PART VII
XVM/RSX ON-LINE TASK DEVELOPMENT

CHAPTER 1

INTRODUCTION TO MULTIACCESS

1.1 FUNCTION OF TASK DEVELOPMENT

RSX has the capability of multiuser on-line task development with concurrent batch processing. Multiuser task development (TDV) is known as the MULTIACCESS facility. RSX offers a full range of TDV functions under control of the MULTIACCESS Monitor. By calling different TDV system modules (function tasks) supplied with RSX, the user can enter or change source code, compile a FORTRAN IV program or assemble a MACRO program, and build this program as a task that will execute under RSX control. (In addition, the user can invoke utility routines to perform a variety of file listing and maintenance functions.) The TDV function tasks supplied as part of RSX are listed in Table 1-1.

Besides the TDV function tasks, the MULTIACCESS Monitor supports a set of commands that control the user TDV environment. The MULTIACCESS Monitor commands are listed in Table 1-2. Furthermore, three control-character facilities have been provided to help the user control his particular task development. The MULTIACCESS control-character facilities are listed in Table 1-3.

JOB, BAT and END (not listed in Table 1-1) are classified as TDV functions, but because they have meaning only when batch processing, they are separtely described in Part VIII of this manual.

Another TDV function task (not listed in Table 1-1), PUP (Peripheral Utility Program), resembles PIP (Peripheral Interchange Program), which is used on other systems. PUP enables the user to transfer files from one device to another. Although it is not supplied with RSX, the user can obtain it from the DEC User Society (DECUS).

The MULTIACCESS facility can be used from any number of terminals up to the maximum number supported by the particular system configuration. However, for most installations, the number of terminals used simultaneously for task development should be limited to six. This is suggested because of the large size of several critical TDV function tasks.

Table 1-1
TDV Function Tasks

TDV Function Task	Function Performed
FOR[TRAN]	Compile a FORTRAN IV program
MAC[RO]	Assemble a MACRO program
EDI[TOR]	Text-edit source code
SLI[P]	Edit batch files
TKB[UILDER]	Build a task for execution
BTK	Basic Task Builder
FIN[PUT]	Transfer a sequential file to disk
DEC[K]	Batch transfer to disk
FOU[TPUT]	Transfer a sequential file from disk
LIS[T]	Transfer a sequential file from disk to printing device
TYP[E]	Transfer a sequential file from disk to printing device
DEL[ETE]	Delete file from disk
REN[AME]	Rename file stored on disk
DIR[ECTORY LIST]	List files in file directory on disk
DTD[IRECTORY]	List DECtape file directory
NEW[DIRECTORY]	Write out a new DECtape file directory
INS[TALL]*	Install a task in the system
REQ[UEST]*	Request task execution
REM[OVE]*	Remove a task from the system
MNT*	Logically mount a disk
DSM*	Logically dismount a disk
CON[STRUCT]	Store a task on a user disk
STA[TUS]	Print MULTIACCESS Monitor statictics
QUE[UE]	Queue a batch job
ODT	Octal Debugging Technique
ACI	Initialize batch account file
ACD	Display batch account file

*Not recommended for use under MULTIACCESS

Table 1-2
MULTIACCESS Monitor Commands

Command	Function Performed
OFF	Log the user off of the MULTIACCESS system
DEV[ICES]	Print the user's LUN-to-device relationship
PAR[TITION]	Specify the user's memory requirements
KPAR[TITION]	Specify the user's memory requirements
ASG (ASSIGN)	Assign the user's LUNs to peripheral devices
XQT	Execute a user task

Table 1-3
MULTIACCESS Control-Character Facilities

Control Character	Function Performed
CTRL/T	Begin the login process, if the user is not logged in. Inform the user of his current job status, if the user is logged in.
CTRL/Y	Abort the user's current task (TDV function or user-written task), if one is active.
CTRL/P	Resume a user task if it has been suspended by a FORTRAN PAUSE statement or MACRO SUSPEND directive.

All TDV function tasks are invoked through the MULTIACCESS Monitor, an exec-mode task that is built for any user-specified partition within the first 32K of core. TDV function tasks, such as the FORTRAN IV Compiler, reside on the disk until requested by the Resident TDV Dispatcher. These tasks typically share a core partition, although they can be built to run in any partition.

To facilitate concurrent operations, it is necessary to assign virtual user LUNs in such a way that devices need not be reassigned when the user switches from one TDV function to another. Table 1-4 contains recommended virtual LUN assignments for TDV use. Device names correspond to devices listed in Table 1-5. Function names are given in Table 1-1.

NOTE

Thoughout this part of the manual and wherever user LUNs are referred to, it must be understood that these are virtual LUNs. Virtual LUNs are automatically "mapped" to system LUNs by the MULTIACCESS Monitor.

The MULTIACCESS Monitor accepts requests for specific functions as input from a dynamically allocated system LUN. The resident task then requests that a specific TDV function task be brought into its core partition from disk. This function task assumes control and can perform the jobs described in subsequent chapters.

TDV function tasks accept command input from LUN-12, in much the same way as the Resident MCR accepts requests for MCR functions from LUN-2. TDV error messages are sent to LUN-13 just as MCR error messages are sent to LUN-3. The MULTIACCESS user should leave virtual LUNs 12 and 13 assigned to TT (the default assignment).

Actual time of execution of TDV function tasks, like user tasks, depends on partition availability and task priority. More detailed descriptions of this process appear below and in Part XI of this manual.

Table 1-4
TDV Virtual LUN Assignments

LUN	1	5	10	11	12	13	14	15	16	17	18	19	20	21
Recommended Device Assignments	DK	RF, RP or RK	RF, RP or RK	RF, RP or RK	TT	TT		RF, RP or RK	LP	RF, RP or RK	RF, RP or RK	DT	TT	TT
LUN Use					TDV cmd in	TDV err log								
					FOR cmd in	FOR err msgs		FOR SRC in	FOR list	FOR BIN out				
					MAC cmd in	MAC err msgs		MAC SRC in	MAC list	MAC BIN out	MAC sec in		MAC aux in	
					EDI cmd in	EDI err msgs			EDI aux out	EDI SRC I/O	EDI temp file		EDI aux in	EDI term I/O
			TKB sys libr	TKB user libr	TKB cmd in	TKB term out				TKB BIN in	TKB TSK out			
					FIN cmd in	FIN err msgs				FIN file out		FIN file in		
					FOU cmd in	FOU err msgs				FOU file in		FOU file out		
					LIS cmd in	LIS err msgs			LIS list	LIS file in				
					DEL cmd in	DEL err msgs				DEL file I/O				
					REN cmd in	REN err msgs				REN file I/O				
	DIR dsk drvr				DIR cmd in	DIR err msgs			DIR list					

(Continued)

Table 1-4 (Cont.)
TDV Virtual LUN Assignments

LUN	1	5	10	11	12	13	14	15	16	17	18	19	20	21
					DTD cmd in	DTD err, list						DTD tape in		
					NEW cmd in	NEW err msgs						NEW tape out		
		INS TSK in			INS cmd in	INS err msgs								
					REQ cmd in	REQ err msgs								
					REM cmd in	REM err msgs								
					MNT cmd in	MNT err msgs								
					DSM cmd in	DSM err msgs								
					XQT cmd in	XQT err msgs								
		CON TSK in			CON cmd in	CON err msgs								
					DFC cmd in	DEC err msgs	DFC file in			DEC file out				

Table 1-5
I/O Devices

Device Name	Device
TTn	Terminal
DTn	DECtape
MTn	Magtape
RF	Fixed-Head Disk
RPn	Disk Pack
RKn	Disk Cartridge
PR	Paper Tape Reader
PP	Paper Tape Punch
CD	Card Reader
CP	Card Punch
LP	Line Printer
AD	Analog-to-Digital Converter
AF	Automatic Flying Capacitor Scanner
UD	Universal Digital Controller
CC	System COMMON Communicator
VTn	Display
VWn	Writing Tablet
XY	XY Plotter
CD	Card Reader (CR11)

1.2 REQUESTING THE RESIDENT TDV DISPATCHER

Task development can be requested from any idle terminal by typing CTRL/T. This usually causes the LOGIN Processor to be invoked. CTRL/T is ignored, however, if the terminal is ATTACHed or if the Resident TDV Dispatcher cannot execute for any reason (e.g., if it has not been INSTALLed, if it is DISABLEd or if its partition is in use).

If the user is already logged into MULTIACCESS, CTRL/T causes the status of the current function to be printed. The format of this status message is:

```
TASKNME PARTITION ATL STATUS ABBREVIATION
```

where TASKNAME is the name of the user program or system function, PARTITION is the name of the partition in which the task is running and ATL STATUS ABBREVIATION is one of the following:

ATL Status Abbreviation	Meaning
CREATING TASK	Task is not yet active
WAITING FOR PARTITION	
ON DISK	ATL status 1 or 2
WAITING AT (address)	ATL status 3 (restart address is printed after the abbreviation)
EXECUTING	ATL status 4,5,7 or 10
SUSPENDED	ATL status 6 (equivalent to PAUSE in FORTRAN)
ABORTING SYSTEM ERROR- TASK NOT IN ATL	

If no user-specified task is running (i.e., if the Resident TDV Dispatcher is waiting for the user to enter a command), the following message is printed:

```
TDV AWAITING COMMAND
TDV>
```

If the user is not logged into MULTIACCESS, the LOGIN Processor determines whether there are sufficient resources available to log the user into the system. If resources are not available, the LOGIN Processor prints the message:

```
MULTIACCESS - TOO MANY JOBS
```

on the terminal and ignores any user-typed commands until CTRL/T is typed again. If resources are available, the LOGIN Processor logs in the user and prints the message:

```
XVM/RSX V1B000 MULTIACCESS

MONTH/DAY/YEAR   HOUR:MINUTE
```

```
NN USERS LOGGED IN

SPECIFY DISK TYPE (RK, RP OR RF), UNIT AND UFD>
```

The user must then respond by typing the requested data in the following format:

$$\left[\begin{Bmatrix}RD\\RP\\RF\end{Bmatrix}\ \begin{Bmatrix}0-7\end{Bmatrix}\ \text{␣}\ \text{<UFD>}\right]$$ (e.g., RK5 <XYZ>)

The default disk type is the same as that of the system disk, the default unit number is 0 and the default UFD is <SCR>.

If the user does not complete the response with a line terminator within 60 seconds, the LOGIN Processor returns the terminal to the idle state. If the UFD specified does not exist for the disk specified, the LOGIN Processor creates it. In this event, the LOGIN Processor informs the user by printing the message:

```
UFD CREATED:  <UFD>
```

During the login process, user errors cause the following messages to be printed:

Error Message	Cause
NONEXISTANT UNIT NUMBER RETYPE>	The unit number specified does not exist.
SYNTAX ERROR RETYPE>	The command line contains a syntax error.
UFD CREATION ERROR	A new UFD was specified and disk errors prevented its creation. This is a fatal error.
SYSTEM ERROR- MISSING PDVL NODE	This is a fatal error.

Fatal errors cause an automatic logoff to occur.

Once the user has successfully logged into MULTIACCESS, the system responds by printing the prompter:

```
TDV>
```

In response to this prompter, the user can enter any legal TDV command.

1.3 TDV/SYSTEM COMMUNICATION

Consider the following command line as an example of invoking the FORTRAN IV compiler:

```
TDV>FOR BPROG1
```

In RSX, both the command to run the Compiler and the command to the Compiler itself are typed on the same line.

When the Compiler is loaded into core and begins to run, it issues a system directive called XFRCMD, which causes the line read by the Resident TDV Dispatcher to be transferred to the Compiler. This is a command used by all TDV function tasks regardless of whether or not they request any command input. Until XFRCMD is issued, the Resident TDV Dispatcher command line buffer is occupied.

Once the command line is transferred and the requested task has completed its operation, the Resident TDV Dispatcher task is automatically requested by the TDV function task (the FORTRAN Compiler in this example). Alternatively, the user can abort the current user task and invoke the Resident TDV Dispatcher by typing CTRL/Y.

1.4 TDV COMMAND CONVENTIONS

Requests for individual TDV function tasks are entered according to the following conventions:

1. Command strings are terminated by either a carriage return or an altmode. These terminators are treated identically by MULTIACCESS.

2. Each element of the command string must be separated by either a comma, a space or a back arrow, as appropriate.

3. Any number of characters (except a comma or space) can be inserted between a TDV function task name and its arguments or command-string terminator (carriage return or altmode). This facility is useful for improving the readability of teleprinter copy. For example, the following two commands illustrate the abbreviated and the more readable way of calling the Editor:

```
TDV>EDI

   or

TDV>EDITOR
```

4. If the user discovers an error in the command string before the terminator has been typed, the line can be deleted as far back as the prompting character (>) by typing a single CTRL/U. An "at" symbol (@) is echoed, informing the user that he can now retype the command string. Rubout, echoed as a backslash (\), can be used to delete characters one by one, starting with the last character typed.

1.5 TDV FUNCTION TASKS

Each of the following chapters describes one TDV function task. In all models and examples included in these chapters, the following conventions apply:

1. A space in the text indicates an actual space in the command line.

2. ← indicates an actual backward arrow in the command line.

3. ∇ indicates a terminator. This can be either a carriage return or an altmode.

4. Square brackets ([,...]) indicate optional characters.

5. Optional items that can be repeated are indicated by dots following the item in square brackets. If a comma precedes these dots ([...]), only the last parameter can be repeated. Repetitions must be separated by commas. Therefore, in:

   ```
   FIN[9] [option]name [ext] [,...]
   ```

   at least one name with an optional extension must be specified. Additional file names, each with an optional extension, can be included, as in:

   ```
   TDV>FIN F1,F2,F3

   TDV>FIN F1 SRC,F2 BIN

   TDV>FIN F1,F2 SRC
   ```

6. In "form" models, upper-case characters (except LUN) indicate those required by the system. Lower-case characters indicate entries that are to be specified by the user and are dependent on his particular task.

7. All LUNs requested are virtual LUNs.

8. Braces ({}) enclosing a stack of items indicate that one item in the stack must be selected.

Five TDV functions described in the following chapters are not recommended for use under MULTACCESS. They are:

INSTALL
REQUEST
REMOVE
MOUNT
DISMOUNT

They can be included in the system (via atypical system-build procedures), but only to maintain compatibility with existing user batch streams.

FOR

CHAPTER 2

FORTRAN: COMPILING A FORTRAN PROGRAM

The FORTRAN TDV Function Task invokes the FORTRAN IV Compiler. This Compiler is a two-pass system program that produces relocatable object code. The Task Builder (TKB) then links this code with required Object-Time System (OTS) library routines and optionally with additional user-specified FORTRAN-compiled or MACRO-assembled routines.

Systems with a Floating-Point Processor (FPP) have a special version of the FORTRAN IV Compiler and OTS which utilizes hardware instructions rather than software calls. For example, RELEAE, the REAL arithmetic package, is not included in FPP systems since REAL arithmetic expressions are evaluated with the aid of hardware instructions. The FPP FORTRAN System consists of the standard FORTRAN IV Compiler and Object-Time System interfaced (via conditional assembly and additional routines) to the hardware FPP (Floating-Point Processor). The interface applies to single and double precision floating-point arithmetic and extended integer arithmetic (double integers). Single integer arithmetic is still handled, in part, by software. Installations with a Floating-Point Compiler should delete FOR... and rename F4F... as FOR... .

The FORTRAN Compiler can also be invoked in batch mode.

2.1 INVOKING THE FORTRAN IV COMPILER

The user can invoke the FORTRAN IV Compiler by typing a command according to the following format:

Form:	FOR[TRAN][option[,]...]←name[,name...]▽
Where:	option is a one-character symbol specifying a compilation option (see Table 2-1); option characters may be either concatenated or separated by commas name of program to be compiled is a string of one to six .SIXBT characters; at least one name is required, but several programs may be compiled in sequence to produce separate binary files
Example:	Compile program named PROG, creating a binary file (B) and a listing (L): TDV>FOR BL←PROG

Table 2-1 illustrates all compiler options available under RSX. Any number of these may be concatenated in a TDV command string.

Table 2-1
FORTRAN IV Compiler Options

Option	Action	Default
B	Binary output	No binary file
H	Use subroutine .SS to generate addresses of two- and three-dimensional array elements	Use in-line code to generate array element addresses (except for I/O parameter list elements)
L	Source listing	No listing
O	Object listing	No listing
S	Symbol map	No symbol map
R	Print Compiler version number and END PASS1 on output terminal	No printout

2.2 INPUT/OUTPUT

The following LUN assignments should be made before the FORTRAN IV Compiler is invoked under MULTIACCESS:

LUN	Assignment
13	Error messages (terminal recommended)
15	Source file input
16	Listing output
17	Binary file output

If an I/O error occurs during compilation, the message:

```
FOR-I/O ERROR LUN xx yyyyyy
```

is produced on LUN-13. In this message, xx represents the logical unit number (decimal) and yyyyyy represents the event variable value (octal), indicating the cause of the error.

2.3 COMPARISON WITH OTHER VERSIONS OF FORTRAN

The FORTRAN IV language employed under XVM/RSX is basically the same as that used under the XVM/DOS system. It differs, however, from the versions of the Compiler used in support of previous versions of RSX. All user FORTRAN programs developed under previous versions of RSX must be recompiled and rebuilt by the Task Builder to enable them to run under XVM/RSX.

The basic language and operating conventions of XVM/RSX FORTRAN are described in the FORTRAN IV XVM Language Manual and FORTRAN IV XVM Operating Environment Manual. Nevertheless, RSX user interaction with FORTRAN IV differs in some ways from the descriptions in those manuals. The main differences occur in support of the auxiliary disk, input/output, the STOP and PAUSE statements, the floating-point processor, OTS output and table initialization. These differences are described in more detail below. Differences in methods for opening, closing, deleting and renaming files are described in Part VI of this manual.

2.3.1 Auxiliary Disk

FORTRAN IV under RSX supports all standard FORTRAN features, except certain magtape and auxiliary disk I/O functions (e.g., REWIND, BACKSPACE, etc.).

2.3.2 Input/Output

The Input Command Decoder of the Compiler has been modified to use standard task development function input and to compile multiple files with the same switches. For example:

```
FOR BLFILE1,FILE2
```

causes the FORTRAN IV Compiler to compile programs FILE1 and FILE2 in sequence and to produce separate binary files and listings.

More specifically, RSX random-access I/O to the disk differs slightly from the description in the manuals listed above. The description of disk I/O in Part VI of this manual gives details on random-access I/O.

2.3.3 STOP and PAUSE Statements

FORTRAN IV STOP and PAUSE statements call library routines that have been slightly modified for RSX use. A STOP message prints on the terminal only if nonzero arguments are present; otherwise, the task simply exits. PAUSE produces a new message that includes the task name. To continue after a PAUSE, the user can type a CTRL/P or the operator can use the RESUME MCR Function Task.

2.3.4 Floating-Point Processor

Under RSX, the JEA register for the floating-point processor (FPP) is initialized by the SETJEA system directive. If the system has not been configured for FPP hardware, an unrecoverable .OTS 52 error is output.

2.3.5 OTS Output

Output from the FORTRAN object-time system, such as .OTS errors and STOP and PAUSE messages, goes to LUN-4, typically assigned to the user TDV terminal, except when the system is batch processing. When FORTRAN jobs are run under MCR, LUN-4 is typically assigned to the MCR terminal.

2.3.6 Table Initialization

Under RSX, the I/O Table Initialization Routine (.FP) has been removed from FIOPS and now exists as a separate routine. This implementation prevents unnecessary loading of FIOPS when no I/O capability is required.

2.4 ERROR MESSAGES

Tables 2-2, 2-3, and 2-4 list possible error messages and their implications. All messages in Table 2-2 are printed in the following form:

Form:	>mA<
Where:	m is the error number A is the alphabetic mnemonic characterizing the error class

Table 2-2
FORTRAN Error Messages

Number	Letter	Meaning
		Common, equivalence, data errors:
01	C	No open parenthesis after variable name in DIMENSION statement
02	C	No slash after COMMON block name
03	C	COMMON Block name previously defined
04	C	Variable appears twice in COMMON
05	C	EQUIVALENCE list does not begin with open parenthesis
06	C	Only one variable in EQUIVALENCE class
07	C	EQUIVALENCE distorts COMMON
08	C	EQUIVALENCE extends COMMON down
09	C	Inconsistent equivalencing
10	C	EQUIVALENCE extends COMMON down
11	C	Illegal delimiter in EQUIVALENCE list
12	C	Non-COMMON variables in BLOCK DATA
15	C	Illegal repeat factor in DATA statement
16	C	DATA statement stores in COMMON in non-BLOCK DATA statement or in non-COMMON in BLOCK DATA statement

(Continued on next page)

Table 2-2 (Cont.)
FORTRAN Error Messages

Number	Letter	Meaning
		DO errors:
01	D	Statement with unparenthesized = sign and comma not a DO statement
04	D	DO variable not followed by = sign
05	D	DO variable not integer
06	D	Initial value of DO variable not followed by comma
07	D	Improper delimiter in DO statement
09	D	Illegal terminating statement for DO loop
		External symbol and entry-point errors:
01	E	Variable in EXTERNAL statement not simple non-COMMON variable or simple dummy variable
02	E	ENTRY name non-unique
03	E	ENTRY statement in main program
04	E	No = sign following argument list in arithmetic statement function
05	E	No argument list in FUNCTION subprogram
06	E	Subroutine list in CALL statement already defined as variable
08	E	Function or array name used in expression without open parenthesis
09	E	Function or array name used in expression without open parenthesis
		Format errors:
01	F	Bad delimiter after FORMAT number in I/O statement
02	F	Missing field width, illegal character or unwanted repeat factor

(Continued on next page)

Table 2-2 (Cont.)
FORTRAN Error Messages

Number	Letter	Meaning
03	F	Field width is 0
04	F	Period expected, not found
05	F	Period found, not expected
06	F	Decimal length missing (no "d" in "Fw.d")
07	F	Missing left parenthesis
08	F	Minus without number
09	F	No P after negative number
10	F	No number before P
12	F	No number or 0 before H
13	F	No number or 0 before X
15	F	Too many left parentheses
		Hollerith errors:
02	H	More than two characters in Integer or Logical Hollerith constant
03	H	Number preceding H not between 1 and 5
04	H	Carriage return inside Hollerith field
05	H	Number preceding H not an integer
06	H	More than five characters inside quotes
07	H	Carriage return inside quotes
		Various illegal errors:
01	I	Unidentifiable statement
02	I	Misspelled statement
03	I	Statement out of order
04	I	Executable statement in BLOCK DATA subroutine

(Continued on next page)

Table 2-2 (Cont.)
FORTRAN Error Messages

Number	Letter	Meaning
05	I	Illegal character in I/O statement, following unit number
06	I	Illegal delimiter in ASSIGN statement
07	I	Illegal delimiter in ASSIGN statement
08	I	Illegal type in IMPLICIT statement
09	I	Logical IF as target of logical IF
10	I	RETURN statement in main program
11	I	Semicolon in COMMON statement outside of BLOCK DATA
12	I	Illegal delimiter in IMPLICIT statement
13	I	Misspelled REAL or READ statement
14	I	Misspelled END or ENDFILE statement
15	I	Misspelled ENDFILE statement
16	I	Statement function out of order or undimensioned array
17	I	Typed FUNCTION statement out of order
18	I	Illegal character in context
19	I	Illegal logical or relational operator
20	I	Illegal letter in IMPLICIT statement
21	I	Illegal letter range in IMPLICIT statement
22	I	Illegal delimiter in letter section of IMPLICIT statement
23	I	Illegal character in context
24	I	Illegal comma in GOTO statement
26	I	Illegal variable used in multiple RETURN statement

(Continued on next page)

Table 2-2 (Cont.)
FORTRAN Error Messages

Number	Letter	Meaning
		Pushdown list errors:
01	L	DO nesting too deep
02	L	Illegal DO nesting
03	L	Subscript/function nesting too deep
04	L	Incomplete DO loop-caused by backwards DO loop, error in DO loop foot statement, or error in I/O statement with implied DO loop
		Overflow errors:
01	M	EQUIVALENCE class list full
02	M	Program size exceeds 8K
03	M	Local array length larger than 8K
04	M	Element position in local array larger than 8K or in common array larger than 32K (EQUIVALENCE, DATA)
06	M	Integer negative or larger than 131071
07	M	Exponent of floating point number larger than 76
08	M	Overflow accumulating constant - too many digits
09	M	Overflow accumulating constant - too many digits
10	M	Overflow accumulating constant - too many digits
		Statement number errors:
01	N	Multiply defined statement number or compiler error
02	N	Statement erroneously labeled
03	N	Undefined statement number

(Continued on next page)

Table 2-2 (Cont.)
FORTRAN Error Messages

Number	Letter	Meaning
04	N	FORMAT statement without statement number
05	N	Statement number expected, not found
07	N	Statement number more than five digits
08	N	Illegal statement number
09	N	Invalid statement label or continuation
		Partword errors:
01	P	Expected colon, found none
02	P	Expected close bracket, found none
03	P	Last bit number larger than 35
04	P	First bit number larger than last bit number
05	P	First and last bit numbers not simple integer constants
		Subscripting errors:
01	S	Illegal subscript delimiter in specification statements
02	S	More than three subscripts specified
03	S	Illegal delimiter in subroutine argument list
04	S	Non-integer subscript
05	S	Non-scalar subscript
06	S	Integer scalar expected, not found
10	S	Two operators in a row
11	S	Close parenthesis following an operator
12	S	Adjustable dimension not in dummy array

(Continued on next page)

Table 2-2 (Cont.)
FORTRAN Error Messages

Number	Letter	Meaning
13	S	Adjustable dimension not a dummy integer
14	S	Two arguments in a row
15	S	Digit or letter encountered after argument conversion
16	S	Number of subscripts stated not equal to number declared
		Table overflow errors:
01	T	Arithmetic statement, computed GOTO list, or DATA statement list too large
02	T	Too many dummy variables in arithmetic statement function
03	T	Symbol and constant tables overlap
		Variable errors:
01	V	Two modes specified for same variable name
02	V	Variable expected, not found
03	V	Constant expected, not found
03	V	Array defined twice
05	V	Error: variable is EXTERNAL or argument (EQUIVALENCE, DATA)
07	V	More than one dimension indicated for scalar variable
08	V	First character after READ or WRITE not open parenthesis in I/O statement
09	V	Illegal constant in DATA statement
11	V	Variables outnumber constants in DATA statement
12	V	Constants outnumber variables in DATA statement

(Continued on next page)

Table 2-2 (Cont.)
FORTRAN Error Messages

Number	Letter	Meaning
14	V	Illegal dummy variable (previously used as non-dummy variable)
16	V	Logical operator has non-integer, non-logical arguments
17	V	Illegal mixed mode expression
19	V	Logical operator has non-integer, non-logical arguments
21	V	Signed variable left of = sign
22	V	Illegal combination for exponentiation
25	V	.NOT. operator has non-integer, non-logical argument
27	V	Function in specification statement
28	V	Two exponents in one constant
29	V	Illegal redefinition of a scalar as a function
30	V	No number after E or D in a constant
32	V	Non-integer record number in random-access I/O
35	V	Illegal delimiter in I/O statement
36	V	Illegal syntax in READ, WRITE, ENCODE, or DECODE statement
37	V	END or ERR exists out of order in I/O statement
38	V	Constant and variable modes don't match in DATA statement
39	V	ENCODE or DECODE not followed by open parenthesis
40	V	Illegal delimiter in ENCODE/DECODE statement

(Continued on next page)

Table 2-2 (Cont.)
FORTRAN Error Messages

Number	Letter	Meaning
41	V	Array expected as first argument of ENCODE/DECODE statement
42	V	Illegal delimiter in ENCODE/DECODE statement
		Expression errors:
01	X	Carriage return expected, not found
02	X	Binary WRITE statement with no I/O list
03	X	Illegal element in I/O list
04	X	Illegal statement number list in computed or assigned GOTO
05	X	Illegal delimiter in computed GOTO
07	X	Illegal computed GOTO statement
10	X	Illegal delimiter in DATA statement
11	X	No close parenthesis in IF statement
12	X	Illegal delimiter in arithmetic IF statement
13	X	Illegal delimiter in arithmetic IF statement
14	X	Expression on left of = sign in arithmetic statement
15	X	Unequal number of right and left parentheses
16	X	Illegal open parenthesis (in specification statements)
17	X	Illegal open parenthesis
19	X	Unequal number of right and left parentheses
20	X	Illegal alphabetic in numeric constant
21	X	Symbol contains more than six characters

(Continued on next page)

Table 2-2 (Cont.)
FORTRAN Error Messages

Number	Letter	Meaning
22	X	.TRUE., .FALSE., or .NOT. preceded by an argument
23	X	Unparenthesized comma in arithmetic expression
24	X	Unary minus in I/O list
26	X	Illegal delimiter in I/O list
27	X	Unterminated implied-DO loop in I/O list
28	X	Illegal = sign in I/O list
29	X	Illegal partword operator
30	X	Illegal arithmetic expression
31	X	Illegal operator sequence
32	X	Illegal use of = sign
33	X	Missing parenthesis in I/O statement with implied DO loop will also cause >041<
34	X	Extraneous characters at the end of assignment statement

In Table 2-3, (R) indicates a recoverable error and (T) indicates a terminal error.

Table 2-3
OTS Error Messages

Number	Possible Source	Meaning
05 (R)	SQRT	Negative REAL square-root argument
06 (R)	DSQRT	Negative DOUBLE PRECISION square-root argument
07 (R)	.GO	Illegal index in computed GOTO
10 (T)	.FR, .FW, .FS, .FX, DEFINE, RANCOM	Illegal I/O device number
11 (T)	.FR, .FA, .FE, .FF, .FS, RANCOM, RBINIO, RBCDIO	Bad input data - IOPS mode incorrect
12 (T)	.FA, .FE, .FF	Bad FORMAT
13 (T)	.BC, .BE, ALOG	Negative or zero REAL logarithmic argument (terminal)
14 (R)	.BD, .BF, .BG, .BH, DLOG, DLOG10	Negative or zero DOUBLE PRECISION logarithmic argument
15 (R)	.BB, .BC, .BD, .BE, .BF, .BG, .BH	Zero raised to a zero or negative power (zero result is passed)
16 (R)	ATAN2	ATAN2 (0.0,0.0) attempted Pl/2 returned
17 (R)	DATAN2	DATAN2 (0.0D0, 0.0D0) attempted Pl/2 returned
20 (T)	FIOPS, AUXIO, RANCOM	Fatal I/O error (RSX only). The message format is "OTS-20-XXX-NNNNNN-TSKNM", where XXX is the LUN, NNNNNN is the event variable (two's complement) and TSKNM is the task name.
	21 to 26 are direct-access errors:	
21 (T)	RANCOM	Undefined file
22 (T)	DEFINE	Illegal record size
23 (T)	RANCOM	Size discrepancy
24 (T)	DEFINE, RANCOM	Too many records per file or illegal record number
25 (T)	RANCOM	Mode discrepancy

(Continued)

Table 2-3 (Cont.)
OTS Error Messages

Number	Possible Source	Meaning
26 (T)	DEFINE	Too many open files
30 (R)	RELEAE, .FPP	Single integer overflow (detected only when fixing a floating-point number)
31 (R)	DBLINT, JFIX, JDFIX, ISNGL	Extended (double) integer overflow; also prints PC with FPP system (with non-floating-point processor system, detected only when fixing a floating-point number)
32 (R)	RELEAE	Single floating-point overflow; also prints PC with FPP system
33 (R)		Double floating-point overflow; also prints PC with FPP system (detected only by FPP system, not by software system)
34 (R)	RELEAE	Single floating-point underflow; also prints PC with FPP system
35 (R)		Double floating-point underflow; also prints PC with FPP system (detected only by FPP system, not by software system)
36 (R)	RELEAE	Floating-point divide check; also prints PC with FPP system
37 (R)	INTEAE	Integer divide check; if extended integer divide check, prints PC with FPP system
40 (T)	ENCODE	Illegal number of characters specified (legal: 0<c<625)
41 (R)	ENCODE	Array exceeded
42 (T)	DDIO	Bad input data
43 (T)	DA	Attempt to pass more arguments than expected
50 (T)		FPP memory-protect/nonexistent memory; also prints PC with FPP system
51 (T)	BCDIO, BINIO	READ to WRITE illegal I/O direction change to disk without intervening CLOSE or REWIND

In software systems, arithmetic errors resulting in the OTS error messages summarized above are detected in the arithmetic package (RELEAE and INTEAE). In the hardware FPP systems, these errors are detected by the hardware (with the exception of single integer divide check) and serviced by a trap routine in the FPP routine .FPP.

Where applicable, on such error conditions, the result is patched for both software and hardware systems as summarized in the following table.

Table 2-4
OTS Error Messages in FPP Systems

Error	Patched Value	
	FPP Hardware System	Software System
Single Floating Overflow (.OTS 32)	± largest single floating value	same
Double Floating Overflow (.OTS 33)	± largest double floating value	not detected
Single Floating Underflow (.OTS 34)	zero	same
Double Floating Underflow (.OTS 35)	zero	not detected
Floating Divide Check (.OTS 36)	± largest single floating value	same
Integer Overflow (.OTS 30)	limited detection	same
Double Integer detection Overflow (.OTS 31)	none	limited
Integer Divide Check (.OTS 37)	none	same

"Limited detection" means that while a floating-point number is being fixed, integer overflow and extended integer overflow are detected. In these instances, plus or minus the largest integer for the data mode is patched as result.

For Double Integer Overflow, "none" means that with the FPP system all extended integer overflow conditions are detected, but the results are meaningless. Elsewhere, "none" means the result is meaningless.

Further, when converting an extended integer, the magnitude of which is >2(17) -1, to a single integer, no error is indicated and the high order digits are lost.

CHAPTER 3

MACRO: ASSEMBLING A MACRO PROGRAM

The MACRO Assembler produces relocatable binary object code that can be loaded for debugging or execution. The Assembler prepares the object program for relocation, and the Task Builder sets up linkages to external subroutines.

The MACRO Assembler processes source programs in either a two-pass or three-pass operation. In the two-pass assembly operation the source program is read twice and the object program (and printed listing when requested) are produced during the second pass. During the first pass (PASS1) the locations to be assigned the program symbols are resolved and a symbol table is constructed by the Assembler. The second pass (PASS2) uses the information computed during PASS1 to produce the final object program.

In an optional three-pass assembly operation, PASS2 calls in a third pass (PASS3) portion of the Assembler program. PASS3, when called, performs a cross-referencing operation during which a listing is produced which contains all user symbols, where each symbol is defined, and the number of each program line in which a symbol is referenced. On completion of its operation, PASS3 calls the PASS1 and PASS2 portions of the Assembler program back into core for further assembly operations.

The MACRO Assembler can also be invoked in batch mode.

3.1 INVOKING THE MACRO ASSEMBLER

The user can invoke the MACRO Assembler by typing a command according to the following format:

Form:	MAC[RO] [option[,]...]←name [ext][,...] ▽
Where:	option is a one-character symbol specifying an assembler option (see Table 3-1); option characters may be either concatenated or separated by commas name of program to be assembled is a string of one to six .SIXBT characters; one or more names may be required, depending on whether or not certain assembly options (e.g., FZ) have been requested; several programs may be specified so that they are assembled in sequence to produce separate binary files ext is a string of one to three .SIXBT characters representing the extension. SRC is the default
Example:	Assemble program named TMAC, creating a binary file (B) and a listing (L): TDV>MAC BL←TMAC Assemble programs P1 and P2 to produce a separate binary file (B) for each; both require the use of a parameter file (P): TDV>MAC PB←PARAM,P1,PARAM,P2

Table 3-1 illustrates all Assembler options. Any number of these options, in any order and optionally separated by commas, may be included in a TDV command string.

Table 3-1
Assembler Options

Option	Action	Default
B	Generate a binary file	No binary file
L	Generate a listing file on the requested output device	No listing file (see options N, C, G)
N	Number each source line (decimal); it is not necessary to type the L option	Do not number source lines
C	Do not print program areas that fall between unsatisfied conditionals; it is not necessary to type the L option	Print all source lines

(Continued on next page)

Table 3-1 (Cont.)
Assembler Options

Option	Action	Default
G	Print only the source line of a MACRO expansion; it is not necessary to type the L option	Generate printouts for MACRO expansions and expandable pseudo-ops (e.g., .REPT)
P	Before assembly begins, read program parameters from LUN-20; (the code read from LUN-20 is read only once; for this reason only direct assignments should be used)	No parameters; begin assembly immediately after command string termination
A	Print symbols at end of PASS2 in alphanumeric sequence	Do not print symbols in alphanumeric sequence (see options V, S, and H)
V	Print symbols at end of PASS2 in value sequence	Do not print symbols in value sequence (see options A, S, and H)
S	Same as selecting both A and V above	Do not print symbols unless option A, option V, or both are requested (see option H)
H	If A, V, or S option is used, print symbols one to a line	If A, V, or S option is used, print symbols horizontally, four to a line
X	At completion of PASS2, load PASS3 to perform the cross-referencing operation; at completion of PASS3, the Assembler calls in PASS1 and PASS2 to continue assembling programs; if the command string is terminated by an ALTMODE, control returns to RSX at the end of assembly	Do not provide a cross-reference and do not call in PASS3
O	Do not output the source extension and the linking loader code 33 as a special code in the binary file; this option must be used when assembling programs in RSX to run in Background/Foreground	Assemble as usual (use of unique extensions permits easy identification of different versions of a program)

(Continued on next page)

Table 3-1 (Cont.)
Assembler Options

Option	Action	Default
E	Print any errors occurring during assembly on the console printer (LUN-13) in addition to the device assigned to LUN-16; the L or N option should be used with the I option; this option is particularly useful to users who assign nonprinting devices to LUN-16	Output errors only to the listing device (LUN-16)
T	Generate a Table of Contents during PASS1; the table will contain the page number and text of all assembled .TITLE statements in the program	No table of contents
R	Identify assembler version number; print END PASS1, END PASS2, and error count on output terminal	Do not print version number, END PASS1, END PASS2, or error count
I	Ignore .EJECTs; treat the .EJECT pseudo-op as a comment	Skip to head of form when .EJECT is encountered
F	Assembly includes an additional file: a macro definition file, a second parameter file, or a second part of the program file; this additional file is specified in the command string and it is read via LUN-18; (see option Z)	No additional file
Z	Assemble the file associated with the F option not only during PASS1 but also PASS2; this allows assembly of a two-part source file on two different devices or on the same device; this option takes effect only if the F option is also specified	F-option file is referenced only during PASS1 and therefore may contain only direct assignment statements and comments

If the L and X options are entered in the same command string, MACRO assumes that the N option is also entered. Without the N option, the user would obtain a cross-reference that would be virtually useless, because the source lines of the listing would not be numbered.

3.2 INPUT/OUTPUT

The following LUN assignments should be made before the MACRO Assembler is invoked under MULTIACCESS:

LUN	Assignment
13	Error messages (terminal recommended)
15	Source file input
16	Listing output
17	Binary file output
18	Secondary source file input
20	Auxiliary input (terminal recommended)

A secondary input device is needed to assemble a program from two different sources. An auxiliary input device is needed to supply parameters for a file that is to be assembled.

If an I/O error occurs during assembly, the message:

```
MAC-I/O ERROR LUN xx yyyyyy
```

is produced on LUN-13. In the message, xx represents the logical unit number (decimal) and yyyyyy represents the event variable value (octal), indicating the cause of the error.

3.3 COMPARISON WITH OTHER VERSIONS OF MACRO

The MACRO Assembler is essentially the same under RSX as it is under the other XVM monitor systems. The basic instructions and operation conventions are described in MACRO XVM ASSEMBLER LANGUAGE MANUAL.

Nevertheless, RSX user interaction with the MACRO Assembler differs in some ways from the descriptions in the manual. The main differences occur in support of dump mode, input/output, parameter input, the definitions of certain standard functions and certain pseudo-ops. These differences are discussed in more detail below.

3.3.1 Dump Mode

MACRO under RSX supports all standard MACRO features except the pseudo-ops .IODEV, .ABS, .ABSP. .FULL, and .FULLP. This is because RSX I/O Device Handler Tasks do not support dump mode (mode 4).

3.3.2 Input/Output

The Assembler's Input Command Decoder has been modified to use standard Task Development function input and to assemble multiple files with the same switches. For example:

```
MAC BNX←FILE1,FILE2
```

causes the MACRO Assembler to assemble programs FILE1 and FILE2 in sequence and to produce separate binary files and listings.

3.3.3 Parameter Input

If the P assembly option is used and input is to come from the terminal, the MACRO Assembler types the following message:

```
MAC-INPUT PARAMETER DEFINITIONS
```

on the output terminal (LUN-20). The user responds by entering input parameters. To terminate input, he types a CTRL/D (↑D) followed by a carriage return.

3.3.4 MACRO Definitions File

The RSX Assembler does not contain within itself the definition of standard, frequently-used System Directive and I/O calls. These exist in a MACRO Definitions File (RMC.nn SRC where nn is the current edit number) which is provided as part of the RSX system. During assembly, this definition file can be accessed via the secondary-input LUN (LUN-18) if assembly option F is specified. (Z is also needed when two passes are required.) This file cannot be read via the auxiliary input LUN (LUN-20) using the P option because many MACROs make forward symbolic references, necessitating a second pass on the file.

3.4 ERROR MESSAGES

The Assembler examines each source statement for possible errors. A statement containing an error is flagged by one or several letters in the left-hand margin of the line, or, if the lines are numbered, between the line number and the location. The following table lists error flags and their meanings.

Table 3-2
MACRO Assembler Error Flags

Flag	Meaning
A	Error in direct symbol table assignment - assignment ignored
B	1. Memory bank error (program segment too large) 2. Page error - the location of an instruction and the address it references are on different memory pages (error in page mode only)
C	A .ENDC or a .CBE is given with no opening conditional. The error appears on the offending line.
D	Statement contains a reference to a multiply-defined symbol - the first value is used
E	1. Symbol not found in user's symbol table during PASS2 2. Operator combined with its operand may produce erroneous results
F	Forward reference - symbol value is not resolved by PASS2
I	Line ignored: 1. Redundant pseudo-op 2. A second .LOCAL pseudo-op appears before a matching .NDLOC pseudo-op 3. An .NDLOC appears without an associated .LOCAL pseudo-op 4. Too many .LTORG pseudo-ops (more than eight)
L	Literal error: 1. Phase error - literal encountered in PASS2 does not equal any literal found in PASS1 2. Nested literal (a literal within a literal)

(Continued on next page)

Table 3-2 (Cont.)
MACRO Assembler Error Flags

Flag	Meaning
M	Multiple symbol definition - first value defined is used
N	Error in number usage (digit 8 or 9 used under .OCT influence)
P	Phase error: 1. PASS1 symbol value not equal to PASS2 symbol value (PASS2 value ignored) 2. A tag defined in a local area (.LOCAL pseudo-op) is also defined in a non-local area
Q	Questionable line: 1. Line contains two or more sequential operators (e.g., LAC A+*B) 2. Bad line delimiter - address field not terminated with a semicolon, carriage return, or comment 3. Bad argument in .REPT pseudo-op
R	Possible relocation error
S	Symbol error - illegal character used in tab field
U	Undefined symbol
W	Line overflow during macro expansion
X	Illegal use of macro name or index register
Y	A .CBS was given with no closing .CBE, a .DEFIN has no corresponding .ENDM and/or a .IF conditional has no closing .ENDC. This error is output on the .END statement at the end of PASS 1.

In addition, certain conditions cause assembly to be terminated prematurely. The following table lists possible error messages and their meanings.

Table 3-3
MACRO Assembler Error Messages

Message	Meaning
SYNTAX ERR	Bad command string; control returns to TDV
NAME ERROR	File named in command string not found; control returns to TDV
TABLE OVERFLOW	Too many symbols and/or MACROs; control returns to TDV
CALL OVERFLOW	Too many embedded MACRO calls; control reutrns to TDV
CORE EXHAUSTED AT LINE nnn	PASS3 error; too many symbol references; control returns to TDV

EDI

CHAPTER 4

EDITOR: TEXT-EDITING SOURCE CODE

The Text Editor is an effective text editing program that allows the modification and creation of symbolic source programs and other ASCII text material. The Editor reads and writes standard IOPS ASCII lines. By means of keyboard commands, the Editor is directed to bring a line or group of lines from an input file to an internal buffer. By means of additional commands, the user can then examine, delete and change the contents of the buffer, and insert new text at any point in the buffer. When the line or block of lines has been edited, it is written into a new file on the output device.

The Editor is most frequently used to modify MACRO and FORTRAN IV source programs, but it can also be used to edit any symbolic text.

The Editor operates in either an "edit" or "input" mode. In the "edit" (or command) mode, the program accepts and acts on control-word and data strings to open and close files; to bring lines of text from an open file into the work area; to change, delete or replace the line currently in the work area; and to insert single or multiple lines after the line in the work area. In the "input" (or text) mode, lines from the keyboard are interpreted as text to be added to the open file. Commands are available for conveniently changing from one operating mode to the other.

Data from the input file are made available for editing in two ways: in line-by-line mode or in block mode. In the line-by-line mode, a single line is the unit of the input file available to the user for modification at any point. In the block mode, a user-specified portion of the input file is held in a core buffer for editing until the user requests that the contents of the buffer be added to the output file. Line-by-line mode is the default data mode.

4.1 INVOKING THE TEXT EDITOR

The user can invoke the Text Editor by typing a command in the format:

Form:	TDV>EDIV

To return to the MULTIACCESS Monitor after editing and closing a file, the user must type:

E[XIT]

An example of interaction with the Text Editor follows (> is a prompter):

```
TDV>EDI                  User invokes Text Editor
EDITOR XVM/RSX V1B000    Editor gives version number
>OPEN FILNAM EXT         Open the file FILNAM, with extension
                         EXT, for editing
EDIT                     Editor indicates EDIT mode
>                        Go to top of file
>T                       Print current line; nothing is printed,
                         because pointer is above first line
                         of file
>N                       Go to next line
>P                       Print current line

     .TITLE FILNAM       /EDIT COMMAND DEMONSTRATOR

>C /TOR/TION/            Change DEMONSTRATOR to DEMONSTRATION

     .TITLE FILNAM       /EDIT COMMAND DEMONSTRATION

>CLOSE                   Close file
>EXIT                    Exit from Text Editor
TDV>
```

4.2 INPUT/OUTPUT

Make the following LUN assignments before the Text Editor is invoked under TDV:

LUN	Assignment
16	Auxiliary output (listing)
17	Source file input/output (disk)
18	Source file scratch input/output (temporary file) (disk)
19	Auxiliary output file (disk)
20	Auxiliary input

If an I/O error occurs during editing, the message:

```
EDI - I/O ERROR LUN xx yyyyyy
```

is sent to LUN-13. In the message, xx represents the logical unit number (decimal) and yyyyyy represents the octal event variable, indicating the cause of the error.

4.3 COMPARISON WITH OTHER VERSIONS OF THE TEXT EDITOR

The Text Editor runs the same way under RSX as it does under the other XVM operating systems. Program commands and conventions are described in the EDIT/EDITVP/EDITVT XVM Utility Manual.

RSX user interaction with the Text Editor differs in some ways from the descriptions in the manual. The main differences occur in disk-to-disk editing, buffer size and partition size. The differences are described in detail in the following sections.

4.3.1 Disk-to-Disk Editing

RSX supports all standard EDIT features, but is restricted to disk-to-disk editing. Therefore, LUN-17 and LUN-18 must be assigned the to disk.

4.3.2 Buffer and Partition Size

On program initialization, the maximum buffer size parameter for the SIZE command is computed. The parameter retains the computed value or is set to 56, whichever is smaller. The SIZE command is then restricted to a value less than or equal to the maximum buffer size.

If the partition size is too small to permit buffers, the message:

```
EDI-PARTITION TOO SMALL
```

is set to LUN-13 and the Text Editor exits.

4.4 ERROR MESSAGES

When a colon follows an error message, the incorrect command line is printed below the error message.

Table 4-1
Editor Error Messages

Message	Meaning
BUFFER NON-EMPTY	BLOCK OFF command issued before Block Buffer emptied by WRITE command
NOT A REQUEST:	Command string error
?	Command string error while BRIEF mode ON
END OF BUFFER REACHED BY:	Command has moved pointer below last line
END OF FILE REACHED BY:	Command has moved pointer below last line
READ ERROR:	Incorrect parity or checksum
TRUNCATED:	Line too long
BUFFER CAPACITY EXCEEDED BY:	Buffer overflowed in Block Mode
NO FILE NAME GIVEN.	File name missing
NO INPUT FILE PRESENT.	Input file missing
FILE name IS PRESENT ON OUTPUT DEVICE. PLEASE RENAME IT OR IT WILL BE DELETED.	More than one file called name
NOTHING IN FILE	No output to file
SIZE TOO SMALL	When Block Mode is ON, number of lines specified in MOVE > size of current block buffer
NOT ENOUGH LINES END OF FILE REACHED BY:	When Block Mode is OFF, number of lines MOVE specifies to be moved > number in current file
NOT ENOUGH BUFFER SPACE	Not enough buffer space to store lines being moved
LINES MOVED TO END	Line containing text string specified in MOVE not found
NO STRING	Convert missing first argument
NOT FOUND	First argument of CONVERT never found

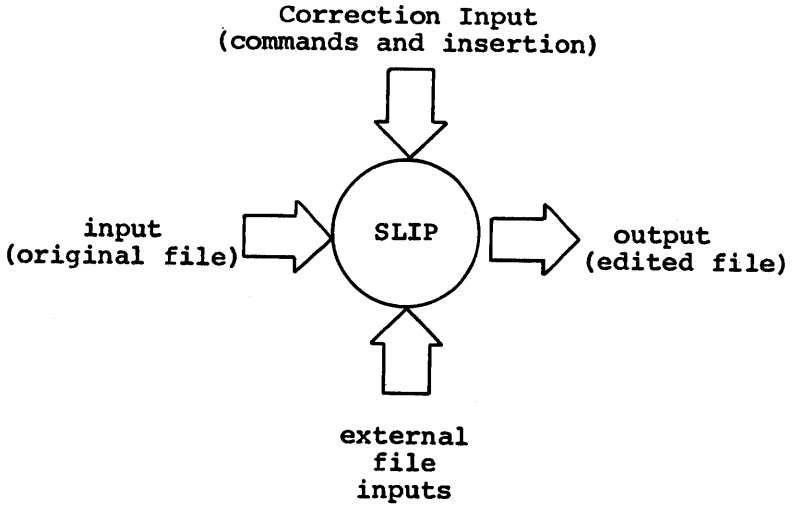
CHAPTER 5

SLIP: EDITING BATCH FILES

The SLIP command invokes a batch-oriented file editor, which not only can edit ASCII source programs and data sets, but can also produce line-numbered listings and perform operations similar to those of PIP.

SLIP complements EDIT, the TDV interactive Text Editor. SLIP is designed particularly for batch operation or for file manipulation, while EDIT is designed for the on-line user who manipulates text within a single file. SLIP works at the record level, while EDIT has extensive intrarecord facilities.

SLIP processes as many as four streams of information in a run, as shown in the following diagram:



This editor SLIPs information from one file to another. It always generates a new file so that the original file can be preserved. Batch operation requires this; otherwise a minor change, incorrectly specified, could destroy an important file. The user can maintain many backup levels.

5.1 INVOKING SLIP

The user can invoke SLIP in one of two ways. From TDV, the command has the following format:

Form:	SLIP option [,...]▽
Where:	option is an option from Table 5-1
Example:	TDV>SLIP N

The user can also invoke SLIP in BATCH mode as part of a job with the following control record:

```
$SLIP options
```

as explained in 5.1.1.

A SLIP run must contain the following four sections:

1. TDV>SLIP options or $SLIP options
2. *FILE specifications
3. SLIP control records and insertion records
4. End-of-file (-$ in DV or $EOF in BATCH or TDV)

5.1.1 $SLIP Options

The $SLIP record has the following format:

Form:	$SLIP option [,...]
Where:	option is an option from Table 5-1
Example:	List output file and suppress trailing spaces of input file: $SLIP L,S

Table 5-1
SLIP Options

Option	Meaning
N[LIST]	No printout except *FILE cards and errors
C[HANGES]	List control records, deleted lines, *FILE cards, and errors (default)
L[IST]	List entire output file, with line numbers, inserted lines, deleted lines, *FILE cards, and errors
K[EEP]	Retain trailing spaces on input records
S[UPPRESS]	Suppress trailing spaces (in multiples of 5) from input record (default)

5.1.2 *FILE Specifications

SLIP uses *FILE, an indirect file selection system. The user must specify exactly one input and one output file as follows:

Form:	*FILE I[NPUT]:LUN='[infile [ext]]'O[UTPUT]: LUN='[outfile [ext]]'$
Where:	LUN is a one- or two-digit Logical Unit Number infile is a string of one to six .SIXBT characters representing the input file and is optional only if the input LUN is non-file-oriented ext is a string of one to three .SIXBT characters representing the file extension outfile is a string of one to six .SIXBT characters representing the output file and is optional only if the output LUN is non-file-oriented
Example:	Input FILE1 RC from LUN-17 to be edited and output on LUN-18: *FILE I:17='FILE1 SRC' :18='FILE1 SRC'$

The infile ext and outfile ext can be identical, but it is better to keep a backup file. If a backup file is desired, the user should enter a sequence such as the following:

```
$DELETE  file bak

$RENAME  file src, file bak

$SLIP  options

*FILE  I:LUN='file bak' O:LUN='file src'
   .
   .
   .

$EOF
```

If the user then makes a bad edit run, she can remove the $DELETE and $RENAME records and rerun the update, because the original file is still intact as "file bak."

Note that certain SLIP commands described below require additional *FILE records.

5.1.3 SLIP Control Records and Insertion Records

SLIP logically numbers all lines within each file it processes so that the first line in each file is number 1. If a SLIP run includes a LIST option, but no control or insertion records, it produces a line printer listing of the file with line numbers as in the example below.

Whether or not the run includes control and insertion records, the listing always shows the final, resultant line numbers, those to be used for the next SLIP performed on that file. Because the renumbering occurs after editing and before output of the listing, the user refers while editing to the original line numbers.

Renumbering does occur, however, when any of the four search commands is used. The line selected by the search becomes line 1. Line references in all subsequent commands are then relative to that line. For instance, after a command causes a search for 'C' in the sample input file below, the command

```
-3,3
```

will delete the line containing 'E', which was originally line 5.

Each search sets the selected line to 1. It is impossible to reference the original line numbers after any search has been conducted.

The file in the example below will be the input file for all subsequent SLIP examples:

```
Example: 1.  A
         2.  B
         3.  C
         4.  D
         5.  E
         6.  F
         7.  G
```

Subsequent sections give details on the nine SLIP control commands, which are:

- INSERT
- INSERT FILE
- REPLACE
- REPLACE WITH FILE
- SEARCH AND INSERT
- SEARCH AND INSERT FILE
- SEARCH AND REPLACE
- SEARCH AND REPLACE WITH FILE
- END-OF-FILE

All control commands must follow the *FILE specification and must begin with a minus sign (-) in column one, followed by one or more parameters. Embedded blanks are allowed within the parameters and are ignored except when between single quotes ('). The commands that permit insertion of an entire file are especially useful for inserting subroutines.

5.1.4 End-of-File

A set of correction cards must be terminated by an end-of-file specification. This is $EOF in BATCH mode or the command -$ in BATCH or TDV.

5.2 CONTROL RECORDS

The following sections summarize SLIP control records and provide examples of usage.

5.2.1 INSERT

The user can specify an INSERT operation by a control record of the following format:

Form:	-n
Where:	n is the number of the line after which the insertion will be made

All records that follow the INSERT control record and precede another control record or an $EOF are inserted into the output file following line n of the input file. For example if the following records

```
-1
X
Y
Z
-$
```

are specified to edit the input file in 5.1.3, the resulting output file is:

```
 1.  A
 2.  X
 3.  Y
 4.  Z
 5.  B
 6.  C
 7.  D
 8.  E
 9.  F
10.  G
```

5.2.2 INSERT FILE

An INSERT FILE operation requires two control records. The first is in the format:

Form:	-n
Where:	n is the number of the line after which the insertion is to be made

The second record is a *FILE record in the format:

Form:	*FILE I[NPUT]:LUN='secfile ext'
Where:	LUN is the logical unit number, which must be different from those used for the input and output files of the edit secfile is a string of one to six .SIXBT characters representing the file to be inserted ext is a string of one to three .SIXBT characters representing the file extension

The entire second file is inserted into the input file after relative line number n. Another control record usually follows the *FILE record. If noncontrol records follow it, they are inserted after the last line of the inserted file and before the next line of the input file.

For example, if the second file, ZZZ RC, consists of:

```
1.  XX
2.  YY
3.  ZZ
4.  KK
```

and is inserted into the input file in 5.1.3 by the commands:

```
-1
*FILE I:15='ZZZ SRC'$
```

the resulting output file is:

```
 1.  A
 2.  XX
 3.  YY
 4.  ZZ
 5.  KK
 6.  B
 7.  C
 8.  D
 9.  E
10.  F
11.  G
```

5.2.3 REPLACE

The user can specify a replacement by a control record of the following format:

Form:	-n,m
Where:	n is the number of the first line to be deleted and replaced m is the number of the last line to be deleted and replaced

All input file records numbered n through m, inclusive, are deleted and all insertion records following the REPLACE control record are inserted in the output file following the (n-1)th record of the input file.

Note that if no records are inserted, lines n through m are deleted. The command

```
-204,99999
```

deletes all lines after 203. The command

```
-204,204
```

deletes only line 204.

The following example illustrates the case in which records are inserted. If the records

```
-2,4
X
Y
Z
-$
```

are specified to edit the input file in 5.1.3, the resulting output file is:

```
1. A
2. X
3. Y
4. Z
5. E
6. F
7. G
```

5.2.4 REPLACE WITH FILE

A REPLACE WITH FILE operation requires two control records. The first is in the following format:

Form:	-n,m*
Where:	n is the number of the first line to be deleted; second file will follow line n-1 m is the number of the last line to be deleted; second file will precede line m+1

The second record is a *FILE record in the following format:

Form:	*FILE I[NPUT]:LUN=*secfile ext*
Where:	LUN is the Logical Unit Number secfile is a string of one to six .SIXBT characters representing the file to replace the deleted lines ext is a string of one to three .SIXBT characters representing the file extension

Lines n through m are deleted and replaced with the entire second file specified. Another control record usually follows the *FILE record. If other insertion records follow it, they are inserted after the last line of the second file.

For example, if the following control records:

```
-3,4*
*FILE I:15='ZZZ SRC'$
-$
```

are specified to delete lines 3 and 4 of the input file in 5.1.3 and to replace them with the second file, ZZZ SRC in 5.2.2,the resulting output file is:

```
1.  A
2.  B
3.  XX
4.  YY
5.  ZZ
6.  KK
7.  E
8.  F
9.  G
```

5.2.5 SEARCH AND INSERT

The user can specify a SEARCH AND INSERT operation by a control record of the following format:

Form:	-'xxxxx'
Where:	xxxxx is a string of one to five characters; the insertion will be made after the line beginning with the string

The search facility is designed to detect FORTRAN line numbers. SLIP examines each line of the input file until it finds one beginning with the string xxxxx. All lines that follow the SEARCH AND INSERT control record and precede another control record or $EOF are inserted into the output file following the line containing xxxxx in the input file.

For example, if the following records

```
-'A'
X
Y
Z
-$
```

are specified to edit the input file in 5.1.3, the resulting output file is:

```
 1.  A
 2.  X
 3.  Y
 4.  Z
 5.  B
 6.  C
 7.  D
 8.  E
 9.  F
10.  G
```

Renumbering occurs when the search is successful. The line selected by the search becomes line 1.

5.2.6 SEARCH AND INSERT FILE

A SEARCH AND INSERT FILE operation requires two control records. The first is in the following format:

Form:	-'xxxxx'*
Where:	xxxxx is a string of one to five characters; the file will be inserted after the line in which the string appears

The second record is a *FILE record in the following format:

Form:	*FILE I[NPUT]:LUN='secfile ext'$
Where:	LUN is the Logical Unit Number secfile is a string of one to six .SIXBT characters representing the file to be inserted ext is a string of one to three .SIXBT characters representing the file extension

SLIP examines each line of the input file until it finds one beginning with the string xxxxx. The entire second file is inserted into the output file after the line containing xxxxx. Another control record usually follows the *FILE record. If other insertion records follow it, they are inserted after the last line of the inserted file.

For example, if the following records

```
-'A'
*FILE I:15='ZZZ SRC'$
-$
```

are specified to insert the second file, ZZZ SRC in 5.2.2, after line 1 of the input file in 5.1.3, the resulting output file is:

```
 1.  A
 2.  XX
 3.  YY
 4.  ZZ
 5.  KK
 6.  B
 7.  C
 8.  D
 9.  E
10.  F
11.  G
```

Renumbering occurs when the search is successful. The line selected by the search becomes line 1.

5.2.7 SEARCH AND REPLACE

The user can specify a SEARCH AND REPLACE operation by a control record of the following format:

Form:	-'xxxxx',m
Where:	xxxxx is a string of one to five characters; the line containing the string is the first to be deleted and replaced m is the number of the last line to be deleted and replaced

SLIP examples each line of the input file until it finds one beginning with the string xxxxx. All input file records from the line containing the string through line m, inclusive, are deleted.

All insertion records following the SEARCH AND REPLACE control record are inserted in the output file following the record just before the record containing xxxxx.

Note that if no records are inserted, the specified lines are deleted. The command

```
-'ABC',99999
```

deletes all lines after the one containing 'ABC'. The command

```
-'ABC',1
```

deletes only that line.

The following example illustrates the case in which records are inserted. If the records

```
-'B',3
X
Y
Z
-$
```

are specified to edit the input file in 5.1.3, the resulting output file is:

```
1.  A
2.  X
3.  Y
4.  Z
5.  E
6.  F
7.  G
```

Renumbering occurs when the search is successful. The line selected by the search becomes line 1.

5.2.8 SEARCH AND REPLACE WITH FILE

A SEARCH AND REPLACE WITH FILE operation requires two control records. The first is in the following format:

Form:	-'xxxxx', m*
Where:	xxxxx is a string of one to five characters; the line containing the string is the first to be deleted; second file will follow the preceding line m is the number of the last line to be deleted; second file will precede line m+1

The second record is a *FILE record in the following format:

Form:	*FILE I[NPUT]:LUN='secfile ext'$
Where:	LUN is the Logical Unit Number secfile is a string of one to six .SIXBT characters representing the file to replace the deleted lines ext is a string of one to three .SIXBT characters representing the file extension

SLIP examines each line of the input file until it finds one beginning with the string xxxxx. All input file records from the line containing the string through line m, inclusive, are deleted and replaced with the file specified. Another control record usually follows the *FILE record. If other insertion records follow it, they are inserted after the last line of the second file.

For example, if the following records:

```
-'C',2*
*FILE I:15='ZZZ SRC'$
-$
```

are specified to delete lines 3 and 4 of the input file in 5.1.3 and to replace them with the second file, ZZZ SRC in 5.2.2, the resulting output file is:

```
1.  A
2.  B
3.  XX
4.  YY
5.  ZZ
6.  KK
7.  E
8.  F
9.  G
```

Renumbering occurs when the search is successful. The line selected by the search becomes line 1.

5.2.9 END-OF-FILE

The user can specify END-OF-FILE by a control record of the following format:

```
-$
```

5.3 INPUT/OUTPUT

The SLIP TDV Function Task is assigned Task name SLI... at Task-Building time. SLIP uses LUN-12 for command input and LUN-16 for listing output. The user may use Logical Units for files as he wishes. Normal editing operations use LUN-17 for input, LUN-18 for output, and LUN-15 for secondary input.

5.4 ERROR MESSAGES

SLIP detects four types of errors:

- *FILE specification errors
- $SLIP control record errors
- SLIP command syntax errors
- SLIP sequencing errors

5.4.1 *FILE Errors

A message of the following form indicates a *FILE error:

```
*FILE ACCESS ERROR xx -- PROGRAM TERMINATED
```

where xx is a code in Table 5-2.

Table 5-2
Error Codes for *FILE Routine

Code	Meaning
1	*FILE does not appear in columns 1-5 of card read
2	File type inputs or outputs not found
3	Colon missing after file type
4	Missing Logical Unit Number (i.e., INPUT:='file')
5	Logical Unit Number does not consist of digits 0-9
6	Logical Unit Number is too long (more than 2 digits)
7	Missing equal sign (=) after Logical Unit Number
8	Missing quote (') after equal sign
9	Invalid delimiter within file name or missing quote (') after file name
10	Illegal file type, not INPUT: or OUTPUT:
11	File not found on input device
12	File already open for output
13	Not enough space in partition for buffer
14	Logical Unit is not assigned to a device
15	Empty system pool
16	File I/O error other than codes 11-15
17	File already open but cannot be closed
18	File name is greater than 8 characters (only first 6 are used); probably missing space between file name and extension or missing quote (')
19	Incorrect number of files specified; input does not meet program requirement

5.4.2 $SLIP Control Record Errors

Selecting a nonexistent processing option causes a nonfatal $SLIP control record error of the following form:

```
SLIP DOES NOT HAVE A x OPTION
```

where x is the nonexistent option.

5.4.3 SLIP Command Syntax Errors

SLIP command syntax errors are more serious, for they might cause considerable damage to a file. Therefore, most errors of this type cause SLIP to perform a reasonable action. It usually runs through to the end-of-file, copying the rest of the input file into the output file.

Illegal characters in the command line are detected, however. They cause the following message to appear:

```
CORRECTION CARD ERROR -- ILLEGAL CHARACTER x
```

where x is the illegal character. Because the faulty line is treated as if it were

```
-99999
```

SLIP processes through to the end-of-file, making no further corrections. The remaining command records are ignored.

5.4.4 SLIP Sequencing Errors

A SLIP sequencing error is caused by a reference to a line number that has already been passed. This indicates that the correction deck is out of order. SLIP exits immediately after producing a message of the following form:

```
CORRECTION CARD ERROR

            CURRENT LINE = xx

            AFFECTED LINE - xx -- xx
```

If an existing file bears the same name as the output file specified, the existing file is retained. If not, a new but incomplete file is generated.

5.5 DEMONSTRATION OF SLIP EDITING

```
$LIST SLTJOB JOB

page eject

$JOB 50(002) DEMONSTRATION OF SLIP EDITING
$LIST SLTJOB JOB
$DECK TESTIN SRC
AAAAAAAA
BBBBBBBB
CCCCCCCC
DDDDDDDD
EEEEEEEE
FFFFFFFF
GGGGGGGG
HHHHHHHH
```

```
IIIIIIII
JJJJJJJJ
KKKKKKKK
LLLLLLLL
MMMMMMMM
$EOF
$LIST TESTIN SRC
$SLIP L,S
*FILE IN:17='TESTIN SRC' OUT:18='TESTOU SRC'  $
-$
$SLIP L,S
*FILE IN:17='TESTIN SRC' OUT:18='TESTOU SRC'  $
-3
XXXXXXXX
YYYYYYYY
ZZZZZZZZ
-$
$LIST TESTOU SRC
$SLIP L,S
*FILE IN:17='TESTIN SRC' OUT:18='TESTOU SRC' $
-2,5
XXXXXXXX
YYYYYYYY
ZZZZZZZZ
-$
$LIST TESTOU SRC
$SLIP L,S
*FILE IN:17='TESTIN SRC' OUT:18='TESTOU SRC'  $
-'GGG'
-2,2
XXXXXXXX
-$
$LIST TESTOU SRC
$DECK EXTIN SRC
/THIS IS A USEFUL WAY
/OF INSERTING OPERATING
/INSTRUCTIONS INTO A LISTING WITHOUT
/HAVING TO TYPE THEM TWO OR MORE TIMES.
$EOF
$LIST EXTIN SRC
$SLIP L,S
*FILE IN:17='TESTIN SRC' OUT:18='TESTOU SRC' $
-3*
*FILE IN:15='EXTIN SRC' $
-6*
*FILE IN:15='EXTIN SRC' $
-$
$LIST TESTOU SRC
$SLIP L,S
*FILE IN:17='TESTIN SRC'  OUT:18='TESTOU SRC'  $
-'HHH',2*

*FILE IN:15='EXTIN SRC'  $
-$
$LIST TESTOU SRC
$END
$DECK TESTIN SRC
$LIST TESTIN SRC
```

```
page eject

AAAAAAAA
BBBBBBBB
CCCCCCCC
DDDDDDDD
EEEEEEEE
FFFFFFFF
GGGGGGGG
HHHHHHHH
IIIIIIII
JJJJJJJJ
KKKKKKKK
LLLLLLLL
MMMMMMMM

page eject

$SLIP L,S
*FILE IN:1717='TESTIN SRC' OUT:18'TESTOU SRC'  $



**CHANGE**  -$
          1 AAAAAAAA
          2 BBBBBBBB
          3 CCCCCCCC
          4 DDDDDDDD
          5 EEEEEEEE
          6 FFFFFFFF
          7 GGGGGGGG
          8 HHHHHHHH
          9 IIIIIIII
         10 JJJJJJJJ
         11 KKKKKKKK
         12 LLLLLLLL
         13 MMMMMMMM


            ********** S L I P  C O M P L E T E **********


page eject

$SLIP L,S
*FILE IN:17='TESTIN SRC' OUT:18='TESTOU SRC'  $



**CHANGE**  -3
          1 AAAAAAAA
          2 BBBBBBBB
          3 CCCCCCCC
          4 XXXXXXXX
```

```
          5 YYYYYYYY
          6 ZZZZZZZZ
**CHANGE**  -$
          7 DDDDDDDD
          8 EEEEEEEE
          9 FFFFFFFF
         10 GGGGGGGG
         11 HHHHHHHH
         12 IIIIIIII
         13 JJJJJJJJ
         14 KKKKKKKK
         15 LLLLLLLL
         16 MMMMMMMM


            ********** S L I P  C O M P L E T E **********



$LIST TESTOU SRC

page eject

AAAAAAAA
BBBBBBBB
CCCCCCCC
XXXXXXXX
YYYYYYYY
ZZZZZZZZ
DDDDDDDD
EEEEEEEE
FFFFFFFF
GGGGGGGG
HHHHHHHH
IIIIIIII
JJJJJJJJ
KKKKKKKK
LLLLLLLL
MMMMMMMM

page eject

$SLIP L,S
*FILE IN:17='TESTIN SRC' UT:18='TESTOU RC' $




**CHANGE**  -2,5
          1 AAAAAAAA
**DELETE**  BBBBBBBB
          2 XXXXXXXX
          3 YYYYYYYY
          4 ZZZZZZZZ
**CHANGE**  -$
```

```
**DELETE**  CCCCCCCC
**DELETE**  DDDDDDDD
**DELETE**  EEEEEEEE
          5 FFFFFFFF
          6 GGGGGGGG
          7 HHHHHHHH
          8 IIIIIIII
          9 JJJJJJJJ
         10 KKKKKKKK
         11 LLLLLLLL
         12 MMMMMMMM


           ********** S L I P  C O M P L E T E **********



$LIST TESTOU SRC

page eject

AAAAAAAA
XXXXXXXX
YYYYYYYY
ZZZZZZZZ
FFFFFFFF
GGGGGGGG
HHHHHHHH
IIIIIIII
JJJJJJJJ
KKKKKKKK
LLLLLLLL
MMMMMMMM

page eject

$SLIP L,S
*FILE IN:17='TESTIN RC' OUT:18='TESTOU SRC'  $



**CHANGE**  -'GGG'
          1 AAAAAAAA
          2 BBBBBBBB
          3 CCCCCCCC
          4 DDDDDDDD
          5 EEEEEEEE
          6 FFFFFFFF
          7 GGGGGGGG
**CHANGE**  -2,2
**DELETE**  HHHHHHHH
          8 XXXXXXXX
**CHANGE**  -$
          9 IIIIIIII
```

```
        10 JJJJJJJJ
        11 KKKKKKKK
        12 LLLLLLLL
        13 MMMMMMMM


           ********** S L I P  C O M P L E T E **********



$LIST TESTOU SRC

page eject

AAAAAAAA
BBBBBBBB
CCCCCCCC
DDDDDDDD
EEEEEEEE
FFFFFFFF
GGGGGGGG
XXXXXXXX
IIIIIIII
JJJJJJJJ
KKKKKKKK
LLLLLLLL
MMMMMMMM
$DECK EXTIN SRC
$LIST EXTIN SRC

page eject

/THIS IS A USEFUL WAY
/OF INSERTING OPERATING
/INSTRUCTIONS INTO A LISTING WITHOUT
/HAVING TO TYPE THEM TWO OR MORE TIMES.

page eject

$SLIP L,S
*FILE IN:17='TESTIN SRC' OUT:18='TESTOU SRC' $



**CHANGE**  -3*
*FILE IN:15='EXTIN SRC' $
          1 AAAAAAAA
          2 BBBBBBBB
          3 CCCCCCCC
          4 /THIS IS A USEFUL WAY
          5 /OF INSERTING OPERATING
          6 /INSTRUCTIONS INTO A LISTING WITHOUT
          7 /HAVING TO TYPE THEM TWO OR MORE TIMES.
**CHANGE**  -6*
```

```
*FILE IN:15='EXTIN SRC' $
          8 DDDDDDDD
          9 EEEEEEEE
         10 FFFFFFFF
         11 /THIS IS A USEFUL WAY
         12 /OF INSERTING OPERATING
         13 /INSTRUCTIONS INTO A LISTING WITHOUT
         14 /HAVING TO TYPE THEM TWO OR MORE TIMES.
**CHANGE**  -$
         15 GGGGGGGG
         16 HHHHHHHH
         17 IIIIIIII
         18 JJJJJJJJ
         19 KKKKKKKK
         20 LLLLLLLL
         21 MMMMMMMM


            ********** S L I P  C O M P L E T E **********



$LIST TESTOU SRC

page eject

AAAAAAAA
BBBBBBBB
CCCCCCCC
/THIS IS A USEFUL WAY
/OF INSERTING OPERATING
/INSTRUCTIONS INTO A LISTING WITHOUT
/HAVING TO TYPE THEM TWO OR MORE TIMES.
DDDDDDDD
EEEEEEEE
FFFFFFFF
/THIS IS A USEFUL WAY
/OF INSERTING OPERATING
/INSTRUCTIONS INTO A LISTING WITHOUT
/HAVING TO TYPE THEM TWO OR MORE TIMES.
GGGGGGGG
HHHHHHHH
IIIIIIII
JJJJJJJJ
KKKKKKKK
LLLLLLLL
MMMMMMMM

page eject

$SLIP L,S
*FILE IN:17='TESTIN SRC'  OUT:18='TESTOU SRC'  $
```

```
**CHANGE**  -'HHH',2*
*FILE IN:15='EXTIN SRC' $
          1 AAAAAAAA
          2 BBBBBBBB
          3 CCCCCCCC
          4 DDDDDDDD
          5 EEEEEEEE
          6 FFFFFFFF
          7 GGGGGGGG
**DELETE**  HHHHHHHH
          8 /THIS IS A USEFUL WAY
          9 /OF INSERTING OPERATING
         10 /INSTRUCTIONS INTO A LISTING WITHOUT
         11 /HAVING TO TYPE THEM TWO OR MORE TIMES.
**DELETE**  IIIIIIII
**CHANGE**  -$
**DELETE**  JJJJJJJJ
         12 KKKKKKKK
         13 LLLLLLLL
         14 MMMMMMMM


            ********** S L I P  C O M P L E T E **********



$LIST TESTOU SRC

page eject

AAAAAAAA
BBBBBBBB
CCCCCCCC
DDDDDDDD
EEEEEEEE
FFFFFFFF
GGGGGGGG
/THIS IS A USEFUL WAY
/OF INSERTING OPERATING
/INSTRUCTIONS INTO A LISTING WITHOUT
/HAVING TO TYPE THEM TWO OR MORE TIMES.
KKKKKKKK
LLLLLLLL
MMMMMMMM
```

CHAPTER 6

TASK BUILDER AND BASIC TASK BUILDER: BUILDING A TASK FOR EXECUTION

The Task Builder is an interactive program used to build user and system tasks from relocatable binary files. This program links the user's binary files to library functions to create an executable task. By responding to questions asked by the Task Builder, the user supplies information on run priority, core partition, COMMON block requirements, and other task operating characteristics.

The Task Builder is flexible, allowing the user to build a simple one-program task with no overlays or very complex tasks made up of multiple programs (some written in FORTRAN and some written in MACRO), a resident portion, and multiple overlays and suboverlays. Tasks may be built for a variety of execution modes with the following characteristics:

1. Uses floating-point hardware or not
2. Runs in BANK addressing mode or in PAGE mode
3. Runs in USER mode (protected) or EXEC mode (privileged)
4. If in USER mode, runs in XVM mode (wide addressing) or not

The user generates a system of overlays - a resident main program which may include resident subprograms, a resident blank COMMON storage area, and a set of subroutines which overlay each other at the user's request. Subroutines are organized into units called LINKs which may overlay each other. Several LINKs may overlay a larger LINK without overlaying each other. A LINK is loaded into core when a subroutine within the LINK is called and it remains resident until overlayed. A LINK's core image is not recorded or "swapped out" when it is overlayed. The same image is brought into core each time a LINK is loaded.

For the user who wants to build a simple Task, the flexibility of the Task Builder can be a nuisance. For that reason, a second version of the Task Builder whose task name is BTK... (Basic Task Builder) is provided. BTK... assumes answers to all the questions that the standard Task Builder, TKB..., asks directly. Section 6.5 describes BTK in more detail.

All disk-resident RSX tasks are built and incorporated in the system in the same way. Throughout the process, the task name is usually used as file name, but the extension changes. Tasks are assembled or compiled from source code to produce relocatable binary (extension BIN) files. The binary files (combined perhaps with other binary files, typically library subprograms) are fed through the Task Builder to produce a binary task file (extension TSK), which is one step closer to the executable form than BIN files. Finally, TSK files are

read by the FINING task or the INSTALL function task, which converts the task file into absolute core-image form, stores it on the disk for rapid loading and records its existence in the System Task List.

6.1 INVOKING THE TASK BUILDER

The Task Builder is invoked under TDV in the following way:

Form:	TDV>TKB⍽

Control is passed to the Task Builder, which establishes dialogue with the user by typing a question and a prompting character (>), and expecting a response. An example of user interaction with TKB appears in Figure 6-1. In all cases, user responses are identified by their placement after the prompting character. Each TKB question is described in detail later in this chapter.

```
TDV>TKB
TASK BUILDER XVM V1A003
LIST OPTIONS
>BKR,NRM,NFP
NAME TASK
>STA...
SPECIFY DEFAULT PRIORITY
>
DESCRIBE PARTITION
>
DEFINE RESIDENT CODE
>STATUS,CORE,USERS,TASKS
DESCRIBE LINKS & STRUCTURE
>

ALLOC. STRATEGY:BOTTOM UP
ACTUAL PARTITION SIZE:032000
EFFECTIVE PARTITION SIZE:032000
VIRTUAL PARTITION SIZE:032000


STATUS      000020-000724
CORE        000725-001003
USERS       001004-001421
TASKS       001422-001677
DATF.5 SRC  001700-001737
SPYF.1 SRC  001740-001763
.DA    015  001764-002063
BCDIO  056  002064-006121
STOP   008  006122-006135
FIOPS  047  006136-007056
DBLINT 007  007057-007455
INTEAE 009  007456-007606
RELEAE 011  007607-010705
OTSER  016  010706-011225
SPMSG  014  011226-011355
.CB    004  011356-011377
.FP    000  011400-011401

MINIMUM EFFECTIVE PARTITION SIZE:012000

CORE REQ'D
            000000-011401
TDV>
```

Figure 6-1
Task Builder Session

An altmode is a null entry. To terminate command input prematurely, the user should type CTRL/Q followed by a carriage return in response to any command output message. This causes the Task Builder to exit.

6.2 INPUT/OUTPUT

The following LUN assignments should be made before the Task Builder is invoked under TDV:

LUN	Assignment
10	System Library input
11	User Library input
12	Terminal input
13	Terminal output
17	Binary file input
18	Task file output

If an I/O error occurs during Task Building, the following message:

TKB - I/O ERROR LUN xx yyyyyy

is produced on LUN-13; xx represents the Logical Unit Number (decimal) and yyyyyy the octal Event Variable indicating the cause of the error.

6.3 COMPARISON WITH CHAIN AND EXECUTE DIALOGUE

The operations of the Task Builder under RSX are very similar to those of the CHAIN and EXECUTE programs which run under control of DOS. Interaction with these programs is described in detail in the CHAIN XVM/EXECUTE XVM UTILITY MANUAL. Note that all command lines logically terminate with an ALTMODE character. Lines terminated by a carriage return are continued on the next physical line; therefore, a logical command line may consist of several physical lines.

User interaction with the Task Builder differs in some ways from the descriptions in the manual. The following sections summarize those components of the dialogue which remain the same as well as those which differ significantly under RSX.

6.3.1 List Options

Any of the list, execution mode, and library options, shown in Table 6-1 may be specified. If more than one option is included, entries must be separated by commas.

Table 6-1
Task Builder List, Execution Mode, and Library Options

Option	Action	Default
BKR	BANK-mode relocation (13-bit addressing)	PGR
BUFFS:n	Reserve n decimal I/O buffers of 422 octal words each when calculating effective partition size	n = 0
EXM	EXECutive mode (neither protection nor relocation)	NRM
FP	Hardware Floating-Point Library (floating-point hardware available)	See MFP option
GM	Output global symbol and file name in load maps	Output program names in load maps
IOT[1]	Allows task to issue IOT instructions. (Permitted only for USER-mode tasks)	Don't allow task to issue IOT instruction unless task is any EXEC-mode task
NFP	Non-hardware Floating-Point Library (no floating point hardware available)	In default, the FP/NFP option is dynamically determined by the availability of floating-point hardware
NM	No load map	Output load map
NRM	USER-mode (protection and relotation)	NRM
PAL	Pause after outputting each link	No pause after outputting each link
PAR	Pause after outputting resident code	No pause after outputting resident code
PGR	Page-mode relocation (12-bit addressing)	PGR

[1] XVM hardware must exist on the installation if this option is to be used.

Table 6-1 (Cont.)
Task Builder List, Execution Mode, and Library Options

Option	Action	Default
RES/name,...,name/	Force the COMMONs named to be part of the resident code and load them from the top of the virtual partition space down	Make only those COMMONs declared in the resident code resident
SAC	Single allocation of COMMON blocks. Elements of labelled COMMONs may be referenced by any co-resident link	Elements of labelled COMMONs may not be referenced by any co-resident link
SHR[1]	Some COMMON blocks should be allocated memory within Shared Addressing Space. Request specification of these COMMONs later in dialogue, (permitted only for USER-mode tasks)	There will be no COMMONs allocated in Shared Addressing Space
SL: name	System Library name (alternate user-specified System Library name)	.LIBRX (non-floating-point hardware) or .LIBFX (floating-point),depending on choice of NFP or FP, respectively
SZ	Output size in load maps	No size in load maps
UL: name	User Library name (alternate User Library name)	.LIBR5
XVM[1]	XVM mode or 17-bit indirect addressing mode, (permitted only for USER-mode tasks)	15-bit addressing

[1] XVM hardware must exist on the installation if this option is to be used.

6.3.1.1 PAGE Mode - Bits 6-17 of a memory-reference instruction are taken as an operand address, and bit 5 is used to select address modification via the Index Register (XR). Thus 4K of core is directly addressable.

6.3.1.2 BANK Mode - Bits 5-17 of a memory reference instruction are taken as an operand address. Thus 8K of core is directly addressable, but the Index Register cannot be used for address modification.

6.3.1.3 EXEC Mode - Tasks running in EXEC mode are not restricted in the core they may reference or alter, or in the instructions which they may execute. I/O Handler and MCR Function tasks must run in EXEC mode. Hardware relocation is not used in this mode. Thus the 15-bit addressing range limits EXEC-mode tasks to partitions below 32K. If a machine does not have hardware relocation, all tasks must be run in EXEC mode. EXEC-mode tasks are assumed to be debugged and well-behaved. For this reason, the system performs practically no checking on EXEC-mode tasks.

6.3.1.4 USER Mode - This is a nonprivileged mode in which a task is prohibited access or execution outside its partition. It may not direct the system by means of I/O Directives or System Directives to alter any core partition but its own.

The only exceptions are:

1. USER-mode Tasks may transmit data to and receive data from System COMMON Blocks by using the COMMON Communicator I/O Handler task as an intermediary.

2. USER-mode Tasks may transmit data to and receive data from System COMMON Blocks or partitions using XVM core sharing hardware, the SHR option, and the SHARE directive.

3. USER-mode Tasks may access memory via the SPY and SPYSET directives.

A USER-mode Task is also prohibited by the hardware from executing such privileged instructions as HLT, IOT, OAS, or double XCT (unless the IOT option was specified when the task was built).

Reasonable protection is provided, but this mode should not be considered "idiot proof," because a USER-mode Task is not restricted in issuing Directives which affect areas other than core. Such a Task may, for example, interfere with the scheduling of other Tasks; it may tie up I/O devices by attaching them indefinitely; and it may enter tight loops, making continued system requests that could exhaust the Pool of Empty Nodes.

USER-mode Tasks are relocated to zero, and use hardware relocation and upper-bound checking to effect memory protection. Unless XVM mode is specified, 15-bit addressing range limits the size of such a task's useful partition space to 32K. However, hardware relocation permits positioning of partitions for USER-mode tasks anywhere in 128K.

The protected/relocated mode is called "user" mode to emphasize the fact that it is the normal mode; the mode recommended for tasks, unless they must run unprotected. A task that is to run "protected" is not protected from other tasks; rather, other tasks are protected from it. Such protection is necessary when debugging a new task.

Most tasks under RSX can be aborted (i.e., forced to exit). In this event, I/O rundown is performed for the task. For example, if a task should exit, leaving some device attached and several files open on the disk, I/O Rundown is invoked. In effect, this detaches the device and closes all of the files.

An advantage to building user-mode tasks is that their partitions can be redefined without requiring that the tasks be rebuilt. All user tasks that run under MULTIACCESS must be built in user mode to preserve system integrity.

If XVM mode as well as user mode is specified for a task, that task can indirectly access memory via 17-bit addressing. Such a task could access up to 128K if such a partition were allowed in the system. In practice, however, the maximum partition size is 114K*. Despite this wide addressing capability, the executable code and initialized COMMON blocks** of such a task cannot exceed 32K. Nevertheless, XVM mode is highly useful if a task contains extensive code as well as large uninitialized COMMON blocks. The ability to access such COMMON blocks is the primary use of XVM mode. The XVM option depends on the existence of XVM hardware. Users without an XVM or PDP-15 with an XM-15 option should not attempt to build a task in XVM mode.

6.3.2 Name Tasks

The user specifies a one- to six-character task name identifying the task to be built.

6.3.3 Specify Default Priority

The user specifies the default priority at which a task will run. This is an optional parameter. It can be entered or changed when the task is installed. If specified, the priority must be between 1 (highest priority) and decimal 512 (lowest priority). If not specified, TKB assumes a default priority of 400.

---

*Since 128K is the maximum supported core size and must include space for the Executive, I/O handlers and MCR partition, the maximum partition size is 114K.

**COMMON blocks are initialized by block data subprograms in FORTRAN and by the .CBS/.CBC/.CBE pseudo-ops in MACRO.

6.3.4 Describe Partition

The user identifies the core partition in which the task will run, using the form:

Form:	[name[(size)]] for a user-mode task [name[(base,size)]] for an exec-mode task
Where:	name identifies a partition that has already been defined base is the octal starting address of the partition size is the octal size of the partition
Example:	DESCRIBE PARTITION >P40.0

Typing an altmode causes the task being built to be relocated for the partition currently in use by the user's copy of TKB. This feature is available for convenience. Normally, the MULTIACCESS Monitor dynamically selects the partition assignment of all user-mode tasks executed under MULTIACCESS.

If the partition base and size are unspecified, the named partition is presumed to exist in the current configuration. The Partition Block Description List (PBDL) is scanned by the Task Builder for the named partition and the existing values of base and size are used. Alternately, if the partition does not exist in the current configuration and the task is being built to run in user mode (relocated), only the partition size need be specified; the base can be omitted. Exec-mode tasks require both base and size specifications.

The execution mode is not a characteristic of a given partition. Consequently, the user can build tasks with different execution modes to run within the same partition. Partitions must be a multiple of 400 (octal) words in size. Partitions can be defined starting immediately above the top of the Executive.

The size of the partition specified to TKB or the existing value of the partition size is called the actual partition size. This value is reduced by the buffer space, specified by the BUFFS option, to obtain what is called the effective partition size. This is the space that could be used by the task, assuming that it were able to address it. For user-mode tasks, buffer space is always rounded to the next highest increment of 400 (octal) words to permit effective use of the memory-protection/relocation hardware. The virtual partition size is that part of the effective partition size that the task can address. Usually the effective partition size is equal to the virtual partition

only reason the effective partition size might exceed the virtual partition size is if the effective partition size were greater than the maximum virtual partition size for a task built with a particular set of options. The maximum virtual partition size for a task can be determined from the following chart.

Options Specified	Maximum Virtual Partition Size
EXM	32K
NRM (not XVM and not SHR)	32K
NRM, XVM (not SHR)	128K
NRM, SHR (not XVM)	24K
NRM, XVM, SHR	120K

The Task Builder allocates code and common blocks within the virtual partition space with one exception. Shared common blocks are always allocated memory within the shared address space (SAS) which is immediately above the virtual partition space whenever the SHR option is specified.

6.3.5 Describe System or Shared COMMON Blocks

If the task is being built in EXEC mode, the user will be asked to specify System COMMON blocks. If the task is being built in USER-mode and the SHR option was declared, the user will be asked to specify Shared COMMON Blocks. The way in which such COMMON blocks are indicated is quite similar but there are some differences between the formats used to specify System or Shared COMMONs.

6.3.5.1 Describe System COMMON Blocks

Here the user identifies COMMON blocks which are referenced by the task but may be common to all tasks in RSX. These System COMMON Blocks are used for communication between tasks; however, a task may also have its own internal COMMON blocks.

EXEC-mode tasks can indirectly access System COMMONs provided such COMMONs lay within the first 32K of core. This is because EXEC-mode tasks reside within the lower 32K of memory and run unprotected. EXEC-mode tasks cannot indirectly access system COMMONs whichreside above 32K.

Each such COMMON block is specified according to the following format:

Form:	name(base,size)[,...]
Where:	name identifies a COMMON block which has been defined in RSX at System Startup time base is the octal starting address of the COMMON block size is the octal maximum size of the COMMON block
Example:	DESCRIBE SYSTEM COMMON BLOCKS >FLAG(36400,400)

The user can enter as many as four COMMON block descriptions by inserting commas between entries. Those COMMON blocks in the user program which are not declared to the Task Builder to be System COMMON Blocks are allocated within the task's virtual partition space and are referenced only by that task. Blank COMMON has the name .XX.

6.3.5.2 Describe Shared COMMON Blocks

The user must identify those COMMON Blocks referenced by the task which should be allocated memory within the task's Shared Address Space. These COMMONs must be named and uninitialized. The physical memory actually accessed when the task references a word in such a COMMON will depend upon the state of the MM register at the time of the access. The MM register will be set when the task issues the SHARE directive. Once the SHARE directive has been issued with the correct parameters, XVM hardware will map the task's accesses to Shared Addressing Space (SAS) into other locations within physical memory. Since a task's SAS is not part of the task's virtual partition space, accesses to shared COMMONs prior to issuing the SHARE directive will cause a memory protection violation, forcing the task to be aborted.

SAS is divided into Internal Shared Address Space (ISAS) and External Shared Address Space (ESAS). ISAS is always 400 octal words long and its base is identical to the base of SAS. ESAS begins at the base of SAS plus 400 octal words (i.e., at the end of ISAS). The length of ESAS depends upon parameters given in the SHARE directive. For tasks not built in XVM-mode, SAS starts at word 60000 octal relative to the partition's base. SAS starts at word 360000 octal relative to the partition's base for tasks built in XVM-mode.

When the task has issued the SHARE directive, accesses to ISAS will be mapped into the first 400 octal words of the task's partition.

References to ESAS will be mapped into an area of physical memory specified in the SHARE directive.

The use of the SHR option depends upon the existence of XVM hardware and the SHARE directive. If the user does not have such hardware or the NOXM assembly parameter was defined when the RSX executive was assembled, no tasks should be built in SHR mode.

Each Shared COMMON Block is specified according to one of the following formats:

Form 1: name (offset, size) [,...]

Form 2: name (size) [,...]

Form 3: name [,...]

In these formats, name identifies a COMMON Block declared by the task. In Form 1, offset is added to the base of ESAS to obtain the base address of the COMMON named. The offset from ESAS can be negative but cannot cause a shared COMMON to start outside of the task's SAS. For example, it is possible to set the offset equal to -400 octal. This would cause the base of the named COMMON to coincide with the base of SAS. Offset and size should be specified as octal numbers. Size specifies the length in words, of the shared COMMON Block. Note that size also specifies the base address of the next shared COMMON relative to the base of the current shared COMMON. The only exception to this rule is that a specification made using Form 1 will override all previous specifications. The value of size must be a positive octal number and cannot direct TKB to allocate space for a shared COMMON outside SAS. Form 3 is simply a short form for Form 2 with a size of zero, (e.g., "COM," means "COM(0),"). The use of each of these forms is apparent from the following example.

Shared COMMON Definition	COMMON Base
COM1(-400, 1000),	SAS = ESAS - 400
COM2(100),	SAS + 1000 = ESAS + 400
COM3,	SAS + 1100 = ESAS + 500
COM4,	SAS + 1100 = ESAS + 500
COM5(0,200),	SAS + 400 = ESAS
COM6(100),	SAS + 600 = ESAS + 200
COM7(2000, 0)	SAS + 2400 = ESAS + 2000

If forms 2 or 3 are used prior to form 1, the base of the first shared COMMON named is made to coincide with the base of ESAS.

The user can enter as many shared COMMON Blocks as are declared in the task by inserting commas between entries. All shared COMMONs must lay within SAS. Those COMMON Blocks in the user's program which are not declared to be shared are allocated within the task's virtual partition wall and are referenced only by that task. Blank COMMON cannot be shared.

Since the Task Builder does not allocate space for shared COMMONs within the task's virtual partition space, the actual size of any shared COMMON is irrelevant. Hence, when calculating the size of a shared COMMON to be printed in load maps, TKB assumes that its top address coincides with the top of ESAS.

6.3.6 Define Resident Code

Here the user lists the names of files containing relocatable binary units of routines to be resident throughout a run and the names of library routines (flagged by library indicators (#)) to be resident throughout a run. These names are listed in the following format:

Form:	**name[,...]**
Where:	**name is the name of a file or of a library routine (see above) to be resident throughout a run**
Example:	**DEFINE RESIDENT CODE** **>ZZZ.12,#LIB1**

RSX transfers initial control to the entry point of the first resident routine relocated, i.e., the first routine of the first file listed, unless resident code consists exclusively of library routines. The response to DEFINE RESIDENT CODE must be at least one name.

6.3.7 Describe Links and Structures

Here the user describes the overlay structure in terms of LINK names. When a LINK is to consist of only one external component, the name of the file containing the external component may be used as the LINK name. However, when a LINK is to consist of more than one external component, the LINK must be named and defined.

In the DOS system, the supervision of core overlays is handled by a system program called EXECUTE, which exists as a separate file from the user's XCT file built by CHAIN. In XVM/RSX, the equivalent of EXECUTE is a subroutine called EXU.13 (the number may vary). Whenever the Task Builder constructs a file with overlays, it expects to find EXU.13 in the system library, .LIBRX or .LIBFX. If the appropriate library file is not present on LUN-10 when an overlay task is being built, an error message will be printed during the expansion of the resident code:

```
TKB-I/O ERROR LUN 10 13
```

Code 13 means that the file was not found.

6.3.7.1 LINK Definitions - Each LINK definition requires one line of command input in the following format:

Form:	name = {extfile[,...][/][intfile][,...] / [extfile][,...][/]intfile[,...]}
Where:	name is the name of the LINK extfile is the name of an external LINK component intfile is the name of an internal LINK component
Example:	Define the LINK named ABC to consist of external components SUB1 and SUB2 and internal components SUB3 and SUB4: DESCRIBE LINKS & STRUCTURES >ABC=SUB1,SUB2/SUB3,SUB4

A LINK definition is a list of the names of files which contain the relocatable binary units that comprise the LINK components. The individual file names listed are separated by commas (,); the two types of LINK components which may be used (external and internal) are separated within the definition by a slash (/). All external LINK component names must be listed before (to the left of) the slash separator; all internal LINK components must be listed after (to the right of) the slash. External LINK components are accepted only from files with names which match the external component name (i.e., GLOBAL symbol definition).

Rules for defining a LINK:

1. A LINK may not be a component of another LINK.

2. The names of the components of a LINK may not be used as LINK names. When a LINK consists of only one component, the component's file name may be used as the LINK name in the overlay structure description, but not in a LINK definition; i.e., it is not necessary to define a single component LINK but, if defined, the LINK name cannot be the component name.

3. A file name used in the resident code description cannot be used in a LINK definition.

4. A file name preceding a slash may be used only once.

5. A file name following a slash may be used in other LINK definitions (following a slash).

6.3.7.2 Overlay Structure Description - An overlay structure is described using the names of defined LINKS, or the names of files containing LINK components and the operators colon (:) and comma (,).

This description has the following basic format:

Form:	overlaid: [...]overlaying[,...]
Where:	overlaid is the name of the part of the structure to be overlaid (a defined LINK or a file containing a LINK component) overlaying is the name of the part of the structure to overlay overlaid (a defined LINK or a file containing a LINK component)
Example:	SUB2 overlays SUB1: DESCRIBE LINKS & STRUCTURES SUB1:SUB2

The following rules apply:

1. A line is an independent statement processed from left to right.

2. A colon signifies "is overlayed by." Core mapping, but no loading order, is implied.

3. A comma signifies "and." The following:

```
SUB1:SUB2
SUB2:SUB3,SUB4
```

   indicates that SUB1 is overlayed by (uses the same core as) SUB2, SUB2 is overlayed by SUB3 and SUB4, but SUB3 and SUB4 do not overlay each other.

4. A colon operator may not be used in a line after a comma has been used. This restriction prevents the following ambiguity:

```
SUB2:SUB3,SUB4:SUB5
```

   The above line is rejected because it is not clear whether SUB5 overlays SUB3 or SUB4 or both. All four of the following examples are acceptable:

```
SUB2:SUB3,SUB4          SUB5 uses the same core as
SUB4:SUB5               SUB4 but not the same core
                        as SUB3.

SUB2:SUB3,SUB4          SUB5 uses the same core as
SUB3:SUB5               SUB3 but not the same core
                        as SUB4.
```

SUB2:SUB5:SUB3,SUB4	SUB5 uses the same core as SUB3 and SUB4. SUB3 and SUB4 are loaded individually (if nonresident) as called.
LINK=SUB3,SUB4 SUB2:LINK:SUB5	SUB5 uses the same core as SUB3 and SUB4. Both SUB3 and SUB4 are loaded (if nonresident) whenever either is called.

5. A LINK name may appear only once preceding a colon and only once following another colon.

6. If a LINK name is used twice, it must be used following a colon before being used before another colon.

7. Several LINKs overlaying each other may be defined in one statement, as in the following:

SUB1:SUB2:SUB3,SUB4	Core mapping, but no loading order, is implied. This is a short method of defining the same overlay structure as in the first example, under rule 3.

Although rules 5 and 6 may appear restrictive, they do not limit the user's description of an overlay structure, but do prevent multiple description of the position of a LINK in an overlay structure. A LINK may be both overlayed and overlaying, and it may not be possible or convenient to describe both conditions by using the LINK name only once, as follows:

SUB1:SUB2:SUB3	SUB2 is overlaying SUB1 and is overlayed by SUB3.

Therefore, when a LINK is both overlaying and overlayed, its LINK name may be used twice, but the LINK(s) overlayed by it must be described before the LINK(s) by which it is overlayed, as follows:

SUB2:SUB3,SUB4	SUB3 overlays SUB2.
SUB3:SUB5	SUB3 is overlayed by SUB5.

NOTE

The description of an overlay structure only defines a desired core mapping; i.e., stating that SUB1 is overlayed by SUB2 means that both are to be relocated to the same core and cannot coreside, but does not imply that SUB1 must be called before SUB2. There is no imposed order in which routines must be called, nor is there restriction of the routines callable by any routine.

6.3.8 Completion of Dialogue

After all of the previously described characteristics have been supplied by the user and accepted by TKB, the program computes the amount of core required by the task and prints it in the format:

Form:	start-end size
Where:	start is the octal starting address of the task in the core partition identified above end is the last filled octal location in this partition size is the octal size of the task
Example:	CORE REQ'D 40000-46266 06267

TKB terminates interaction with the user by returning control to the MULTIACCESS Monitor. When in control, the Monitor requests another function by typing:

```
TDV>
```

6.4 CONVERSION TO XVM/RSX

XVM/RSX user interaction with the Task Builder differs from interaction with previous versions of the Task Builder. All user FORTRAN programs developed under previous versions of RSX must be recompiled and rebuilt by the Task Builder to enable them to run under XVM/RSX. Similarly, assembly language programs should be reassembled and rebuilt for XVM/RSX operation.

6.5 BTK: BASIC TASK BUILDER

BTK, the Basic Task Builder, closely resembles TKB, but requires different input. BTK is useful for building simple tasks, particularly for batch-processing. BTK is invoked in the following way:

Form:	BTK name[,name...]∇
Where:	name is the name of the task to be built
Example:	TDV>BTK SCAN,SCSUB

BTK assumes the following options and conditions (refer to Table 6-1):

- NRM
- FP or NFP
- PGR
- priority of 400
- TDV partition
- The resident code that has been entered

BTK generates a load map. As an aid to the system, the batch handler removes trailing spaces from card images passed to the MULTIACCESS Monitor.

6.6 ERROR MESSAGES

Tables 6-2 and 6-3 list possible error messages and their implications. Messages regarding command string errors immediately follow the erroneous logical line. The command is ignored and must be retyped.

The > prompt follows the message if the error is recoverable. When possible, the faulty character or name is output after the error message. If the error is not recoverable, TKB exits to TDV after typing the message.

Below is a sample sequence of miscellaneous recoverable errors:

```
TDV>TKB
TASK BUILDER XVM V1A000
LIST OPTIONS
>BAR,SZ
^ UNRECOGNIZED SYMBOL -- BAR
>SZ
NAME TASK
>F2
SPECIFY DEFAULT PRIORITY
>600
-

^ IMPROPER PRIORITY
SPECIFY DEFAULT PRIORITY
>500
DESCRIBE PARTITION
>BLK1^ PARTITION NOT IN SYSTEM

>BLK
DEFINE RESIDENT CODE
>F1/F2
^ IMPROPER BREAK CHAR -- /
>F1,F2
DESCRIBE LINKS & STRUCTURE
>LK1$=SUB1/$DTA
^ LIB IND ON LINK NAME -- LK1
>LK1=SUB1,F2
^ EXTERNAL NAME USED PRV -- F2
>LK1=SUB1/LBDTA
-
>LK2=SUB2/LBDTA/SUB1

^ IMPROPER BREAK CHAR -- /
>LK2=SUB2/LBDTA
-
>LK1:LK2:F1
^ RES ROUTINE NAME USED AS LINK NAME -- F1
>LK1:LK2
>LK1:LK456
^ NAME USED LEFT OF COLON TWICE -- LK1
>LK2:LK456
>
```

Table 6-2
Messages Produced by Recoverable TKB Errors

Error Message	Meaning
↑UNRECOGNIZABLE SYMBOL	Unrecognizable symbol in command string
↑RES ROUTINE REQ'D	No resident routine has been declared
↑LINK NAME USED PRV	Name used previously
↑NAME LENGTH ERR	Legal name has 1-6 characters
↑IMPROPER BREAK CHAR	Break character used incorrectly
↑INTERNAL NAME REPEATED IN LINE	Internal LINK component name used more than once within LINK
↑EXTERNAL NAME USED PRV	External LINK component name used previously within overlay system
↑COMPONENT NAME USED AS LINK NAME	Name of LINK component used a s name of a LINK
↑LINK DEF WITHIN OVERLAY DESCRIPTION	LINK definition within overlay description
↑COLON MUST FOLLOW FIRST LINK NAME	Colon missing after first LINK name
↑MORE THAN ONE LINK OVERLAYED	More than one LINK overlayed
↑NAME RIGHT OF COLON USED PRV	LINK name used more than once to right of colon
↑NAME USED MORE THAN TWICE	LINK name used more than twice
↑NAME USED LEFT OF COLON TWICE	LINK name used more than once to left of colon
↑LIB IND ON LINK NAME	Library indicator (#) on LINK name
↑INTERNAL NAME USED PRV	Internal name used before
↑RES ROUTINE NAME USED AS LINK NAME	Name of resident routine used as LINK name
↑NAME USED MORE THAN ONCE	LINK name used more than once
INCONSISTENT OPTION	The option specified is incompatible with other declared options.

Table 6-2
Messages Produced by Recoverable TKB Errors

Error Message	Meaning
^IMPROPER PRIORITY	The specified priority is out of range (greater than 512)
^PARTITION STARTS ABOVE 32K	The partition for an exec-mode task starts above 32K
^PARTITION ENDS ABOVE 32K	The partition for an exec-mode task ends above 32K
^TOO MANY DEFINITIONS	The amount of table space in free core is exceeded for user information
^PARTITION NOT IN SYSTEM	The specified partition does not exist in the system
^ILLEGAL SIZE	The partition size is not a multiple of 400 (octal) words
^LIB IND ON EXTERNAL NAME	The user has incorrectly applied the library indicator to an external name
^LIB IND ON INTERNAL NAME	The user has incorrectly applied the library indicator to an internal name
^COMMON OUT OF SAS	The common block specified is larger than the shared address segment and cannot be addressed

Table 6-3
Messages Produced by Unrecoverable TKB Errors

Error Messages	Meaning
TABLE OVERLAP	Patch table and symbol table overlap occurred during relocation of resident code and links to the TSK file
READ ERROR	An error occurred on the input device
ILLEGAL LOADER CODE	The input file contains unrecognizable loader code
LABELED COMMON BLK SIZE ERR--	The labeled common block is now declared larger than in a previous declaration. The block name follows the message.
UNRESOLVED GLOBAL(S):	One or more global symbols are unresolved during relocation of the resident code and links to the TSK file. The system generates a list of all unresolved globals.
ABS PROG	.ABS programs are not allowed, because they are not relocatable
MISSING GLOBAL DEF	A global definition is missing. The system lists it.
DUPLICATE GLOBAL DEF	There is a duplicate global definition. The system lists it.
TASK IS LARGER THAN PARTITION	The task is too large for its partition
TKB-PARTITION TOO SMALL	The specified partition cannot run the job
TOO MANY BUFFERS SPECIFIED	The described buffer space exceeds the partition size
CORE OVERFLOW	The executable code portion, including initialized common blocks of the overlay structure defined by the user, does not fit into available core
MODULE TOO LARGE--	The named module is larger than 4K for page mode or 8K for bank mode. The module name follows the message.

(Continued)

Table 6-3 (Cont.)
Messages Produced by Unrecoverable TKB Errors

Error Messages	Meaning
COMMON BLOCKS DECLARED RESIDENT	A resident module has been specified with the same name as a common block
ABSOLUTE LOAD ADDRESS	A module has been read that has an absolute starting address
GLOBAL SYMBOL TOO BIG--	The user has attempted to reference an area beyond the scope of the given symbol. The symbol follows the message.
ILLEGAL ATTEMPT TO INITIALIZE COMMON BLOCK--	The user has attempted to do one of the following: 1. Initialize a common block that is part of another link 2. Initialize a common block that has been declared with the RES option 3. Initialize a shared common block 4. Initialize a system common block
*** BLOCK DATA SUBROUTINE--	A block data subroutine has been encountered. The subroutine name follows the message.
COMMON BLOCK TOO BIG--	The user has attempted to allocate a common block larger than 77777. The block name follows the message.

FIN
DEC
FOU
LIS
TYPE

CHAPTER 7

FILE INPUT, DECK, FILE OUTPUT, LIST AND TYPE

The FILE INPUT, DECK, FILE OUTPUT, FILE LIST and TYPE TDV Function tasks transfer sequential-access files from one device to another. The four tasks are built and invoked in identical fashion. They differ only in their assignment of LUNs for I/O. They perform the following operations:

- FILE INPUT (FIN) transfers a sequential file from LUN-19 to LUN-17
- DECK (DEC) transfers a sequential file from LUN-14 to LUN-17
- FILE OUTPUT (FOU) transfers a sequential file from LUN-17 to LUN-19
- FILE LIST (LIS) transfers a sequential file from LUN-17 to LUN-16; a line printer or some other printing device
- TYPE (TYPE) is identical to LIS, but outputs data to LUN-13

These function tasks can also be invoked in batch mode.

The most common use for these functions is for backup and restoration. FOU is often used to backup a disk file on DECtape and sometimes on magtape or paper tape. FIN is used to restore the copied file to disk. LIS usually dumps a file from disk to the line printer. DEC is usually used during batch operations to transfer a file from LUN-14 (normally assigned to the card reader) to LUN-17 (normally assigned to a disk). It is possible to use FIN, DEC, FOU, LIS and TYPE for any copy function, simply by reassigning LUNs.

7.1 INVOKING FIN, DEC, FOU, LIS AND TYPE

FIN, DEC, FOU, LIS and TYPE are invoked according to the same format:

Form:	{FIN / DEC / FOU / LIS[T] / TYPE} [9] [option[,]...]name [ept][,.,]∇
Where:	9 preceding the first space character signifies 9-track magtape operation. In its absence, 7-track operation is assumed. option is a one-character symbol: N or F. N specifies no parity check; F allows form feeds in the output (see the following paragraphs for more information). The option characters can be either concatenated or separated by commas. name is a string of one to six .SIXBT characters and represents the input file. At least one name must be supplied. ext is a string of one to three .SIXBT characters and identifies a file extension. It is optional. SRC is the default.
Example:	Copy a binary file from DECtape (LUN-19) to disk (LUN-17): TDV>FIN SCAN BIN Make no parity check on the file NOCHEK SRC, but perform parity checking on CHECK SRC and CHECK2 SRC: TDV>FIN NNOCHEK,CHECK,CHECK2 SRC Insert form feeds in the output for the files FORMAT SRC, FORMT2 SRC, and FORMT3 SRC, but not for NOFMT SRC: TDV> LIS FFORMAT,FORMT2,NOFMT,FFORMT3 Perform parity checking, but not formatting, for NOFMT SRC, NOFMT2 SRC and NOFMT3 SRC (A is ignored as an illegal option letter): TDV>LIS NOFMT,NOFMT2,ANOFMT3

The options N and F are entered by typing either or both option letters followed by the file name, and so on. When the back-arrow appears with no preceding option characters, it is assumed that these options are not to be used. Because the characters space, comma and

back-arrow are treated as delimiters in the command string, they cannot be used in file names even though they are part of the .SIXBT character set.

The N option suppresses the typeout of parity error messages (N stands for "no parity check"). It is used to allow input of nonstandard paper tapes by using the RSX paper tape reader handler task PR.....

The F option allows form feeds to be inserted in the output at expected locations. The purpose is to format FORTRAN and MACRO language source programs on the output listing device. The F option has no effect on binary files. A form feed is inserted after every line that contains an .EJECT optionally preceded by any number of spaces and horizontal tabs. A form feed is inserted prior to every line containing a .TITLE optionally preceded by any number of spaces and horizontal tabs. Finally, a form feed is inserted after every 56 lines of text following the last form feed.

Files with extensions of BIN or TSK are transferred in IOPS binary data mode. All others are transferred in IOPS ASCII mode. Image mode files are not supported.

$DECK causes Batch to read all lines following it and preceding $EOF. It resembles a FIN function task, because it performs transfers of data, FORTRAN programs and so forth to disk. It can transfer anything but a job file. The sequence below is legal:

```
$DECK name1, name2, name3

$EOF

$EOF

$EOF
```

7.2 INPUT/OUTPUT AND TASK BUILDING

The FILE INPUT, DEC, FILE OUTPUT, FILE LIST and TYPE TDV Function tasks are assigned task names FIN..., DEC..., FOU..., LIS... and TYP..., respectively, at task-building time. These tasks can be built to run in either user mode (protected and relocated) or exec mode. All of these tasks expect LUN-13 (recommended dedicated terminal) to receive error messages and expect the following LUNs as file devices:

Task Name	Input LUN	Recommended Input Device	Output LUN	Recommended Output Device
FIN...	19	DECtape	17	Disk
FOU...	17	Disk	19	DECtape
LIS...	17	Disk	16	Line printer
DEC...	14	Card reader	17	Disk
TYP...	17	Disk	13	Terminal

For all three Tasks, if the input device has a directory, a SEEK is invoked to open a file for input. If not, an ATTACH is issued to obtain exclusive control of the device. Similarly, if the output device has a directory, ENTER opens a file for output; if not, an ATTACH is issued. If several files are being transferred, ATTACH-DETACH is done for each file, to allow a higher-priority Task to use the device if necessary.

If the output device is a line printer or terminal, a page eject is performed before each file is transferred. If the input device is a terminal, a check is made for end-of-file by examining the first word of the input line for CTRL/D followed by ALTMODE or carriage return.

7.3 MAGTAPE OPERATION

Since the software cannot determine dynamically whether a tape drive is 9-track or 7-track, 7-track operation is assumed unless the user specifies a 9 in the appropriate place in the command line.

The following FORMAT parameters are issued for the different data modes:

Mode	Parameters
9-track ASCII	9-track; 800 BPI; odd parity
9-track BINARY	9-track; 800 BPI; odd parity; core-dump mode
7-track ASCII	7-track; 800 BPI; even parity
7-track BINARY	7-track; 800 BPI; odd parity

Data cannot be transferred from a 7-track drive to a 9-track drive, or vice versa, without using the disk as an intermediary.

Whenever end-of-tape is encountered after a READ, WRITE, or WRITE END-OF-FILE, the tape is dismounted (REWIND followed by a SPACE FORWARD RECORD) and the following messages are printed (example is for tape unit 3):

```
FIN-DISMOUNT MT3
FIN-THEN, "RESUME FIN..."
```

As soon as tape motion has stopped and the message printouts have ended, the function (FIN... in this case) suspends itself. This allows the operator to unload the tape manually and mount a continuation tape at the load point if desired. When the new tape is ready, the operator must resume the indicated Task. If no continuation is desired, he must abort the file transfer function.

7.4 ERROR MESSAGES

Table 7-1 lists possible error messages and their implications. All messages are printed in the following format:

```
TDV>FIN string
FIN-message
TDV>
```

In the above example and in Table 7-1, FIN can be replaced by FOU or LIS. All messages are applicable to the three Tasks.

Table 7-1
FILE INPUT, OUTPUT, and LIST Error Messages

Error Message	Meaning	System Action
FIN-LINE TOO LONG	Command string exceeds permissible length (75 characters)	Command ignored
FIN-SYNTAX ERROR	Violation in command string formation	Remainder of command ignored; because some files may have been transferred, the user should request a DIRECTORY LIST
FIN-HINF ERROR	Input or output device Handler does not perform HINF Directive; possible that nothing is assigned to the input or output LUN	Command ignored
FIN-NOT INPUT DEV	I/O Handler assigned to input LUN cannot perform input; assignments should be checked	Command ignored
FIN-NOT OUTPUT DEV	I/O Handler assigned to output LUN cannot perform output; assignments should be checked	Command ignored
FIN-FILE NOT FOUND	One of the specified files cannot be found in the file directory	Remainder of command ignored; because some files may have been transferred, the user should request a DIRECTORY LIST

(Continued on next page)

Table 7-1 (Cont.)
FILE INPUT, OUTPUT, and LIST Error Messages

Error Message	Meaning	System Action
FIN-SEEK ERR	Error (other than FILE NOT FOUND) occurred while SEEK Directive was being processed	Remainder of command ignored; because some files may have been transferred, the user should request a DIRECTORY LIST
FIN-ATTACH ERR	ATTACH Directive to the input or output device rejected (ATTACH is not issued to a directoried device); error might occur in the unlikely event that the input or output LUN was reassigned to another device before completion of the transfer	Command ignored
FIN-ENTER ERR	Error occurred while ENTER Directive was being processed	Remainder of command ignored; because some files may have been transferred, the user should request a DIRECTORY LIST
FIN-READ ERR	Error (other than PARITY, CHECKSUM, or BUFFER OVERFLOW) occurred while READ Directive was being processed	Remainder of command ignored; because some files may have been transferred, the user should request a DIRECTORY LIST
FIN-PARITY ERR	Parity error exists somewhere within the device block (usually 256 words) from which the last input record was read	File transfer continues after printing error message; file probably contains altered data as a result of this error
FIN-CHECKSUM ERR	CHECKSUM error exists in data record just read	File transfer continues after printing error message; file contains altered data as a result of this error
FIN-BUF OVERFLOW	Record larger than 68 words (including header) encountered in input	Only a partial record is written; file transfer continues after printing error message

(Continued on next page)

Table 7-1 (Cont.)
FILE INPUT, OUTPUT, and LIST Error Messages

Error Message	Meaning	System Action
FIN-WRITE ERR	Error occurred while WRITE Directive was being processed	Remainder of command ignored; because some files may have been transferred,the user should request a DIRECTORY LIST
FIN-CLOSE INPUT ERR	Error occurred while CLOSE Directive on input file was being processed	Remainder of command ignored; because some files may have been transferred,the user should request a DIRECTORY LIST
FIN-CLOSE OUTPUT ERR	Error occurred while CLOSE Directive on output file was being processed	Remainder of command ignored; because some files may have been transferred, the user should request a DIRECTORY LIST
FIN-DETACH ERR	DETACH Directive issued to the input or output device was rejected (DETACH is not issued to a directoried device)	Remainder of command ignored
FIN-FILE STILL OPEN	Some other Task is referencing the file in a manner that conflicts with the current request	Remainder of command ignored; because some files may have been tranferred, the user should request a DIRECTORY LIST
FIN-DISMOUNT MT3 FIN-THEN, "RESUME FIN..."	End-of-tape reached during Magtape READ, WRITE, or WREOF	Task is suspended until resumed after a new tape is mounted or until aborted
FIN-WRITE EOF ERR	Error occurred while writing an end-of-file mark on Magtape	Remainder of command ignored; because some files may have been transferred, the user should request a DIRECTORY LIST

DEL

CHAPTER 8

DELETE: DELETING A FILE FROM DISK

The DELETE FILE TDV Function Task is used to DELETE files from disk. It can be invoked in batch mode.

8.1 INVOKING DELETE FILE

The user can invoke DELETE FILE by typing a command according to the following format:

Form:	DEL[ETE] name[ext][,name[ext]...]▽
Where:	name of file DELETEd is a string of one to six .SIXBT characters; if more than one name is included, the entries must be separated by commas ext is a string of one to three .SIXBT characters and identifies a file extension; SRC is the default extension
Examples:	TDV>DELETE FILE TDV>DEL F1,F2,F3 TDV>DEL F1,F2 003,F3 BIN,F4 SRC TDV>

8.2 INPUT/OUTPUT AND TASK BUILDING

The DELETE FILE TDV Function Task is assigned Task name DEL... at Task-Building time. This Task can be built to run in either USER (protected and relocated) or EXEC mode. It expects LUN-13 (recommended dedicated terminal) to receive error messages and LUN-17 (normally disk) as the file device.

8.3 ERROR MESSAGES

Table 8-1 lists possible error messages and their implications. All messages are printed in the following format:

```
TDV>DEL
DEL-message
TDV>
```

Table 8-1
DELETE FILE Error Messages

Error Message	Meaning	System Action
DEL-LINE TOO LONG	Command string exceeds permissible length (75 characters)	Command ignored
DEL-SYNTAX ERR	Violation in command string formation	Remainder of command ignored; because some files may have been DELETEd, the user should request a DIRECTORY LIST
DEL-FILE NOT FOUND	One of the specified files cannot be found in the file directory	Remainder of command ignored; because some files may have been DELETEd, the user should request a DIRECTORY LIST
DEL-DELETE ERR	DELETE Directive rejected; possible that: 1. no I/O Handler assigned to proper LUN 2. I/O Handler does not perform DELETE 3. disk hardware error occurred	Remainder of command ignored; because some files may have been DELETEd, the user should request a DIRECTORY LIST
DEL-FILE STILL OPEN	Some other Task is referencing the file in a manner that conflicts with the current request	Remainder of command ignored; because some files may have been DELETEd, the user should request a DIRECTORY LIST

REN

CHAPTER 9

RENAME: RENAMING A FILE STORED ON DISK

The RENAME TDV Function Task is used to RENAME files stored on disk. It can be invoked in batch mode.

9.1 INVOKING RENAME FILE

The user can invoke RENAME FILE by typing a command according to the following format:

Form:	REN[AME] old [oldext], {new [newext] / [new] newext} ▽
Where:	old is a string of one to six .SIXBT characters and represents the old name of the file oldext is a string of one to three .SIXBT characters and identifies the old file extension; SRC is the default extension new is a string of one to six .SIXBT characters and represents the new name of the file; old is the default newext is a string of one to three .SIXBT characters and identifies the new file extension; oldext or its default (SRC) is the default
Examples:	TDV>RENAME FILE1,FILE2 TDV>REN OLDFIL SRC,NEWFIL 003 TDV>REN OLDFIL 002, 003 TDV>REN OLDFIL BIN,NEWFIL TDV>

Either a new name or a new extension is required.

9.2 INPUT/OUTPUT AND TASK BUILDING

The RENAME FILE TDV Function Task is assigned Task name REN... at Task-Building time. This Task can be built to run in either USER (protected and relocated) or EXEC mode. It expects LUN-13 (recommended dedicated terminal) to receive error messages and LUN-17 (disk) as the file device.

9.3 ERROR MESSAGES

Table 9-1 lists possible error messages and their implications. All messages are printed in the following format:

```
TDV>REN string
REN-message
TDV>
```

Table 9-1
RENAME FILE Error Messages

Error Message	Meaning	System Action
REN-LINE TOO LONG	Command string exceeds permissible length (75 characters)	Command ignored
REN-SYNTAX ERR	Violation in command string formation	Command ignored
REN-FILE NOT FOUND	Specified file cannot be found in the file directory	Command ignored
REN-CAN'T RENAME	I/O Handler does not perform RENAME (only for disk Handlers)	Command ignored
REN-RENAME ERR	Error occurred while file being opened or closed; disk hardware error possible	Command ignored (there is a remote chance that RENAME was performed if disk error occurred while rewriting UFD block)
REN-FILE STILL OPEN	Some other Task is referencing the file in a manner that conflicts with the current request	Command ignored

CHAPTER 10

DIRECTORY LIST: LISTING FILES IN DISK DIRECTORY

The DIRECTORY LIST TDV Function Task is used to list all files in a file directory on RF DECdisk, RK cartridge disk, or RP disk pack. These may be either sequential or random-access files.

This Function Task can be invoked in batch mode.

10.1 INVOKING DIRECTORY LIST

The user can invoke DIRECTORY LIST by typing a command according to the following format:

Form:	DIR[ECTORY LIST] Rmn<UFD> ▽
Where:	m is F, K, or P and represents the type of disk: RF DECdisk, RK cartridge disk, or RP disk pack respectively n represents the unit number of the disk UFD is the name of the user file directory
Example:	TDV> DIR RK2<ABC> 3027 USER BLOCKS 6361 FREE BLOCKS FILE RND 426 20 3-AUG-71 1000 10 TASK01 BIN 446 3 18-AUG-71 DIRECT 008 0 0 2-SEP-71*

In the above example, the number of user blocks is the amount of storage already allocated on the disk. The number of free blocks is the amount of storage available to the user. Each of the next output lines has the following components:

- File name: One to six characters
- File Name Extension: One to three characters
- Starting Block Number: One- to six-character octal number (0 if sequential-access file is truncated)
- File Size (Number of Blocks): One- to six-character octal number (0 if sequential-access file is truncated)
- File Creation Date: In form day-month-year where day and year have one or two decimal digits
- Truncation Mark: Asterisk if file is truncated. Truncation usually indicates a file in the process of being created but occasionally identifies one that was never properly closed. The latter might occur as the result of a disk hardware error. "Truncation" applies only to sequential-access files.
- Random-Access Information: Two one- to six-character octal numbers of accounting information are supplied if it is a random-access file. If the file was created using the FORTRAN CALL DEFINE statement, these numbers represent the number of records in the file and the record size, respectively. If the file was created using MACRO, the user determines the meaning of these numbers.

  Bit 0 in the second word is 0 to indicate a BINARY (unformatted) file or 1 to indicate an ASCII (formatted) file.

10.2 INPUT/OUTPUT AND TASK-BUILDING

The DIRECTORY LIST TDV Function Task is assigned Task name DIR... at Task-Building time. This Task can be built to run in either USER (protected and relocated) or EXEC mode. It expects LUN-13 (recommended dedicated terminal) to receive error messages, LUN-16 to accept the file listing, and LUN-1 for the Multi-Disk Driver Task.

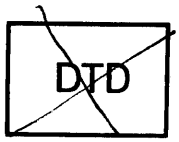
10.3 ERROR MESSAGES

Table 10-1 lists possible error messages and their implications. All messages are printed in the following format:

```
TDV>DIR
DIR-message
TDV>
```

Table 10-1
DIRECTORY LIST Error Messages

Error Message	Meaning	System Action
DIR-ATTACH ERR	ATTACH Directive to the listing device rejected; possible that nothing is assigned to the listing LUN	Command ignored
DIR-NOT A LISTING DEV	HINF function indicates that the device assigned to the listing LUN is not a listing device (i.e., it cannot perform output or it has a directory)	Command ignored
DIR-DISK ERR	Attempt to read in a UFD block failed; possible disk hardware malfunction	Printing of DIRECTORY ceases
DIR-PRINTOUT ERROR	WRITE to the listing device declared to be in error; error does not occur without drastic cause (e.g., exhausting Pool of Empty Nodes)	Printing of DIRECTORY ceases
DIR-DETACH ERR	DETACH Directive to the listing device declared to be in error; error does not occur unless operator reassigns listing LUN in midstream	Command ignored
DIR-EMPTY	No files in the file directory	Command ignored
DIR-UFD DOES NOT EXIST	The UFD specified is not listed in the disk's MFD (Master File Directory)	Command ignored
DIR-TDV ERROR	DIR... could not read the TDV command string	Command ignored
DIR-FORMAT ERROR	Format error detected in command string	Command ignored
DIR-NON-EXISTENT DISK	Disk specified in command string is logically not present	Command ignored
DIR-DEVICE IS NOT A DISK	Device name typed was not RF, RK, or RP	Command ignored



CHAPTER 11

DECTAPE DIRECTORY: LISTING FILES IN DECTAPE DIRECTORY

The DECTAPE DIRECTORY LIST TDV Function Task is used to list all files recorded in the file directory of a DECtape in standard format.

11.1 INVOKING DECTAPE DIRECTORY LIST

The user can invoke DECTAPE DIRECTORY LIST by typing a command according to the following format:

Form:	DTD[IRECTORY LIST] ▽
Example:	TDV>DTD DECTAPE UNIT 4 31-DEC-71 1004 FREE BLKS 3 USER FILES 10 SYSTEM BLKS FILNAM EXT 1 15 TABLE BIN 2 23 DTD.1 SRC 3 24 TDV>

DTD prints out the following:

- DECtape Unit Number
- Today's Date: In form day-month-year where day and year have one or two decimal digits
- Free Blocks: Number of free blocks in the file directory in octal format
- User Files: Number of user files in the file directory in octal format; this is equivalent to the number of files listed below; system tapes consist of user files and system files; both are listed, but system files are not included in the count of user files
- System Blocks: Number of system blocks in the file directory in octal format; the minimum number of system blocks is 10 octal; system blocks include blocks occupied by the file directory (blocks 71 through 100 octal) and system files (none if tape initialized and written under RSX control)

Each of the next output lines has the following components:

- File name: One to six characters
- File Name Extension: One to three characters
- Starting Block Number: one- to six-character octal number
- File Size (Number of Blocks): One- to six-character octal number (0 if system file, not user file)

11.2 INPUT/OUTPUT AND TASK BUILDING

The DECTAPE DIRECTORY LIST TDV Function Task is assigned Task name DTD... at Task-Building time. This Task can be built to run in either USER (protected and relocated) or EXEC mode. It expects LUN-13 (recommended dedicated terminal) to receive error messages, LUN-13 to accept the file listings, and LUN-19 for DECtape input.

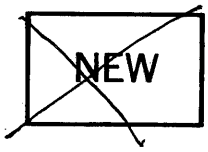
11.3 ERROR MESSAGES

Table 11-1 lists possible error messages and their implications. All messages are printed in the following format:

```
TDV>DTD
DTD-message
TDV>
```

Table 11-1
DECTAPE DIRECTORY LIST Error Messages

Error Message	Meaning	System Action
DTD-ATTACH ERR	ATTACH Directive rejected; possible that proper I/O Handler not assigned to the input or listing LUNs	Command ignored
DTD-NOT A LISTING DEV	HINF function indicates that the device assigned to the listing LUN is not a listing device (i.e., it cannot perform output or it has a directory)	Command ignored
DTD-NOT DECTAPE	Input LUN not assigned to the DECtape Handler	Command ignored
DTD-DECTAPE ERR	Attempt to read in one of the DECtape directory blocks failed	Command ignored
DTD-PRINTOUT ERR	WRITE to the listing device declared to be in error; error does not occur without drastic cause (e.g., exhausting Pool of Empty Nodes)	Printing of DIRECTORY ceases
DTD-DETACH ERR	DETACH Directive to the listing device or to the DECtape Handler declared to be in error; error does not occur unless operator reassigns listing LUN in midstream	Command ignored



CHAPTER 12

NEW DECTAPE DIRECTORY: WRITING NEW DECTAPE DIRECTORY

The NEW DECTAPE DIRECTORY TDV Function Task is used to write a new file directory on DECtape in standard format.

12.1 INVOKING NEW DECTAPE DIRECTORY

The user can invoke NEW DECTAPE DIRECTORY by typing a command according to the following format:

Form:	NEW[DIRECTORY] ▽
Example:	TDV>NEW NEW DIRECTORY ON DECTAPE UNIT 4 TDV>

12.2 INPUT/OUTPUT AND TASK-BUILDING

The NEW DIRECTORY TDV Function Task is assigned Task name NEW... at Task-Building time. This Task can be built to run in either USER (protected and relocated) or EXEC mode. It expects LUN-13 (recommended dedicated terminal) to receive error and confirmation messages and LUN-19 for DECtape output.

12.3 ERROR MESSAGES

Table 12-1 lists possible error messages and their implications. All messages are printed in the following format:

```
TDV>NEW
NEW-message
TDV>
```

Table 12-1
NEW DIRECTORY Error Messages

Error Message	Meaning	System Action
NEW-ATTACH ERR	ATTACH Directive to the output device rejected; possible that DECtape I/O Handler not assigned to the output LUN	Command ignored
NEW-NOT DECTAPE	Output LUN not assigned to the DECtape I/O Handler	Command ignored
NEW-DECTAPE ERR	Attempt to write out one of the DECtape directory blocks failed	No further action
NEW-DETACH ERR	DETACH Directive to the listing device rejected; error does not occur unless operator reassigns listing LUN in midstream	Command ignored

INS

CHAPTER 13

INSTALL: INSTALLING A TASK IN THE SYSTEM

The INSTALL TDV Function task (also an MCR function task) adds a task to RSX previously built using the Task Builder. The Task Builder creates a binary file as output. When the INSTALL TDV Function task is invoked, the binary file is read from LUN-5 and recorded as an absolute image on the disk. The existence of the task is recorded in the System Task List.

This function task has been implemented for batch processing, not for use under the MULTIACCESS Monitor. Its inclusion in the system (via atypical system build procedures) is permitted only to maintain compatibility with existing user batch streams.

13.1 INVOKING INSTALL

The user can invoke INSTALL by typing a command according to the format:

Form:	INS[TALL] name [p]∇
Where:	name of task to be INSTALLed is a string of one to six .SIXBT characters p is an integer in the decimal range 1 to 512, specifying the task priority
Examples:	Priority has been set during task building: TDV>INSTALL SCAN TDV> Priority was not set at task building time or priority redefined here as 10: TDV>INS SCAN 10 TDV>

The user can override a priority specified during task building by indicating a priority in the INSTALL command line. If a priority has not previously been specified during task-building, an INSTALL priority is a required parameter.

13.2 INPUT/OUTPUT AND TASK-BUILDING

The INSTALL Function Task is assigned task name INS... at Task-Building time. This task must be built to run EXEC mode. It expects LUN-13 (recommended dedicated terminal) to receive error messages and LUN-5 for binary file input.

13.3 ERROR MESSAGES

Table 13-1 lists possible error messages and their implications. All messages are printed in the following format:

```
TDV>INS
INS-message
TDV>
```

Table 13-1
INSTALL Error Messages

Error Message	Meaning	System Action
INS-SYNTAX ERROR	Task name omitted or priority invalid	Command ignored
INS-TASK ALREADY IN SYSTEM	Task to be INSTALLed has node in STL	Command ignored
INS-PARTITION NOT IN SYSTEM	Partition name specified at Task-Building time not available	Command ignored
INS-TASK WOULD OVER-FLOW PARTITION	Task to be INSTALLed too large for available partition	Command ignored
INS-OUT OF POOL	No nodes left in pool to create new STL entry	Command ignored
INS-OUT OF DISK	No room left on the disk	Command Ignored
INS-INPUT CHECKSUM ERR	Error while performing checksum processing	Command ignored
INS-INPUT PAR ERR	Error while performing parity checking	Command ignored
INS-SYS COM BLK ERR	INSTALL needs system common block not currently in system	Command ignored
INS-READ ERR ON LUN-5	Error in reading binary file from LUN-5	Command ignored
INS-DISK ERR	Error while performing disk get or allocate operation	Command ignored
INS-NO DEFAULT PRIORITY	Priority not specified in INSTALL command, and no priority included at Task-Building time	Command ignored
INS-FILE NOT FOUND ON LUN-5	Binary file not available for input from LUN-5	Command ignored
INS-RELOCATION HARD-WARE NOT AVA	No relocation hardware available on machine	Command ignored
INS-FLOATING POINT HARDWARE NOT AVA	No floating-point hardware available on machine	Command ignored

REQ

CHAPTER 14

REQUEST: REQUESTING TASK EXECUTION

The REQUEST TDV Function task (also an MCR function task) REQUESTs the execution of a task at a specified software priority. Actual time of execution depends on task priority and partition availability.

This function task has been implemented for batch processing, not for use under the MULTIACCESS Monitor. Its inclusion in the system (via atypical system build procedures) is permitted only to maintain compatibility with existing user batch streams.

Tasks requested by this TDV function will not execute under control of the MULTIACCESS Monitor.

14.1 INVOKING REQUEST

The user can invoke REQUEST by typing a command according to the format:

Form:	REQ[UEST] name [p]∇
Where:	name of task REQUESTed is a string of one to six .SIXBT characters p is an integer in the decimal range 1 to 51, specifying the task priority
Examples:	Priority has been set during task building or installation: TDV>REQUEST SCAN TDV> Priority is redefined here at 50: TDV>REQ SCAN 50 TDV>

The priority that has been specified during task building or installation can be overridden when the task is REQUESTed. If a new priority is not included in the command line, the task runs at the previously specified default priority.

14.2 INPUT/OUTPUT AND TASK BUILDING

The REQUEST TDV Function task is assigned task name REQ... at task-building time. This task can be built to run in either user mode (protected and relocated) or exec mode. It expects LUN-13 (recommended dedicated terminal) to receive error messages.

14.3 ERROR MESSAGES

Table 14-1 lists possible error messages and their implications. All messages are printed in the following format:

```
TDV>REQ
REQ-message
TDV>
```

Table 14-1
REQUEST Error Messages

Error Message	Meaning	System Action
REQ-SYNTAX ERR	Task name omitted or priority invalid	Command ignored
REQ-TASK NOT IN SYSTEM	STL node for REQUESTed Task cannot be found	Command ignored
REQ-TASK ALREADY ACTIVE	REQUESTed Task is currently active	Command ignored
REQ-TASK DISABLED	Task has been disabled and is unavailable	Command ignored
REQ-POOL EMPTY	No nodes left in pool to create new CKQ entry	Command ignored
REQ-PART LOST	Partition in which Task is to run has been lost because of reconfiguration	Command ignored

REM

CHAPTER 15

REMOVE: REMOVING A TASK FROM THE SYSTEM

The REMOVE TDV Function task (also an MCR function task) deletes an inactive task from the system. If the user plans to alter and reinstall a task, he must REMOVE it first.

This function task has been implemented for batch processing, not for use under the MULTIACCESS Monitor. Monitor. Its inclusion in the system (via atypical system build procedures) is permitted only to maintain compatibility with existing user batch streams.

15.1 INVOKING REMOVE

The user can invoke REMOVE by typing a command according to the format:

Form:	REM[OVE] nameV
Where:	name of task to be REMOVEd is a string of one to six .SIXBT characters
Example:	TDV>REMOVE SCAN TDV>

15.2 INPUT/OUTPUT AND TASK BUILDING

The REMOVE TDV Function task is assigned task name REM... at task-building time. This task must be built to run in exec mode. It expects LUN-13 (recommended dedicated terminal) to receive error messages.

15.3 ERROR MESSAGES

Table 15-1 lists possible error messages and their implications. All messages are printed in the format:

```
TDV>REM
REM-message
TDV>
```

Table 15-1
REMOVE Error Messages

Error Message	Meaning	System Action
REM-TASK ACTIVE	Task to be REMOVEd is currently active	Command ignored
REM-SYNTAX ERR	Task name omitted	Command ignored
REM-TASK NOT IN SYSTEM	STL node for task to be REMOVEd cannot be found	Command ignored
REM-DISK ERR	Error while performing disk GET operation	Command ignored
REM-ALLOCATE ERROR	Error while performing disk ALLOCATE operation	Command ignored
REM-TASK HAS MULTIPLE STL ENTRIES	Disk space allocated to this task cannot be deallocated at this time	Command ignored

MNT

CHAPTER 16

MOUNT: LOGICALLY MOUNTING A DISK

The MNT TDV Functions task (also an MCR function task) specifies UFDs for all LUNs assigned to the named disk where no UFD specification has previously been made. For additional information, refer to the documentation on the MNT MCR Function task in Part IV of this manual.

This function task is not recommended for use under the MULTIACCESS Monitor. Its inclusion in the system (via atypical system build procedures) is permitted only to maintain compatability with existing user batch streams.

16.1 INVOKING MOUNT

The user can invoke MOUNT by typing a command according to the format:

Form:	MNT Rnm UFDV
Where:	Rn is disk type: RF, RP or RK m is a valid disk unit number UFD is a valid three-character user file directory
Example:	TDV>MNT RP3 ABN TDV>

16.2 INPUT/OUTPUT AND TASK BUILDING

The MOUNT TDV Function task is assigned task name MNT... at task-building time. This task must be built to run in exec mode. It expects LUN-13 (recommended dedicated terminal) to receive error messages.

16.3 ERROR MESSAGES

Table 16-1 lists possible error messages and their implications. All messages are printed in the format:

```
TDV>MNT
MNT-message
TDV>
```

Table 16-1
MOUNT Error Messages

Error Message	Meaning	System Action
MNT-ALLOCATION ERROR	Error while performing disk ALLOCATE operation	Command ignored
MNT-DISK PUT ERROR	Error while performing disk PUT operation	Command ignored
MNT-FORMAT ERROR	Invalid device name, unit, or UIC	Command ignored
MNT-DISK HAS NO PDVL NODE	PDVL node for specified disk cannot be found	Command ignored
MNT-DEVICE IS NOT A DISK	Device name does not correspond to a disk in the RSX system	Command ignored
MNT-ILLEGAL TO MOUNT THE SYSTEM DISK	Device name corresponds to the system disk	Command ignored
MNT-DISK NOT DISMOUNTED	Specified disk has not been dismounted since last mount	Command ignored
MNT-DISK GET ERROR	Error while performing disk GET operation	Command ignored

DSM

CHAPTER 17

DISMOUNT: LOGICALLY DISMOUNTING A DISK

The DSM TDV Function task (also an MCR function task) dismounts a user disk and, therefore, disables all file-oriented I/O addressed to the disk. For additional information, refer to the documentation on the DSM MCR Function task in Part IV of this manual.

This function task is not recommended for use under the MULTIACCESS Monitor. Its inclusion in the system (via atypical system build procedures) is permitted only to maintain compatability with existing user batch streams.

17.1 INVOKING DISMOUNT

The user can invoke DISMOUNT by typing a command according to the format:

Form:	DSM Rnm∇
Where:	Rn is disk type: RF, RP or RK m is a valid disk unit number
Example:	TDV>DSM RK6 DISK IS READY FOR DISMOUNTING TDV>

17.2 INPUT/OUTPUT AND TASK BUILDING

The DISMOUNT TDV Function task is assigned task name DSM... at task-building time. This task can be built to run in exec mode. It expects LUN-13 (recommended dedicated terminal) to receive error messages.

17.3 ERROR MESSAGES

Table 17-1 lists possible error messages and their implications. All messages are printed in the format:

```
TDV>DSM
DSM-message
TDV>
```

Table 17-1
DISMOUNT Error Messages

Error Message	Meaning	System Action
DSM-FORMAT ERROR	Invalid device name or unit	Command ignored
DSM-DISK HAS NO PDVL NODE	PDVL node for specified disk cannot be found	Command ignored
DSM-DEVICE IS NOT A DISK	Device name does not correspond to a disk in the RSX system	Command ignored
DSM-ILLEGAL TO DISMOUNT THE SYSTEM DEVICE	Device name corresponds to the system disk	Command ignored
DSM-DEVICE NOT MOUNTED	Disk to be dismounted is not currently mounted	Command ignored
DSM-DEVICE IS IN USE	Specified disk is currently being used for I/O	Command ignored
DSM-MARK TIME ERROR	Error encountered while marking time until all open files are closed	Command ignored

This page intentionally left blank.

This page intentionally left blank.

CON

CHAPTER 19

CONSTRUCT: STORING A TASK ON A USER DISK

The CONSTRUCT TDV Function task (also an MCR function task) adds a task to RSX previously built by the Task Builder. When CONSTRUCT is invoked, the binary file is read from LUN-5. Unlike the INSTALL function, however, the task core image is stored in a created file on a disk, not simply in allocated space on the system disk. Because space on the user disk is allocated by the disk file handler, a LUN that specifies the disk on which the created file is to reside must usually be included in the CONSTRUCT command. Unlike INSTALL, CONSTRUCT does not affect the System Task List in any way.

19.1 INVOKING CONSTRUCT

The user can invoke CONSTRUCT by typing a command according to the format:

Form:	CON[STRUCT] name [LUN]∇
Where:	name of task to be CONSTRUCTed is a string of one to six .SIXBT characters LUN is an integer representing a logical unit number currently associated with a disk (the default value is decimal 14)
Example:	TDV>CON SCAN 16 TDV>

19.2 INPUT/OUTPUT AND TASK BUILDING

The CONSTRUCT TDV Function task is assigned task name CON... at task-building time. This task can be built to run in either user mode (protected and relocated) or exec mode. It expects LUN-13 (recommended dedicated terminal) to receive error messages and LUN-5 to be assigned for binary file input.

19.3 ERROR MESSAGES

Table 19-1 lists possible error messages and their implications. All messages are printed in the format:

```
TDV>CON
CON-message
TDV>
```

Table 19-1
CONSTRUCT Error Messages

Error Message	Meaning	System Action
CON-SYNTAX ERROR	Task name or LUN omitted or invalid or out-of-range LUN	Command ignored
CON-CREATE ERR	Error while performing create operation	Command ignored
CON-READ ERR	Error while performing READ operation	Command ignored
CON-DISK ERR	Error while performing disk GET or ALLOCATE operation	Command ignored
CON-FILE NOT FOUND	Binary file not available on LUN-5	Command ignored

This page intentionally left blank.

This page intentionally left blank.

This page intentionally left blank.

This page intentionally left blank.

QUE

CHAPTER 21

QUEUE: QUEUING A BATCH JOB

The QUEUE TDV Function task (also an MCR function task) informs the batch processor that a job is ready to be run, whether or not the batch handler is in core. The user can specify the name of the job to be queued, the LUN from which it comes and a series of job characteristics, including:

- Maximum time that the job can run (in minutes)
- Class at which the job can run
- Memory use
- Use of sequencing (run in order of submission)
- Whether the job requires operator availability
- Whether QUEUE expects to find the job on the specified device at this time
- Whether the job file should be deleted after the job runs
- Use of hold mode
- Whether the job is to be forced

21.1 INVOKING QUEUE

Job characteristics can be delimited in the command string by one or more spaces, commas or both. The following examples show various forms of delimited parameters. The user can invoke QUEUE by typing a command according to the format:

Form:	QUE[UE] [name] [LUN] [T=time] [C=class] [M=memory] [SEQ] [OPR] [NCK] [DEL] [HLD] [STK] [FRC]∇
Where:	name of task to be queued is a string of one to six characters (first character must be alphabetic). EXT MUST BE "JOB" LUN is an integer representing a logical unit number currently in the system time is an integer in the decimal range 1 to 1023, representing the maximum number of minutes that the job can run class is an integer in the decimal range 0 to 7 memory is an integer in the decimal range 1 to 128, representing memory use (in K)
Examples:	TDV>QUE COMPIL TDV>QUE NAMX 15 SEQ C=2 TDV>QUE SCAN 5 T=C C=3 M=28 SEQ SPN TDV>QUE 14 NCK

All parameters of the QUEUE command are optional. In practice, however, a name or LUN is often specified for one of the following reasons:

1. The default LUN is LUN-17; usually assigned to the disk. Specification of an explicit LUN is necessary to override this default.

2. The name can be omitted if the LUN from which it comes is not associated with a file-oriented device. In the example:

   ```
   TDV>QUE 14
   ```

   14 is associated with the card reader and the command requires no explicit name definition.

3. If the first character of a QUEUE command line is numeric, it is assumed that the name has been omitted and the number represents the LUN of a non-file-oriented device. The user should ensure that names included in the command line begin with an alphabetic character.

The HLD option causes QUEUE to queue the job, but to leave it in the hold mode until the operator releases it. The STK option causes QUEUE to stack the job by copying it into a temporary file on disk and queuing it. STK implies deletion after execution.

If the file is on DECtape or disk and if NCK and STK have not been specified, QUEUE opens the file, reads the JOB record and outputs errors, if necessary. If a memory size is specified and not enough core is available to run, the job waits until the TDV partition is large enough.

Job information defaults are:

Parameter	Default
T	Time specified in the job file
C	Zero
M	Core available at the time the job is run (0)
SEQ	No use of sequencing
OPR	Job does not require operator availability
NCK	Job is on the specified device at this time
DEL	Preserve the job file after the run
HLD	Run the job as soon as priority permits
STK	Do not copy the job file to the disk
FRC	Do not force

21.2 INPUT/OUTPUT AND TASK BUILDING

The QUEUE Function task is assigned task name QUE... at task building time. This task can be built to run in either user mode (protected and relocated) or exec mode. It uses LUN-13 to receive error messages.

21.3 ERROR MESSAGES

Table 21-1 lists possible error messages and their implications. All messages are printed in the format:

```
TDV>QUE
message
TDV>
```

Table 21-1
QUEUE Error Messages

Error Message	Meaning	System Action
FORMAT ERROR IN COMMAND	One or more nonexistent options in command line	Command ignored
ILLEGAL VALUE FOR ARGUMENT	Illegal T,C,M, or U argument	Command ignored
CANNOT INPUT FROM DEVICE SPECIFIED	Nothing to input from that LUN	Command ignored
NEED FILE NAME FOR THIS DEVICE	File-oriented LUN	Command ignored
FILE NOT FOUND	File cannot be found	Command ignored
READ ERROR	Error during READ operation	Command ignored
JOB RECORD MUST BE FIRST LINE	First line not JOB record	Command ignored
INCORRECT JOB LINE FOLLOWS	Incorrect time limit specification; not three digits or not in correct sequence	Command ignored
IMPOSSIBLE TO QUEUE JOB	Exec or empty nodes unavailable	Command ignored
TDV COMMAND TRANSFER ERROR	Error in TDV Command transfer	Command ignored

CHAPTER 22

ODT: OCTAL DEBUGGING TECHNIQUE

The ODT TDV Function task allows users to debug tasks with an "octal debugging technique" under XVM/RSX. It permits users to start and stop tasks, examine and modify task registers and locations within the task partition, set and remove breakpoints within the task, and proceed with task execution after a breakpoint has been reached. ODT allows the user to define symbols for program addresses so that command parameter inputs can be given as either octal numbers or previously defined symbols. Furthermore, ODT monitors task progress to ensure that the task does not permanently prevent other system activities if it enters an infinite loop.

Because of the way that breakpoints are handled, the following restrictions apply to tasks debugged using ODT:

1. Tasks must be built in user mode. If the user attempts to debug an exec-mode task with ODT, the error message "EXEC MODE" is printed and ODT is reinitialized.

2. Tasks can have no overlays. If the user debugs a task with overlays using ODT, he should use extreme caution so that no breakpoints are set within a link.

3. Tasks cannot modify virtual address 17 (autoincrement register X17).

4. Tasks cannot modify virtual addresses 0 to 3.

5. Tasks cannot modify the instruction at a breakpoint.

The last four restrictions are not checked by ODT. If the user task violates these restrictions, system integrity is not compromised, but the user task will probably fail to run correctly.

22.1 INVOKING ODT

The user can invoke ODT by typing a command according to the format:

Form:	ODT name [LUN]∇
Where:	name of a CONSTRUCTed task image is a string of one to six .SIXBT characters LUN is an integer representing the logical unit number on which the file resides (the default value is decimal 14)
Example:	TDV>ODT SCAN 16 TDV>

22.2 DEBUGGING WITH ODT

When ODT begins execution, it prints a header message and attempts to fix the specified task in core. If the task can be fixed, ODT requests additional commands by typing the prompter:

```
ODT>
```

If the task cannot be fixed in its partition for any reason, ODT prints the message "FIX ERR" and exits.

In response to the ODT prompter, the user can enter any one of the following commands:

Command	Function
OPEN	Opens a register or memory location
START	Starts the task
EXIT	Exits and aborts the task
DEFINE	Defines a symbol
SET	Sets a breakpoint
REMOVE	Removes a breakpoint
CONTINUE	Continues task execution after reaching a breakpoint
RESTART	Restarts the task at some location
RELOAD	Reloads the task with symbols and breakpoints
DUMP*	Dumps the task partition into a created file on disk
DECODE*	Decodes the opened memory locations or stops decoding the opened memory locations
REGISTERS*	Prints all registers when the task has reached a breakpoint

Each of these commands is described in detail in the following paragraphs.

---

* Legal only if ONEPLS was defined

22.2.1 OPEN

The OPEN command examines and optionally modifies the contents of task registers or locations within the task partition. The format of this command is:

```
ODT>OPE[N] nnnnnn
```

where nnnnnn is an address relative to the base of the task partition. The nnnnnn specification can be a symbol or one of the following registers: $AC, $MQ, $XR, $LR, $LINK or $X10 TO $X16.

ODT prints the contents of the location or register and waits for a response. If a value (octal only) is then given as a response, that value replaces the former contents. If only a terminator is typed, ODT does not modify the current contents of the register or location. If the response is terminated with an altmode, a new command is requested. If the response is terminated with a carriage return, the next location is opened (unless a register was specified). If a register was specified, a new command is always requested.

If a new value for the link is to be entered, only bit 17 of the octal word typed is relevant.

This command is legal any time, but registers can be opened only if the task is at a breakpoint.

22.2.2 START

The START command begins execution of the task being debugged. The format of this command is:

```
ODT>STA[RT]
```

This command can be used only if the task has not already been started.

22.2.3 EXIT

The EXIT command terminates the debugging session for a particular task. The format of this command is:

```
ODT>EXI[T]
```

The EXIT command causes the task to abort. This command is legal any time.

22.2.4 DEFINE

The DEFINE command defines a symbol. Once a symbol has been defined, it can replace program addresses indicated in octal notation in other command string inputs. The format of this command is:

```
ODT>DEF[INE]    xxx=nnnnnn
```

where xxx is a one- to three-character symbol and nnnnnn is any octal number. After the first three characters of a symbol are input, subsequent characters are ignored. Unlike other commands, nnnnnn for this command cannot be a symbol.

This command is legal any time.

22.2.5 SET

The SET command enters a breakpoint into the task at some location specified. When the executing task reaches a breakpoint, its progress is stopped prior to the execution of the instruction at which the breakpoint is SET. The format of this command is:

```
ODT>SET nnnnnn
```

- where nnnnnn is an address relative to the base of the task partition, and can be a symbol.

This command is legal any time.

22.2.6 REMOVE

The REMOVE command removes a breakpoint from the task. It complements the SET command. The format of this command is:

```
ODT>REM[OVE] nnnnnn
```

- where nnnnnn is an address relative to the base of the task partition, and can be a symbol.

This command is legal any time.

22.2.7 RESTART

The RESTART command causes the task to be started at the location specified once that task has reached a breakpoint. The format of this command is:

```
ODT>RES[TART] nnnnnn
```

where nnnnnn is an address relative to the base of the task partition, and can be a symbol.

This command is legal only if the task is at a breakpoint.

22.2.8 RELOAD

The RELOAD command instructs ODT to reload the task image (i.e., refresh the task partition with a new copy of the task). Breakpoints and symbols defined when this command is issued are retained, but the contents of user-modified locations are lost. The format of this command is:

```
ODT>REL[OAD]
```

This command is legal any time.

22.2.9 CONTINUE

The CONTINUE command is used to resume task execution once a breakpoint has been reached. The format of this command is:

```
ODT>CON[TINUE]
```

This command is legal only if the task is at a breakpoint.

22.2.10 REGISTERS

The REGISTERS command is legal only if the assembly parameter ONEPLS was defined. The format of this command is:

```
ODT>REG[ISTERS]
```

This command prints the contents of the AC, MQ, XR, LR and link.

This command is legal only if the task is at a breakpoint.

22.2.11 DECODE

The DECODE command sets a flag to tell the OPEN command whether it should print the opcodes of opened locations as well as their contents. The format of this command is:

```
ODT>DEC[ODE] string
```

Where string is either "ON" or "OFF"

If string is "ON", the opcodes are decoded. If string is "OFF", the opcodes are not decoded. (No opcode decoding is the default mode of operation for the OPEN command.) The DECODE command is valid only if ONEPLS was defined when ODT was assembled.

This command is legal any time.

22.2.12 DUMP

The DUMP command is used to write an image of the task partition into an RSX-created file on disk. The file is named TSKNAM DMP, where TSKNAM is the task name. The image of the task does not contain breakpoints, but all breakpoints are restored when the dumping process is complete. This command is valid only if ONEPLS was defined when ODT was assembled. The format of this command is:

```
ODT>DUM[P]
```

This command is legal any time.

22.3 MONITORING TASK PROGRESS

Whenever a task has been started, restarted or continued after a breakpoint, ODT monitors its progress. If the task does not exit or reach a breakpoint within several seconds, ODT prints the task ATL status and asks the user whether the task should be aborted. If the user types "NO", ODT resumes monitoring task progress. If the user types "YES", ODT reloads the task, aborting it in the process. If the task exits while ODT is monitoring its progress, the message "TASK HAS EXITED" is printed. The task is then reloaded.

22.4 AN EXAMPLE OF ODT USE

The following listing gives an example of a task to be debugged with ODT. This program contains two errors that will be found and subsequently corrected using ODT. The program consists of a main section and a single subroutine. The function of the main section is to simulate input data and call the subroutine. The subroutine, which is presumably used in another program, is intended to count the bits that are set in the accumulator on entry to the subroutine. It is this subroutine that contains errors and can be debugged using ODT.

When the example program is task built, it is relocated so that location 0 in the listing corresponds to the base address of the partition for this task plus 20 (octal).

The example program is listed on the following page:

```
 1                                      /
 2                                      / THIS PROGRAM IS INTENDED TO TEST THE SUBROUTINE 'ONECNT'
 3                                      / WHICH COUNTS THE 1 BITS IN THE AC ON ENTRY TO THE
 4                                      / SUBROUTINE.
 5                                      /
 6                                      / AS GIVEN HERE, THIS SUBROUTINE HAS TWO ERRORS.
 7                                      /
 8                 440000 A     IDX=ISZ                          /USED IF INTENT IS TO INCREMENT ONLY
 9                 000010 A     X10=10                           /AUTOINCREMENT REGISTER 10
10                                      /
11          00000 R 200053 R     START   LAC     (BUFF-1          /INITIALIZE X10
12          00001 R 060054 R             DAC*    (X10             /USE X10 AS A BUFFER POINTER FOR
13                                                                /STORING RESULTS.
14          00002 R 200055 R             LAC     (654321          /GET 1ST DATA WORD
15          00003 R 100013 R             JMS     ONECNT           /GO COUNT THE ONE'S
16          00004 R 200056 R             LAC     (123456          /GET 2ND DATA WORD
17          00005 R 100013 R             JMS     ONECNT           /GO COUNT THE ONE'S
18          00006 R 777777 A             LAW     -1               /GET 3RD DATA WORD
19          00007 R 100013 R             JMS     ONECNT           /GO COUNT THE ONE'S
20          00010 R 750000 A             CLA                      /GET 4TH DATA WORD
21          00011 R 100013 R             JMS     ONECNT           /GO COUNT THE ONE'S
22          00012 R 000054 R             CAL     (10              /EXIT
23                                      /
24                                      / SUBROUTINE ONECNT -- COUNT THE ONE BITS IN THE AC ON ENTRY,
25                                      / STORE RESULTS VIA X10
26                                      /
27          00013 R 000000 A     ONECNT  0
28          00014 R 652000 A             LMQ                      /SAVE THE DATA WORD IN MQ
29          00015 R 777756 A             LAW     -22              /SET UP A BIT COUNTER
30          00016 R 040051 R             DAC     BITCNT
31          00017 R 744000 A             CLL                      /CLEAR THE LINK
32          00020 R 641002 A             LACQ                     /RETRIEVE DATA WORD
33          00021 R 740010 A     LOOP    RAL                      /SHIFT BITS AND TABULATE ONE'S
34          00022 R 740400 A             SNL
35          00023 R 440052 R             IDX     ONES
36          00024 R 440051 R             ISZ     BITCNT           /DONE WITH WORD?
37          00025 R 600021 R             JMP     LOOP             /NO -- CONTINUE
38          00026 R 200052 R             LAC     ONES             /YES -- STORE RESULT
39          00027 R 060010 A             DAC*    X10
40          00030 R 620013 R             JMP*    ONECNT           /RETURN
41                                      /
42          00031 R         A    BUFF    .BLOCK  20               /RESULTS BUFFER
43          00051 R 000000 A     BITCNT  0                        /BIT COUNTER
44          00052 R 000000 A     ONES    0                        /ONE'S COUNTER
45                 000000 A             .END
            00053 R 000030 R *L
            00054 R 000010 A *L
            00055 R 654321 A *L
            00056 R 123456 A *L
                SIZE=00057     NO ERROR LINES
```

The following terminal output shows how ODT can be used to debug the subroutine in the example program:

```
TDV>ODT TEST                    ←Name program to be debugged.

ODT V1B000
ODT>DEF BUF=31                  ⎫
ODT>DEF BIT=51                  ⎬   Define symbols for later use
ODT>DEF ONE=52                  ⎭   in command inputs.
ODT>SET 24                      ⎫
ODT>SET 26                      ⎬   Set breakpoints at all
ODT>SET 30                      ⎬   subroutine return addresses.
ODT>SET 32                      ⎭
ODT>START                       ←Start execution of the task.
BRKPT AT 000024                 ←A breakpoint has been reached at
                                 location 4 (L1).
ODT>OPEN BUF                    ⎫
000051 :000011 :                ⎬   Open those locations that show
000052 :101550 :                ⎬   the results of the subroutine.
000053 :600411 :                ⎭   The first result is correct.
ODT>OPEN ONE                    ⎫
000072 :000011 :                ⎬   Open subroutine temporary
ODT>OPEN BIT                    ⎬   variables to check them. They
000071 :000000 :                ⎭   are OK so far.
ODT>CONT                        ←Continue task execution.
BRKPT AT 000026                 ←A breakpoint has been reached at
                                 location 6 (L2).
ODT>OPEN BUF                    ⎫
000051 :000011 :                ⎬   Check the results buffer and
000052 :000022 :                ⎬   subroutine temporary storage
000053 :600411 :                ⎬   locations. An error has been
ODT>OPEN ONE                    ⎬   found. The temporary storage
000072 :000022 :                ⎬   location ONES must be zeroed
                                ⎭   before counting.
ODT>RELOAD                      ←Reload the task into memory
                                 keeping symbols and breakpoints.
ODT>OPEN 37                     ←Change location 17 from a CLL to
000037 :744000 :140072           a DZM ONES.
ODT>START                       ←Start task execution.
BRKPT AT 000024                 ←The first breakpoint has been
                                 reached.
ODT>OPEN BUF                    ⎫
000051 :000011 :                ⎬
000052 :101550 :                ⎬   Check results. They look OK.
000053 :600411 :                ⎬
ODT>OPEN ONE                    ⎬
000072 :000011 :                ⎭
ODT>CONT                        ←Continue task execution.
BRKPT AT 000026                 ←The next breakpoint has been
                                 reached.
ODT>OPEN BUF                    ⎫
000051 :000011 :                ⎬
000052 :000011 :                ⎬   Check results again. OK.
000053 :600411 :                ⎬
ODT>OPEN ONE                    ⎬
000072 :000011 :                ⎭
ODT>CONT                        ←Continue.
BRKPT AT 000030                 ←Another breakpoint has been
                                 reached.
```

```
ODT>OPEN BUF
000051 :000011 :
000052 :000011 :
000053 :000000 :
ODT>RELOAD
ODT>OPEN 42
000042 :740400 :741400
ODT>OPEN 37
000037 :744000 :140072
ODT>START
BRKPT AT 000024
ODT>OPEN BUF
000051 :000011 :
000052 :101550 :
000053 :600411 :
ODT>CONT
BRKPT AT 000026
ODT>OPEN BUF
000051 :000011 :
000052 :000011 :
000053 :600411 :
ODT>CONT
BRKPT AT 000030
ODT>OPEN BUF
000051 :000011 :
000052 :000011 :
000053 :000022 :
000054 :101635 :
000055 :204217 :
ODT>CONT
BRKPT AT 000032
ODT>OPEN BUF
000051 :000011 :
000052 :000011 :
000053 :000022 :
000054 :000000 :
000055 :204217 :
ODT>OPEN $X10
777770 :000054 :
ODT>EXIT

TDV>
```

Check results again. Another error has been found. The subroutine is counting cleared bits instead of set bits.

←Reload the task once again.

Change location 22 from an SNL to an SZL instruction.

←Enter previous correction.

←Start task execution and check the results at each breakpoint as they are reached. Use the same procedures as above.

←The final breakpoint has been reached and the subroutine output looks fine.

←Examine autoincrement register 10 to be sure that it points to the correct location within the data buffer. It does, so leave ODT.

22.5 ERROR MESSAGES

Table 22-1 lists possible error messages and their implications.

Table 22-1
ODT Error Messages

Message	Meaning	Routines Generating Error Message
HINF ERR	HINF error on LUN-14	DUMP
PUT ERR	ODT is unable to write the task image to disk	DUMP
CREATE ERR	ODT is unable to CREATE a file on LUN-14	DUMP
ILLEGAL	Registers cannot be opened unless the task is at a breakpoint	OPEN
BRDPT AT XCT	Illegal to have a breakpoint at XCT instruction	SET, OPEN
EXEC MODE*	Task is in exec mode	ODT INITIALIZATION
READ ERR*	Terminal read error	ODT INITIALIZATION, COMMAND DISPATCH, OPEN, MONITOR ROUTINE
FORMAT ERR	Violation of command syntax	ODT INITIALIZATION, COMMAND DISPATCH, RESTART, SET, REMOVE, OPEN, DECODE
FIX ERR*	ODT is unable to FIX the task	ODT INITIALIZATION,
TASK NOT IN STL*	Task specified is not in the system**	ODT INITIALIZATION
WHAT?	Unrecognized command	COMMAND DISPATCH
ALREADY STARTED	Task was previously started	START
REQUEST ERR*	ODT is unable to REQUEST the task	START

(Continued)

*Causes ODT to exit.

**This error should never appear unless the system list structure has been corrupted.

Table 22-1 (Cont.)
ODT Error Messages

Message	Meaning	Routines Generating Error Message
TABLE FULL	Breakpoint or symbol table full	DEFINE, SET
ALREADY DEFINED	Breakpoint has already been defined or set	SET
NOT AT A BRKPT	Task is not at a breakpoint	RESTART, CONTINUE, REGISTERS
NOT IN TABLE	Breakpoint is not in the table	CONTINUE, REMOVE
ATTACH ERR*	ODT is unable to ATTACH the terminal	ODT INITIALIZATION, COMMAND DISPATCH, MONITOR
TASK NOT IN ATL*	Task is deleted from ATL	ODT INITIALIZATION, CONTINUE
OUT OF BOUNDS	Address is not within the task partition space	RESTART, SET, REMOVE, OPEN

*Causes ODT to exit.

STA

CHAPTER 23

STATUS: MULTIACCESS STATUS REPORT

The STATUS TDV Function task prints a status report of internal MULTIACCESS information. This status information is intended for use by the system manager and, in general, is not of interest to the TDV user.

23.1 INVOKING STATUS

The user can invoke STATUS by typing a command according to the format:

Form:	STA[TUS]V
Example:	TDV>STA TDV>

23.2 INPUT/OUTPUT AND TASK BUILDING

The STATUS TDV Function task is assigned task name STA... at task-building time. This task is written in FORTRAN and should be built to run in user mode (protected and relocated). Output status information is sent to LUN-16.

A sample STATUS report is listed below:

```
TDV>STA
MULTIACCESS STATUS REPORT  13-JUL-76   0:13 HOURS
 1 ACTIVE USER(S)
 TOTAL NUMBER OF USER(S) =     1
 TOTAL NUMBER OF JOB(S) =    15
 MEAN WAIT TIME IS  0.00 SECONDS/JOB
 ACTIVE TERMINAL NUMBERS ARE:  0
 THERE ARE  198 (10) WORDS OF DYNAMIC STORAGE
                TASK SIZE HISTOGRAM
 000000-001777    0    002000-003777    3    004000-005777    2
 006000-007777    0    010000-011777    0    012000-013777    1
 014000-015777    3    016000-017777    0    020000-021777    4
 022000-023777    0    024000-025777    0    026000-027777    0
 030000-031777    0    032000-033777    0    034000-035777    0
 036000-037777    0    040000-041777    0    042000-043777    0
 044000-045777    0    046000-047777    0    050000-051777    0
 052000-053777    0    054000-055777    0    056000-057777    0
 060000-061777    0    062000-063777    0    064000-065777    0
 066000-067777    0    070000-071777    0    072000-073777    0
 074000-075777    0    076000-077777    0    100000-777777    0
TDV>
```

The status report presents important current information and historical information compiled since TDV started running. Current information includes:

- Number of active users
- Active terminal numbers
- Words of dynamic storage (this system management concept is fully documented in the TDV code listing)

Historical information includes:

- Total number of different users
- Total number of jobs handled
- Mean wait time per job
- Number of tasks that have been run in each of 33 size ranges, each progressively increasing by 2000 (octal) words

ACI
ACD

CHAPTER 24

BATCH ACCOUNT FILE INITIALIZATION AND DISPLAY

Two TDV function tasks, ACI and ACD, control batch job accounting and account summaries. The use of ACI and ACD is described in Part VIII of this manual, Batch Processing.

24.1 ACI: INITIALIZE BATCH ACCOUNT FILE

ACI allows the system manager to initialize (clear to zero) the Batch processor account file or create an account file if one does not exist.

24.2 ACD: DISPLAY BATCH ACCOUNT FILE

ACD allows the system manager to list the Batch Processor account file.

INDEX