error encountered on device referenced by fd.
RENM	RECV	FD=fd	Rename failed because of recoverable I/O error encountered on device referenced by fd.
RENM	TYPE	FD=fd	Attempt to rename the NULL device.
SEND	PRES	TSK=taskid	Print failed because Spooler not active.
SEND	NMSG	TSK=taskid	Send failed because task cannot receive message.
STAT	ACTV	none	Specified partition contains an active task.
STAT	PRIV	none	Specified device or file is assigned or contains assigned files and command requires exclusive access.
STAT	LU	none	LU not assigned as specified.
STAT	VAL	none	Specified partition addresses are out of order.
STAT	NOFF	none	Specified device was dismounted without being marked off. The volume can be marked on with protect. Disc Integrity Check must be run.
VOLU	none	none	Specified direct access volume contains an invalid volume name

APPENDIX 3

OS/16 SYSTEM MESSAGES

MESSAGE	DESCRIPTION
OS/16 MT2 01-00	System initialization message.
PWR RSTR-RESET DEV	Recovery from power failure. Devices must be initialized and RUN entered at the display panel (core-memory systems) or the run key "◇" depressed (Series Sixteen systems) for restoration to continue
ILGL INST AT XXXX	Illegal instruction at XXXX in task code. Results in task being PAUSEd.
ILGL SVC XXXX	Illegal SVC at XXXX in task code. Results in task being PAUSEd.
ILGL SVC ADDR XXXX	Illegal address in SVC at XXXX in task code. Results in task being PAUSEd.
MEMF XXXX	Memory parity error at XXXX. Results in task being PAUSEd.
MEMPROT ERR XXXX	Memory Violation PSW loc = XXXX. Results in task being PAUSEd. If the memory protect option is enabled, the background task is protected from writing over system and foreground areas. Foreground tasks are protected from writing over system and background areas.
ARITH ERR XXXX	Arithmetic fault has occurred, PSW loc = XXXX. Results in task being PAUSEd unless the arithmetic continue option is specified.
END OF TASK YYYY	Task identification by Task ID has terminated with Return Code YYYY.
PAUSE	Task identified by Task ID has entered console wait state (Paused).

APPENDIX 4

OS/16 SYSTEM CRASH CODES

CODE	DESCRIPTION
2	Invalid queue entry on the Command Processors's queue
7	Invalid VMT during mark processing
100	Arithmetic fault in system code X'48' = PSW at time of fault
101	Memory Protect fault in system code.
102	Illegal instruction in system code IIIPSW = PSW at time of interrupt
121	No OS file found for overlayed system
132	Illegal SVC in system code
142	Illegal address in SVC in system code
152	Memory parity error in system code X'38' = PSW at time of interrupt
153	Attempt at recovery from power failure on a system with no power fail recovery logic
154	Attempt to run an OS SYSGENed for extended memory on a processor with less than 64KB, or when the OS itself is > 32KB
162	Illegal interrupt on Device address 0
321	Attempt to release a free sector
330	Attempt to manipulate allocation for an invalid sector address
401	Attempt to boot load a hardware protected or completely full disc with incorrect directory. The disc from which the OS is being boot loaded must be marked on without hardware protection. Once this is done, the OS can be reloaded with the hardware protection in effect if desired. If subsequent load attempts fail, there is insufficient disc storage available for marking the disc on. Delete a file on the disc using another OS.
66666666	Console device is unavailable – recovers when console device is restored.
99999999	OS volume is unavailable – recovers when volume is restored.

OS/16 BOOTSTRAP LOADER ERROR CODES

CODE	DESCRIPTION
1	Unable to find the specified OS file on the disc. i.e. VOLN: OS16XXXX.NNN where XXXX = up to four optional characters NNN = three hexadecimal digits specified in location X'7E'as X'0NNN'
DNXX	I/O Error occurred while attempting to access the disc. DN = device number on which error occurred. XX = device status byte

APPENDIX 5

SUMMARY OF TET/16 COMMANDS

COMMAND SYNTAX	SEQUENCE CONSIDERATIONS	VALID FOR:			
		TA	RL	TC	OS
AMAP [fd]	anywhere	X	X	X	X
BATCH	anywhere	X	X	X	X
BIAS[xxxx [,xxxx]]	must follow ESTAB and precede INCLUDE	X	X	X	
BUILD {TASK / RL / TCOM / OVLY / OS} ,fd	must follow INCLUDE				
EDIT fd	must follow INCLUDE	X			X
END	last	X	X	X	X
ESTABLISH {TASK / RL / TCOM / OS}	must precede everything except VOLUME and REMOTE	X	X	X	X
GET xxxx		X			
INCLUDE fd [,program / label]	must follow ESTABLISH	X	X	X	X
INTERACTIVE	anywhere	X	X	X	X
JOB name	anywhere	X	X	X	X
LOG fd	anywhere	X	X	X	X
MAP [fd]	anywhere	X	X	X	X
NOLOG	anywhere	X	X	X	X
OPTIONS opt [,opt]	must precede all INCLUDEs and EDITs	X			
OVERLAY name [,NEW]	after main segment definition; must have following INCLUDE	X			
PAUSE	anywhere	X	X	X	X
PRIORITY rp,mp	anywhere in pass one	X			
REMOTE	anywhere	X	X	X	X
RESOLVE fd,xxxx [,xxxx]	after INCLUDE	X			
REWIND [fd]	anywhere	X	X	X	X
TSKCOM xxxx	must follow ESTABLISH and precede INCLUDE	X			
TSW status [,start / address]	anywhere in pass one	X			
VOLUME voln	anywhere	X	X	X	X
WFILE [fd]	anywhere	X	X	X	X

APPENDIX 6

TET/16 ERROR MESSAGES

TET/16 error messages contain a TET number and message field.

Example: TET006 I/O DEV ERR (status)

TET number — message field

The TET-number field is normally TET001 in which case it has no special meaning. When it is other than TET001, it serves to better define the condition causing the message. In the previous example the message field explains the problem and shows the relevant I/O status while TET006 indicates the error occurred while outputting the map.

All possible TET/16 messages are listed below in the error table.

The meaning of the table headings are as follows:

MESSAGE:	The message field output
TET NO:	Further identifies the cause of the problem
MEANING:	An explanation of the message.
INTERACTIVE:	TET's action if run in interactive (Command stream from console device)
BATCH:	TET's action if run is batch (Command stream not from console device)
REMOTE:	TET's action if run is remote (REMOTE Command Specified)

MESSAGE	TET NO.	MEANING	INTER-ACTIVE	BATCH	REMOTE
ILG CMD	001	Command verb invalid	Read next command	Abort	Abort
ILG CMD SEQ	001	Command not valid at this point in TET operation	Read next command	Abort	Abort
ILG CMD PARAM		Improper command parameter:	Read next command	Abort	Abort
	001	Illegal parameter			
	002	Missing parameter			
	003	Format invalid			
MEM FULL		TET workspace error:	Abort	Abort	Abort
	001	No room for LIB			
	002	No room for dictionary			
	003	No room for load module			
	004	Build below top of dictionary (invalid chain)			

MESSAGE	TET NO.	MEANING	INTER-ACTIVE	BATCH	REMOTE
ILG OBJ ITEM XX	001	Illegal Object format item in input (XX=item)	Abort	Abort	Abort
ILG COMN		Common processing error:	Abort	Abort	Abort
	001	Common item encountered during Reentrant Library build			
	002	Labeled Common increased in size from previous			
	003	Block data item encountered for TSKCOM during task build			
	004	Task common symbol not TSKCOM during Task Common build			
	005	TSKCOM referenced but no TSKCOM command processed			
	006	Block Data item encountered in wrong module during overlay build			
	007	Blank common referenced in overlay not found in root segment			
NO DCHN END XXXX	001	Endless chain (32767 links) found on resolving reference.	Abort	Abort	Abort
	002	Chain loops (REF=DEF) on resolving reference (XXXX gives DEF or ENTRY address)			
MULT DEFD	001	Multiple definitions	Ignore	Ignore	Ignore
	002	Definition in more than one overlay is referenced by the root			
RESOLVE ERR	001	Second Resolve command gives different partition start address	Ignore	Abort	Abort
ILG GET STOR	001	GET command specifies too large an amount of memory	Read next command	Abort	Abort
(SYMBOL) NOT FOUND	001	Program label search failed	Read next command	Abort	Abort
WRONG PROG	001	New definition or common block encountered on Pass 2	Abort	Abort	Abort
	002	Different definition address encountered on Pass 2			
	003	Different program sizes encountered on Pass 2			
TET ABORTED	001	No scratch or scratch not rewindable in REMOTE	Abort	Abort	Abort
	002	REF-DEF error			
	003	Dictionary error			

MESSAGE	TET NO.	MEANING	INTER-ACTIVE	BATCH	REMOTE
	004	Exceeded maximum number of programs in EDIT libraries (4096)			
	005	Common or Block Data reference without definition			
	006	Error in error handling			
LU (lu) NOT ASSIGNED	001	Optional fd omitted; associated LU not assigned.	Read next command	Abort	Abort
I/O DEV ERR XXXX (fd)		I/O error encountered; xxxx is the SVC 1 returned error status.	Pause	Pause	Abort
	001	Reading Command input			
	002	Reading Object input			
	003	Copying overlays			
	004	Writing to scratch file			
	005	Outputting load module			
	006	Outputting map			
CKSM ERR	001	Checksum error encountered on input file	Abort	Abort	Abort
SEQ ERR	001	Sequence number error encountered on input file (Record lost or out of sequence)	Abort	Abort	Abort
EOM	001	End of medium on LU 1	Pause	Pause	Abort
FD SYNTAX	001	File descriptor syntax-error	Read next command	Abort	Abort
FILE ERROR (xxxx) (fd)		Error encountered; xxxx is the SVC 7 returned error status	Read next command	Abort	Abort
	001	Closing a file			
	002	Assigning a file			
	003	Allocating a file			
	004	Reassigning at end of Pass 1			
	005	Error during reassignment at TET termination			
UNDEFD SYMBOLS	001	Warning that undefined symbols exist	Ignore	Ignore	Ignore
SHORT RECORD (fd)	001	Build file has logical record length less than 256	Abort	Abort	Abort
	002	Scratch file has logical record length less than 108			

MESSAGE	TET NO.	MEANING	INTER-ACTIVE	BATCH	REMOTE
ILG ADRS	001	ORG to address outside partition encountered	Abort	Abort	Abort
	002	ORG to absolute address encountered in overlay			
LOAD PROGRAMS (fd)	001	Prompt if no scratch			
REPOSN SCRATCH	001	Operator prompt for non-rewindable scratch			
TCOM ERR	001	Illegal object item for TSKCOM build	Abort	Abort	Abort
CO FILE RQD	001	Overlayed OS requires CO file	Read next command	Abort	Abort
ILG OV EXTRN	001	Illegal External reference in overlayed OS (See Program Logic Manual 29-434, Chapter 4)	Abort	Abort	Abort

APPENDIX 7

SUMMARY OF OS COPY COMMANDS

DEVICE COMMANDS

BSP	LU,n	backspace device on LU by n filemarks.
WFM	LU	write filemark on device on LU.
SKP	LU,n LU,ALL	skip forward n or ALL filemarks on device on LU.
RWD	LU	rewind device on LU.

FUNCTION COMMANDS

CPYA	FN or **NNNNNNNN,R,N or ALL,F,L	Copy ASCII file.
CPYB	FN or LLLLLL,R,N or ALL,F,L	Copy Binary file.
VERA	FN or **NNNNNNNN,R,N or ALL,F,L	Verify ASCII file.
VERB	FN or LLLLLL,R,N or ALL,F,L	Verify Binary file.
DSPA	FN or **NNNNNNNN,R,N or ALL,F,L	Display ASCII file.
DSPB	FN or LLLLLL,R,N or ALL,F,L	Display Binary file.
LBLA	**NNNNNNNN	Write label for ASCII file
LBLB	LLLLLL	Write label for Binary file.
FNDA	**NNNNNNNN,LU	Search for label on ASCII file.
FNDB	LLLLLL,LU	Search for label on Binary file.
LSTA	LU	List labels on ASCII file.
LSTB	LU	List labels on Binary file.
CONT		Restart an interrupted OSCOPY operation.
P		Pause OS COPY
END		Terminate OS COPY
R126		Set OS COPY to process 126 byte object code records
R108		Reset OS COPY to process 108 byte object code records. (OSCOPY initialized to this mode)

APPENDIX 8

SUMMARY OF SOURCE UPDATER COMMANDS

In the syntax, reference is made to the following operands:

ifd	input source fd. If omitted or '*', defaults to LU 1.
ofd	output source fd. If omitted or '*', defaults to LU 2.
ufd	update commands fd.
label	source label of form **label. If optional and omitted, defaults to current position on fd.
str	ending string on a source line, given to identify the last source record to be processed. Must be specified in quotes. If omitted, stops on // in columns 1 and 2, EOF, or EOM.
col	column number identifying the first column or character number to be checked when searching for the ending string. If omitted, defaults to 1.
n	number of characters counting from the beginning of a source line. If omitted, defaults to 80.
seq/seq1/seq2	sequence number in columns 73 through 80
incr	sequence number increment.

OPERATOR COMMANDS

FIND label [,ifd]	search for label on input fd
COPY [ifd] [,ofd] [,str [,col]]	Copy from ifd to ofd
LIST [ifd] [,label]	search for label on input fd, then list remaining source on LU 3.
TABLE ifd	rewind ifd and print on LU 3 a table of all labels.
VERIFY [ifd][,ofd][,n [,str]]	compare source on ifd with source on ofd, comparing the first n characters on each line. Print on LU 3 any which do not match.
EXCEPTION [ifd] [,ofd]	compare source on ifd with source on ofd, comparing sequence numbers. Print on LU 3 all lines which (by number) appear in one source and not the other, and all lines which (by number) appear in both but do not otherwise match.
RESEQUENCE [ifd] ,seq1 [,incr][,ofd]	input source from ifd, resequence starting from seq1 with increments of incr, output to ofd.
REWIND fd	rewind fd
BUILD ofd	put update into Build mode where only INCLUDE commands are accepted.
INCLUDE ifd, label	search for label on ifd, then copy rest of source to ofd given in BUILD command.
ENDB	end Build mode.
UPDATE ufd [,ifd][,ofd][,str [,col]]	put updater into Update mode where update commands are read from ufd, old source is input from ifd and new source is output to ofd.

UPDATE COMMANDS

INSERT seq	insert the line/s which follow this command immediately after the line with sequence number seq. A line with /* in columns 1 and 2 terminates the insert.
DELETE seq1 [,seq2]	delete from seq1 to seq2 inclusive. If seq2 is omitted, delete seq1 only.
MODIFY seq1	delete line seq1 and replace by single source line which follows this command.
REPLACE seq1	replace line seq1 by the line/s which follow this command. A line with /* in columns 1 and 2 terminates the replacement line/s.
SELECT	insert and/or replace by sequence number source lines which follow. Those already in input source are replaced; those not are inserted in the correct position by number. The source lines are terminated by a line with /* in columns 1 and 2.
ENDUP	terminate UPDATE mode.

APPENDIX 9

SUMMARY OF EDIT COMMANDS

UTILITY COMMANDS	DESCRIPTION
MO [A / B]	Set mode
LL n	Set line length
PA	Pause
KI	Kill edit buffer
SL n	Set list device record length
EN	End editor operation
LU n	Change input file logical unit
BC	Request for byte count
TB n	Set tabulation
TC [C]	Set or display tab character
BT [n_1, n_2, . . . ,n_{20}]	Set or display tab points n_1, . . . ,n_{20}
RW LU	Rewind
WF LU	Write file mark
FF LU [,n]	Skip forward n file marks
BF LU [,n]	Backspace n file marks
FR LU [,n] FS LU [,n]	Skip forward n records
BR LU [,n] BS LU [,n]	Backspace n records

INPUT/OUTPUT COMMANDS	DESCRIPTION	CURRENT LINE
OR L_1,L_2	Output and Read	First line read
OR xS_1x [S_2x] [n]		
RE L_1,L_2	Read into Edit Buffer	First line read
RE xS_1x [S_2x] [n]		
WR L_1,L_2	Write from Edit Buffer	Last line output
WR xS_1x [S_2x] [n]		
ML L_1	Move to a line	L_1
ML xS_1x [n]	Move to a line	Line containing S_1
PL L_1	Move to a line and print	L_1
PL xS_1x [n]		Line containing S_1
UP n	Up n lines from the current line	Previous current line +n
DN n	Down n lines from the current line	Previous current line -n
CH xS_1xS_2x [n]	Change a string	nth line
CC cxS_1x [n]	Change char(s) beginning at a specified column	nth line
PR xS_1x [S_2x] [n]	Print Line	Last line printed
PR L_1,L_2		Last line printed
L_1 [text]	Insert, delete or replace line	L_1
CR	Create text	First line entered
AP	Append text following last line of EB.	First line entered.
IL [L]	Insert lines following current line or line L.	Unchanged or line L.
DL [n]	Delete n lines at current line.	Line following last line deleted.
DL /string/ [m,n]	Delete line containing /string/n times; if m then consider /string/ starting at column m only.	Line following last line deleted.
MV L_1, L_2 [, L_3]	Move line(s) from L_2 up to and including L_3, if specified and insert them following L_1.	Last line having line number renumbered.
SA xS_1x [m,n]	Save all lines in EB containing string S_1 starting at Column m, for n occurrances, when specified.	Unchanged
SV xS_1x [m,n]	Save all lines on input file containing string S_1, starting at Column m, for n occurrances, when specified.	First line read

APPENDIX 10

SUMMARY OF LIBRARY LOADER COMMANDS

AMAP lu	display map on logical unit (alphabetic sort)
BIAS xxxx	set bias for program
BC xxxx	set number of bytes of memory (hex) to be reserved for blank common.
BF lu	move backward on LU until filemark found
BR lu	move backward one record on LU
COPY lulu [program / label]	Copy one program from first LU to second LU; named program if specified, otherwise next program.
DUPE lulu [program / label]	Copy all programs from first LU to second LU; up to but not including named program if specified or up to EOF or EOM if not.
EDIT lu [program / label]	edit against all programs on logical unit up to EOF to EOM. Link in any program whose program label matches an outstanding reference in previously loaded programs. If program label is specified search for this program before starting to edit.
END	Terminate library loader.
FF lu	move forward on LU until filemark found
FIND lu program / label	search for named program on logical unit
FR lu	move forward one record on LU
GO	transfer control to the transfer address of the program/s loaded by the library loader.
LABEL lu program / label	output to the logical unit specified, one object code record representing the given program label. (A program without a label can then be copied or assembled and output immediately following on the same LU.)
LC xxxx	set number of bytes of memory (hex) to be reserved for labelled common.
LF lu [program / label]	link into programs loaded so far all programs on logical unit up to EOF or EOM. If program label is specified, search for this program before starting to link in programs.
LINK lu [program / label]	link into programs loaded so far one program from logical unit; named program if specified, otherwise next program.
LOAD lu [program / label]	load one program from logical unit; named program if specified, otherwise next program.
MAP lu	display map on logical unit (address sort)
OUT lu [program / label]	put library loader into output mode, the linked programs to be output as a single object code load module on logical unit. Label output module if program label is given.
PAUSE	Pause Library Loader
REWIND lu	rewind LU
TABLE lulu	Search first LU and list all program labels on second LU.
TOPCOR	Define top of physical memory.
WF lu	write filemark on LU
XOUT	complete output module and terminate output mode

APPENDIX 11

UTILITY COMPATIBILITY

The following utilities must be run with OP COMP:

CAL MACRO
CAL
FORTRAN V
FORTRAN RTL

APPENDIX 12

SYSTEM JOURNAL CODES

JOURNAL ENTRY

TCBNBR	JOURNAL CODE
Register 12	
Register 13	
Register 14	
Register 15	

where: TCBNBR = Task number of currently active task

1 Command Processor
2 Background Task
3 Foreground Task 1
4 Foreground Task 2
etc.

JOURNAL CODES

CODE (HEX)	DESCRIPTION
10	I/O termination
11	Task scheduled
13	Illegal instruction fault
14	Floating point fault
15	Queue termination interrupt
16	Divide fault
17	Memory protect fault
20	Parameter added to task queue
6X	SVC X call

REGISTER DEFINITION

CODE (HEX)	REG 12	REG 13	REG 14	REG 15
10	TCB PNTR of task using DCB	A(DCB)	not defined	not defined
11	not defined	not defined	Task's current PSW	
13	not defined	not defined	Illegal Instruction OLD PSW	
14	not defined	not defined	Floating Point Fault OLD PSW	
15	not defined	not defined	Queue Termination Interrupt OLD PSW	
16	not defined	not defined	Divide Fault OLD PSW	
17	not defined	not defined	Memory Protect Fault OLD PSW	
20	ADTSKQ return adrs.	A(UDL)	A (task queue)	Parameter
6X	1st halfword of parameter block	A(SVC parameter block)	SVC OLD PSW	

APPENDIX 13

BOOT PUNCHER ERROR MESSAGES

FD1-ERR	Illegal syntax on input fd
FD2-ERR	Illegal syntax on output fd
ASGN-ERR xxxx LU=n	Assignment error on LU n
IO-ERR xxxx LU=n	I/O error on LU n
CKSM-ERR	Checksum error in object code
SEQ-ERR	Object code records out of sequence
OBJ-ERR	Illegal object code item
MEM-ERR	Insufficient memory to hold image

(xxxx same as OS mnemonic for type of error)

NOTE:

Legal object code items for the boot puncher are 0,1,3,5,8, A, and F. See the 16-bit Loader Description Manual, Publication Number 29-231, for a description of object code items.

APPENDIX 14

TABLE TO CONVERT USER PROGRAM ADDRESS TO PHYSICAL MEMORY ADDRESS

PSW BITS 8-11	USER ADDRESSES 0000-7FFF	USER ADDRESSES 8000-FFFF
0	0000-7FFF	8000-FFFF
1	0000-7FFF	18000-1FFFF
2	0000-7FFF	28000-2FFFF
3	0000-7FFF	38000-3FFFF
4	0000-7FFF	48000-4FFFF
5	0000-7FFF	58000-5FFFF
6	0000-7FFF	68000-6FFFF
7	8000-FFFF	0000-7FFF
8	8000-FFFF	8000-FFFF
9	8000-FFFF	18000-1FFFF
A	8000-FFFF	28000-2FFFF
B	8000-FFFF	38000-3FFFF
C	8000-FFFF	48000-4FFFF
D	8000-FFFF	58000-5FFFF
E	8000-FFFF	68000-6FFFF
F	0000-7FFF	8000-FFFF

PHYSICAL ADDRESS

INDEX

PUBLICATION COMMENT FORM

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

From _____ Date _____

Title _____ Publication Title _____

Company _____ Publication Number _____

Address _____

_____

_____

FOLD FOLD

Check the appropriate item.

☐ Error Page No. _____ Drawing No. _____

☐ Addition Page No. _____ Drawing No. _____

☐ Other Page No. _____ Drawing No. _____

Explanation:

FOLD FOLD

CUT ALONG LINE

Fold and Staple

No postage necessary if mailed in U.S.A.

STAPLE

STAPLE

FOLD

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

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

POSTAGE WILL BE PAID BY ADDRESSEE

PERKIN-ELMER
Computer Systems Division
2 Crescent Place
Oceanport, NJ 07757

TECH PUBLICATIONS DEPT. MS 322A

FOLD

FOLD

STAPLE

STAPLE