4. The user should also take care that the USR is never swapped over any of the following areas: the program stack; any parameter block for calls to the USR; any I/O buffers, device handlers, or completion routines being used when the USR is called.

For example:

```
.TITLE  USR.MAC
;THIS IS AN EXAMPLE OF THE WAY A BACKGROUND PROGRAM CAN AVOID
;UNNECESSARY USR SWAPPING.
        .MCALL  .SETTOP,.EXIT,.GVAL
USRLOC  = 266                   ;POINTER TO USR LOCATION IS
                                ;AT 266 BYTES INTO RMON.
START:
        .GVAL   #AREA,#USRLOC   ;R0 => USR
        TST     -(R0)           ;POINT JUST BELOW
        CMP     R0,#50          ;DOES USR SWAP OVER US?
        BHI     1$              ;NO, OK
        MOV     #-2,R0          ;YES, USR MUST SWAP
1$:     .SETTOP                 ;ASK FOR MEMORY UP TO USR
        MOV     R0,HILIM        ;R0 = HIGH LIMIT OF MEMORY
                                ;ACTUALLY GRANTED BY MONITOR.
        .EXIT
HILIM:  .WORD   0               ;CONTAINS HI LIMIT OF MEMORY
AREA:   .BLKW   2               ;EMT ARGUMENT BLOCK
        .END    START
```

2.2.6 Offset Words

There are several words that always have fixed positions relative to the start of the resident monitor. It is often advantageous for user programs to be able to access these words. This is done using the .GVAL programmed request in the following form:

```
.GVAL  #area,#offse
```

Here, area is a two-word argument block and offse is a number from the following list.

OFFSET (Bytes)	Contents
266	Start of normal USR area. This is where the USR resides when it is called into memory and location 46 is 0. It is useful to be able to perform a .SETTOP in a background job so that the USR does not swap, and once called in, remains resident. (An example is in Section 2.2.5.)
270	Address of I/O exit routine for all devices. The exit routine is an internal queue management routine through which all device handlers exit once the I/O transfer is complete. Any new device handlers added to RT-11 must also use this exit location.

OFFSET (Bytes)	Contents
272	Special device error word. This word is used by non-RT-11 file structured devices (such as MT and CT) to report errors to the monitor.
275	Unit number of system device (device from which system was last bootstrapped).
276	Monitor version number. The user can always access the version number to determine if the most recent monitor is in use. For RT-11 Version 3B, this value is 3.
277	Monitor release level. This number identifies the release level of the monitor version specified in byte 276. For version 3B, the value is 2.
300	Configuration word. This is a string of 16 bits that indicates information about either the hardware configuration of the system or a software condition. Another configuration word is available at offset 370 that contains additional data. The bits and their meanings are:

Bit #	Meaning
0	0 = SJ Monitor 1 = FB Monitor if bit 12 is not set, XM monitor if bit 12 is set
2	1 = graphics display hardware exists (VT11 or VS60)
3	1 = RT-11 BATCH is in control of the background
5	0 = 60-cycle clock 1 = 50-cycle clock
6	1 = FP11 floating-point hardware exists
7	0 = No foreground job is in memory 1 = Foreground job is in memory
8	1 = User is linked to the graphics scroller
9	1 = USR is permanently resident (via a SET USR NOSWAP - USR is always resident in XM)
11	1 = Processor is a PDP-11/03 (that is, there is no program status word on this processor)
12	1 = a mapped system running under XM monitor
13	1 = The system clock has a status register
14	1 = KW11-P clock exists and can be used by programs
15	1 = either an L clock or a P clock (depending on the system generation procedure used) is present

The other bits are reserved for future use and should not be used by user programs.

OFFSET (Bytes)	Contents
304-313	These locations contain the addresses of the console terminal control and status registers (but they are not used when the multi-terminal option is present). The order is: 304 Keyboard status 306 Keyboard buffer 310 Printer status 312 Printer buffer These locations can be changed, for example, to reflect a second terminal; thus RT-11 can be made to run on any terminal connected to the machine through the DL11 terminal interface.
314	The maximum file size allowed in a 0 length .ENTER. This can be adjusted by the user program or by using the PATCH program to be any reasonable value. The default value is 177777 (octal) blocks, allowing an essentially unlimited file size.
324	Address of .SYNCH entry. User interrupt routines can enter the monitor through this address to synchronize with the job they are servicing.
354	Address of VT11 or VS60 display processor display stop interrupt vector.
360	Move to PS routine. The routine is called by the .MTPS macro to do processor independent moves to the program status word.
362	Move from PS routine. The routine is called by the .MFPS macro to do processor independent moves from the program status word.
366	Indirect file and command language state word.
370	Extension configuration word. This is a string of 16 bits used to indicate the presence of an additional set of hardware options on the system. The bits and their meanings are: Bit # — Meaning 0 — 1 = cache memory is present 1 — 1 = parity memory is present 2 — 1 = readable switch register is present 3 — 1 = writeable console display register is present 8 — 1 = EIS option is present

OFFSET (Bytes)	Contents

Bit #	Meaning
9	0 = VT11 display hardware exists if bit 2 at offset 300 is set
	1 = VS60 display hardware exists if bit 2 at offset 300 is set
14	1 = processor is PDP-11/70
15	1 = processor is PDP-11/60

The other bits are reserved for future use and should not be used by user programs.

372 SYSGEN options word. The bit pattern indicates important SYSGEN options and must not be modified by user programs or patches. The bits and their meanings are:

Bit #	Meaning
0	1 = error logging option is present
1	1 = memory management option is present
2	1 = device I/O time-out option is present
9	1 = memory parity option is present
10	1 = SJ mark time option is present
11-12	00 = no escape sequences recognized
	01 = option to recognize DIGITAL escape sequences is present
	10 = option to recognize ANSI escape sequences is present
13	1 = multi-terminal option is present

The other bits are reserved for future use and should not be used by user programs.

374 Size of USR in bytes. Programs should use this information to dynamically determine the size of the region needed to swap the USR.

377 Depth of nesting of indirect files (default is 3). This value must be referred to as a byte. It can be patched or set by programs to change the nesting depth of indirect files.

400 Internal offset for use by BATCH only.

402 Byte offset to fork request processor from start of resident monitor (contents of 54).

2.2.7 File Structure

RT-11 uses a contiguous file structure. This type of structure requires that every file on a device be made up of a contiguous group of physical blocks. Thus, a file that is 19 blocks long occupies 19 contiguous blocks on the device.

A contiguous area on a device can be in one of the following categories:

1. Permanent file. This is a file that has been created with .ENTER and then .CLOSEd. Any named files that appear in a directory listing are permanent files.

2. Tentative file. Any file that has been created with .ENTER but not .CLOSEd is a tentative file entry. When the .CLOSE request is given, the tentative entry becomes a permanent file. If a permanent file already exists under the same name, the old file is deleted when the tentative file is .CLOSEd. If a .CLOSE is never given, the tentative file is treated like an empty entry. The tentative file is deleted when a new tentative file with the same name is created.

3. Empty entry. When disk space is unused or a permanent file is deleted, an empty entry is created. Empty entries appear in a full directory listing as <UNUSED> n, where n is the decimal block length of the empty area.

Since a contiguous structure does not automatically consolidate unused disk space, a device can eventually become fragmented with many scattered empty entries. RT-11 has a SQUEEZE command to collect all empty areas and create a single empty entry at the end of a device.

2.2.8 Completion Routines

Completion routines are user-written routines that are entered following the completion of some external operation. A completion routine can be entered after an I/O data transfer, after some number of clock ticks or after a user-specified interrupt. On entry to an I/O completion routine, R0 contains the contents of the channel status word for the operation and R1 contains the octal channel number of the operation. The carry bit is not significant.

Completion routines are handled differently in the SJ and the FB or XM versions of RT-11. In the SJ version, completion routines are totally asynchronous and can interrupt one another. In FB and XM, completion routines do not interrupt each other. Instead they are queued and made to wait until the correct job is running. For example, if a foreground job is running and an I/O transfer initiated by a background job completes with a specified completion routine, the background routine is queued and does not execute until the foreground gives up control of the system. If the foreground is running and a foreground I/O transfer completes and wants a completion routine, that routine is entered immediately if the foreground is not already executing a completion routine. If it is in a completion routine, that routine continues to termination, at which point any other completion routines are entered in a first in/first out order. If the background is running and a foreground I/O transfer completes with a specified completion routine, the background is suspended and the foreground routine is entered immediately.

The restrictions that must be observed when writing completion routines are:

1. Completion functions cannot issue a request that would cause the USR to be swapped in. They are primarily used for issuing .READ and .WRITE requests, not for opening or closing files, etc. A fatal monitor error is generated if the USR is called from a completion routine.

2. Completion routines should never reside in the memory space that is used for the USR, since the USR can be interrupted when I/O terminates and the completion routine is entered. If the USR has overlaid the routine, control passes to a random place in the USR, with a HALT or error trap the likely result.

3. The routine must be exited with an RTS PC (because it is called from the monitor with a JSR PC,ADDR, where ADDR is the user-supplied entry point address).

4. If a completion routine uses registers other than R0 or R1, it must save them upon entry and restore them before exiting. Other requests cannot transfer data between the mainline program and the completion routine.

5. In XM, completion routines must remain mapped while the request is active and the routine can be called.

2.2.9 Using the System Macro Library

User programs for RT-11 should always be written using the macros provided in the system macro library (SYSMAC.MAC), supplied with RT-11. This ensures source level compatibility among all user programs and allows easy modification by redefining a macro. A listing of SYSMAC.MAC appears in Appendix B.

Suggestions for writing foreground programs are in Chapter 1 (FB Programming and Device Handlers). Chapter 1 should be read in conjunction with Chapter 2 before coding FB programs.

2.2.10 Error Reporting

In processing a programmed request, the monitor can detect an error condition that must be reported to the user program. RT-11 programmed requests use three methods of reporting these errors: the carry (C) bit, the error byte (byte 52 in the system communication area), and the monitor error message. The carry bit is returned clear after normal termination of a programmed request, and set after an abnormal termination. Almost all requests should be followed by a BCS or BCC instruction to detect a possible error. When the carry bit is set, the error byte usually contains additional information about the error. The meanings of values in the error byte are described individually for each request. In most cases, the user program should test the error byte when the carry bit is set. The values contained in the error byte are not significant when the carry bit is clear. Certain serious or non-recoverable error conditions cause a monitor error message to be printed at the console terminal. A user-program can use the .SERR programmed request to cause these errors to be reported through the carry bit and the error byte, in which case the error byte will contain a negative error code.

2.3 TYPES OF PROGRAMMED REQUESTS

There are three types of services that the monitor makes available to the user through programmed requests. These are:

1. Requests for file manipulation
2. Requests for data transfer
3. Requests for miscellaneous services

Table 2-1 summarizes the programmed requests in each of these categories alphabetically. Some requests function only in a FB and XM environment and are ignored under the SJ monitor. The EMT and function code for each request (where applicable) are shown in octal. It should be noted as a general rule that only six characters (such as .CHCOP) are significant to the MACRO assembler. Longer forms are shown for readability only.

Table 2-1
Summary of Programmed Requests

Mnemonic	EMT Code	Purpose
File Manipulation Requests		
.CHCOPY*	375 13	Establishes a link and allows one job to access another job's channel.
.CLOSE	374 6	Closes the specified channel.
.DELETE	375 0	Deletes the file from the specified device.
.ENTER	375 2	Creates a new file for output.
.LOOKUP	375 1	Opens an existing file for input and/or output via the specified channel.
.PURGE	374 3	Clears out a channel.
.RENAME	375 4	Changes the name of the indicated file to a new name. If this request is attempted with magtape, the handler returns an illegal operation code.
.REOPEN	375 6	Restores the parameters stored via a .SAVESTATUS request and reopens the channel for I/O.
.SAVESTATUS	375 5	Saves the status parameters of an open file in user memory and frees the channel for future use.

(continued on next page)

Table 2-1 (Cont.)
Summary of Programmed Requests

Mnemonic	EMT Code	Purpose
Data Transfer Requests		
.MTIN*	375 37	Operates as a .TTYIN for multi-terminal configuration.
.MTOUT*	375 37	Operates as a .TTYOUT for multi-terminal configuration.
.MTPRNT*	375 37	Operates as a .PRINT request for a multi-terminal configuration.
.PRINT	351 --	Outputs an ASCII string terminated by a 0 byte or a 200 byte.
.RCVD* .RCVDW* .RCVDC*	375 26	Receives data. Allows a job to read messages or data sent by another job in an FB environment. The three modes correspond to the READ, .READC, and .READW modes.
.READ	375 10	Transfers data on the specified channel to a memory buffer and returns control to the user program when the transfer request is entered in the I/O queue. No special action is taken upon completion of I/O.
.READC	375 10	Transfers data on the specified channel to a memory buffer and returns control to the user program when the transfer request is entered in the I/O queue. Upon completion of the read, control transfers asynchronously to the routine specified in the .READC request.
.READW	375 10	Transfers data via the specified channel to a memory buffer and returns control to the user program only after the transfer is complete.
.SDAT* .SDATC* .SDATW*	375 25	Allows the user to send messages or data to the other job in an FB environment. The three modes correspond to the .WRITE, .WRITC, and .WRITW modes.
.SPFUN	375 32	Performs special functions on magtape, cassette, diskette and some disk devices.
.TTYIN .TTINR	340 --	Transfers one character from the keyboard buffer to R0.
.TTYOUT .TTOUTR	341 --	Transfers one character from R0 to the terminal input buffer.

(continued on next page)

Table 2-1 (Cont.)
Summary of Programmed Requests

Mnemonic	EMT Code	Purpose
.WRITE	375 11	Transfers data on the specified channel to a device and returns control to the user program when the transfer request is entered in the I/O queue. No special action is taken upon completion of the I/O.
.WRITC	375 11	Transfers data on the specified channel to a device and returns control to the user program when the transfer request is entered in the I/O queue. Upon completion of the write, control transfers asynchronously to the routine specified in the .WRITC request.
.WRITW	375 11	Transfers data on the specified channel to a device and returns control to the user program only after the transfer is complete.
Miscellaneous Services		
.CDFN	375 15	Defines additional channels for I/O.
.CHAIN	374 10	Chains to another program (in the background job only).
.CRAW**	375 36	Creates a window in virtual memory.
.CRRG**	375 36	Creates a region in extended memory.
.CMKT	375 23	Cancels an unexpired mark time request.
.CNTXSW*	375 33	Requests that the indicated memory locations be part of the FB context switch process.
.CSIGEN	344 --	Calls the Command String Interpreter (CSI) in general mode.
.CSISPC	345 --	Calls the CSI in special mode.
.CSTAT*	375 27	Returns the status of the channel indicated.
.DATE	374 12	Moves the current date information into R0.
.DEVICE*	375 14	Allows the user to disable device interrupts in FB upon program termination.
.DSTATUS	342 --	Returns the status of a particular device.

(continued on next page)

Table 2-1 (Cont.)
Summary of Programmed Requests

Mnemonic	EMT	Code	Purpose
.ELAW**	375	36	Cancels an address window in virtual memory.
.ELRG**	375	36	Cancels an allocated region in extended memory.
.EXIT	350	--	Exits the user program.
.FETCH	343	--	Loads device handlers into memory.
.GMCX**	375	36	Returns mapping status of a specified window.
.GTIM	375	21	Gets time of day.
.GTJB	375	20	Gets parameters of the current job.
.GTLIN	345	--	Accepts an input line from either an indirect file or from the console terminal.
.GVAL	375	34	Returns monitor fixed offsets in a pseudo-protected manner.
.HERR	374	5	Specifies termination of the job on fatal errors.
.HRESET	357	--	Terminates I/O transfers and does a .SRESET operation.
.INTEN	---	--	Notifies the monitor that an interrupt has occurred, requests system state and sets the processor priority to the correct value.
.LOCK	346	--	Makes the monitor User Service Routines (USR) permanently resident until .EXIT or .UNLOCK is executed. The user program is swapped out, if necessary.
.MAP*	375	36	Maps a virtual address window to extended memory.
.MFPS	---	--	Reads the priority bits in the processor status word (but does not read the condition codes).
.MRKT	375	22	Marks time; that is, sets an asynchronous routine to occur after a specified interval.
.MTATCH*	375	37	Attaches a terminal for exclusive use by the requesting job.

(continued on next page)

Table 2-1 (Cont.)
Summary of Programmed Requests

Mnemonic	EMT Code	Purpose
.MTDTCH*	375 37	Detaches a terminal from one job and frees it to be used by other jobs.
.MTGET*	375 37	Returns status of specified terminal to caller.
.MTSET*	375 37	Determines and modifies terminal status in a multi-terminal configuration.
.MTPS	--- --	Sets the priority bits, condition codes, and T bit in the processor status word.
.MTRCTO*	375 37	Resets the CTRL/O flag for the designated terminal.
.MWAIT*	374 11	Waits for messages to be processed.
.PROTECT*	375 31	Requests that vectors in the area from 0-476 be given exclusively to the current job.
.QSET	353 --	Increases the size of the monitor I/O queue.
.RCTRLO	355 --	Enables output to the terminal.
.RELEAS	343 --	Removes device handlers from memory.
.RSUM*	374 2	Causes the main line of the job to be resumed when it was suspended with .SPND.
.SCCA	375 35	Enables intercept of CTRL/C commands.
.SERR	374 4	Inhibits most fatal errors from aborting the current job.
.SETTOP	354 --	Specifies the highest memory location to be used by the user program.
.SFPA	375 30	Sets user interrupt for floating point processor exceptions.
.SPND*	374 1	Causes the running job to be suspended.
.SRESET	352 --	Resets all channels and releases the device handlers from memory.
.SYNCH	--- --	Enables user program to perform monitor programmed requests from within an interrupt service routine.

(continued on next page)

Table 2-1 (Cont.)
Summary of Programmed Requests

Mnemonic	EMT	Code	Purpose
.TLOCK*	374	7	Indicates if the USR is currently being used by another job and performs a .LOCK if the USR is available.
.TRPSET	375	3	Sets a user intercept for traps to locations 4 and 10.
.TWAIT*	375	24	Suspends the running job for a specified amount of time.
.UNLOCK	347	--	Releases the USR if a .LOCK was done and swaps in the user program, if required.
.UNMAP*	375	36	Unmaps a virtual memory address window.
.UNPROTECT*	374	31	Cancels the .PROTECT vector protection request.
..V1..	---	--	Provides compatibility with version 1 format.
..V2..	---	--	Provides compatibility with version 2 format.
.WAIT	374	0	Waits for completion of all I/O on a specified channel.

*FB and XM monitors

**XM monitor only

Requests requiring the USR (as explained in Section 2.2.5) differ between the SJ and FB monitors. Table 2-2 indicates which requests require the USR to be in memory. The .CLOSE request on non-file-structured devices (LP:, PC:, TT:, etc.) does not require the USR under any monitor.

The USR is not reentrant and cannot be shared by concurrent jobs. When the USR is in use by one job, another job requiring it must queue up for it. This is particularly important for concurrent jobs when devices such as magnetic tape or cassette are active.

For example, USR file operations on tape devices require a linear search of the tape. When a background job is running the USR, the foreground job is locked out until the tape operation is completed. Since that can take considerable time, the programmer should be aware of the problem. In the FB and XM monitors, the .TLOCK request (see Section 2.4.56) can be used by a foreground job to check USR availability.

Table 2-2
Requests Requiring the USR

Request	SJ	FB	XM
.CDFN	Yes*	No	No
.CHAIN	No	No	No
.CHCOPY	--	No	No
.CLOSE (see Note 1)	Yes	Yes	Yes
.CMKT	No	No	No
.CNTXSW	--	No	No
.CRAW	--	--	No
.CRRG	--	--	No
.CSIGEN	Yes	Yes	Yes
.CSISPC	Yes	Yes	Yes
.CSTAT	--	No	No
.DATE	No	No	No
.DELETE	Yes	Yes	Yes
.DEVICE	--	No	No
.DSTATUS	Yes	Yes	Yes
.ELAW	--	--	No
.ELRG	--	--	No
.ENTER	Yes	Yes	Yes
.EXIT	Yes	Yes	Yes
.FETCH	Yes	Yes	Yes
.GMCX	--	--	No
.GTIM	No	No	No
.GTJB	--	No	No
.GTLIN	Yes*	Yes	Yes
.GVAL	No	No	No
.HERR	No	No	No
.HRESET	Yes*	No	No
.INTEN	No	No	No
.LOCK (see Note 2)	Yes	Yes	Yes
.LOOKUP	Yes	Yes	Yes
.MAP	--	--	No
.MFPS	No	No	No
.MRKT	No	No	No
.MTATCH	--	No	No
.MTDTCH	--	No	No
.MTGET	--	No	No
.MTIN	--	No	No
.MTOUT	--	No	No
.MTPRNT	--	No	No
.MTPS	No	No	No
.MTRCTO	--	No	No
.MTSET	--	No	No
.MWAIT	--	No	No
.PRINT	No	No	No

* Those requests marked with an asterisk always require a fresh copy of the USR to be read in before they can be executed. When executing such a request, the USR must be read in from the system device even if there is a copy of the USR presently in memory.

Note 1: Only if channel was opened with .ENTER.

Note 2: Only if USR is in a swapping state.

Note 3: Only if USR is not in use by the other job.

(continued on next page)

Table 2-2 (Cont.)
Requests Requiring the USR

Request	SJ	FB	XM
.PROTECT	--	No	No
.PURGE	No	No	No
.QSET	Yes*	Yes*	Yes*
.RCTRLO	No	No	No
.RCVD/.RCVDC/.RCVDW	--	No	No
.READ/.READC/.READW	No	No	No
.RELEAS	Yes	Yes	Yes
.RENAME	Yes	Yes	Yes
.REOPEN	No	No	No
.RSUM/.SPND	--	No	No
.SAVESTATUS	No	No	No
.SCCA	No	No	No
.SDAT/.SDATC/.SDATW	--	No	No
.SERR	No	No	No
.SETTOP	No	No	No
.SFPA	No	No	No
.SPFUN	No	No	No
.SRESET	Yes*	No	No
.SYNCH	No	No	No
.TLOCK (see Note 3)	Yes	Yes	Yes
.TRPSET	No	No	No
.TTINR/.TTYIN	No	No	No
.TTOUTR/.TTYOUT	No	No	No
.TWAIT	--	No	No
.UNLOCK	No	No	No
.UNMAP	--	--	No
.UNPROTECT	--	No	No
.WAIT	No	No	No
.WRITE/.WRITC/.WRITW	No	No	No

* Those requests marked with an asterisk always require a fresh copy of the USR to be read in before they can be executed. When executing such a request, the USR must be read in from the system device even if there is a copy of the USR presently in memory.

Note 1: Only if channel was opened with .ENTER.

Note 2: Only if USR is in a swapping state.

Note 3: Only if USR is not in use by the other job.

2.3.1 System Macros

The following macros are included in the system macro library, but are not programmed requests because they do not generate an EMT instruction:

```
..V2..
..V1..
```

They can be used in the same manner as the other macro calls; their explanations follow.

..V1../..V2..

2.3.1.1 ..V1../..V2..

Any version 1 and/or version 2 program that uses system macros must specify the version format in which the macro calls are to be expanded. Assembly errors at macro calls result if the proper version designation is not made. In version 3B, ..V1.. and ..V2.. are unnecessary since the expansions are made automatically. The ..V1.. and ..V2.. macros are retained only for compatibility with earlier systems.

The ..V1.. macro call enables all macro expansions to occur in Version 1 format.

```
Macro Call:     .MCALL ..V1..
                ..V1..
```

This causes all macros in the program to be assembled in version 1 form and the symbol ...V1 to be set equal to 1. User programs should not use the ...V1 symbol.

To cause all macro expansions to occur in version 2 format, the ..V2.. macro call is used.

```
Macro Call:     .MCALL ..V2..
                ..V2..
```

The ..V2.. macro causes the ...V1 symbol to equal 2. As with the V1 case, user programs should not use the ...V2 symbol.

Run-time or assembly errors can occur if both the ..V1.. and ..V2.. macro calls are used in a program.

All examples in this chapter illustrate the format for version 3 and later systems.

NOTE

It is possible for user programs to exist in which version 1 and version 2 or 3 macros are present. This is allowable by invoking the ..V1.. macro and using those macros that have no version 1 counterpart as if the ..V1.. macro had not been used.

This causes all macros that existed in version 1 to assemble in version 1 format, while those macros new to version 2 or version 3 are correctly generated as version 2 or version 3 macros. Note that in this case a macro that existed in version 1 (such as .READ) will expand in the version 1 format.

2.4 PROGRAMMED REQUEST USAGE

This section presents the programmed requests alphabetically and describes each one in detail. The following parameters are commonly used as arguments in the various calls:

addr — an address, the meaning of which depends on the request being used.

area — a pointer to the EMT argument list (for those requests that require a list) -- see Section 2.2.3.

blk — a block number specifying the relative block in a file where an I/O transfer is to begin.

buf — a buffer address specifying a memory location into which or from which an I/O transfer is to be performed; this address has to be word-aligned, i.e., an even address and not a byte or odd address.

cblk — the address of the five-word block where channel status information is stored.

chan — a channel number in the range 0-377(octal).

chrcnt — a character count in the range 1-255 (decimal).

code — a flag used to indicate whether the code in an area form (EMT 375) of a programmed request is to be set.

crtn — the entry point of a completion routine -- see Section 2.2.8.

dblk — the address of a four-word Radix-50 descriptor of the file to be operated upon -- see Section 2.2.2.

func — a numerical code indicating the function to be performed.

num — a number, the value of which depends on the request.

seqnum — file number -- for cassette operations if this argument is blank, a value of 0 is assumed.

For magtape operation, it describes a file sequence number that can have the following values:

Value	Meaning
0	For .LOOKUP, this value rewinds the magtape and spaces forward until the file name is found. For .ENTER it rewinds the magtape and spaces forward until the file name is found or until the logical end of tape is detected. If the file name is found, an error return is taken.

n	Where n is any positive number. This value positions the magtape at file sequence number n. If the file represented by the FSN is greater than two files away from the beginning of tape, a rewind is performed. If not, the tape is backspaced to the beginning of the file.

Value	Meaning
-1	For .LOOKUP or .ENTER, this value suppresses rewinding and searches for a file name from the current tape position. Note that if the position is unknown, the handler executes a positioning algorithm that involves backspacing until an EOF label is found. The user should not use any other value since all other negative values are reserved for future use.
-2	For .ENTER, the tape is rewound and spaces forward until the file is found or end of tape is detected. The file is then entered causing a new end of tape when the file is closed.

unit	the logical unit number of a particular terminal in a multi-terminal system.
wcnt	a word count specifying the number of words to be transferred to or from the buffer during an I/O operation.

The RT-11 MACRO assembler supports keyword macro arguments. All of the arguments described above can be encoded using their keyword form (see the PDP-11 MACRO-11 Language Reference Manual for details).

A new argument code is included for all EMT 375 area versions of the macros. It is used for explicit control in expanding an EMT programmed request. In the 375 EMTs, the high byte of the area (pointed to by R0) contains an identifying code for the request. Normally, this byte is set if the macro invocation of the programmed request specifies the area argument, and remains unaffected if the programmed request omits the area argument. If the macro invocation contains CODE=SET, the high byte of the first word of the area is always be set to the appropriate code. This is true whether or not area is specified.

If CODE=NOSET is in the macro invocation, the high byte of the first word of area remains unaffected. This is true whether or not area is specified. The latter case can be used to avoid setting the code when the programmed request is being set up. This might be done because it is known to be set correctly from an earlier invocation of the request using the same area, or because the code was statically set during the assembly process.

Additional information concerning these parameters (and others not defined here) is provided as necessary under each request.

.CDFN

2.4.1 .CDFN

The .CDFN request redefines the number of I/O channels (see Section 2.2.1). Each job, whether foreground or background, is initially provided with 16(decimal) I/O channels, numbered 0-15. .CDFN allows the number to be expanded to as many as 255 (decimal) channels (0-254).

The space used to contain the new channels is taken from within the user program. Each I/O channel requires five words of memory. Therefore, the user must allocate 5*n words of memory, where n is the number of channels to be defined.

It is recommended that the .CDFN request be used at the beginning of a program, before any I/O operations have been initiated. If more than one .CDFN request is used, the channel areas must either start at the same location or not overlap at all. The two requests .SRESET and .HRESET cause the user's channels to revert to the original 16 channels defined at program initiation. Hence, any .CDFNs must be reissued after using .SRESET or .HRESET

Note that .CDFN defines new channels; the space for the previously defined channels cannot be used. Thus, a .CDFN for 20(decimal) channels (while 16 original channels are defined) creates 20 new I/O channels; the space for the original 16 is unused, but the contents of the old channel set are copied to the new channel set.

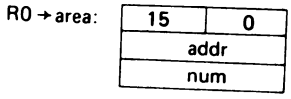
Note that if a program is overlaid, channel 15 (decimal) is used by the overlay handler and should not be modified. (Other channels can be defined and used as usual.)

Macro Call: .CDFN area,addr,num

where:

area	is the address of a three-word EMT argument block
addr	is the address where the I/O channels begin
num	is the number of I/O channels to be created

Request Format:



Errors:

Code	Explanation
0	An attempt was made to define fewer channels than already exist.

Example:

```
.TITLE  CDFN.MAC
;THIS EXAMPLE DEFINES 40 (DECIMAL) CHANNELS TO START
;AT LOCATION CHANL. AN ERROR OCCURS IF 40 OR MORE CHANNELS
;ARE ALREADY DEFINED.
        .MCALL  .CDFN,.PRINT,.EXIT
START:  .CDFN   #RQLIST'#CHANL'#40.
        BCS     BADCDF
        .PRINT  #MSG1
        .EXIT
BADCDF: .PRINT  #MSG2
        .EXIT
MSG1:   .ASCIZ  /.CDFN O.K./
.EVEN
MSG2:   .ASCIZ  /BAD .CDFN/
        .EVEN
RQLIST: .BLKW   3               ;EMT ARGUMENT LIST
CHANL:  .BLKW   40.*5           ;ROOM FOR CHANNELS
        .END    START
```

.CHAIN

2.4.2 .CHAIN

This request allows a background program to pass control directly to another background program without operator intervention. Since this process can be repeated, a large "chain" of programs can be strung together.

The area from locations 500-507 contains the device name and file name (in Radix-50) to be chained to. The area from locations 510-777 is used to pass information between the chained programs.

Macro Call: .CHAIN

Request Format:

R0 = | 10 | | 0 |

Notes:

1. No assumptions should be made concerning which areas of memory remain intact across a .CHAIN. In general, only the resident monitor and locations 500-777 are preserved across a .CHAIN.

2. I/O channels are left open across a .CHAIN for use by the new program. However, new I/O channels opened with a .CDFN request are not available in this way. Since the monitor reverts to the original 16 channels during a .CHAIN, programs that leave files open across a .CHAIN should not use .CDFN. Furthermore, non-resident device handlers are released during a .CHAIN, and must be .FETCHed again by the new program.

3. An executing program can determine whether it was CHAINed to or RUN from the keyboard by examining bit 8 of the JSW. The monitor sets this bit if the program was invoked with .CHAIN. If the program was invoked with R or RUN command, this bit remains cleared. If bit 8 is set, the information in locations 500-777 is preserved from the program that issued the .CHAIN, and is available for the currently executing program to use.

   An example of a calling and a called program is MACRO and CREF. MACRO places important information in the chain area, locations 500-777, then chains to CREF. CREF tests bit 8 of the JSW. If it is clear, it means that CREF was invoked with the R or RUN command and the chain area does not contain useful information. CREF aborts itself immediately. If bit 8 is set, it means that CREF was invoked with .CHAIN and the chain area contains information placed there by MACRO. In this case, CREF executes properly.

Errors:

.CHAIN is implemented by simulating the monitor RUN command and can produce any errors that RUN can produce. If an error occurs, the .CHAIN is abandoned and the keyboard monitor is entered.

When using .CHAIN, care should be taken for initial stack placement, since the program being chained to is started. The linker normally defaults the initial stack to 1000(octal); if caution is not observed, the stack can destroy chain data before it can be used.

Example:

```
.TITLE  CHAIN.MAC
;THIS EXAMPLE CHAINS TO THE PROGRAM CALLED MYPROG.SAV
;AND PASSES A TYPED LINE TO THE PROGRAM.
        .MCALL  .CHAIN,.TTYIN
START:  MOV     #500,R1         ;SET UP TO CHAIN
        MOV     #CHPTR,R2       ;DEVICE, FILE NAME TO 500-511
        .REPT   4
        MOV     (R2)+,(R1)+
        .ENDR
LOOP:   .TTYIN                  ;NOW GET A COMMAND LINE
        MOVB    R0,(R1)+        ;AND PASS IT TO THE JOB
        CMPB    R0,#12          ;IN LOCATIONS 512 AND UP
        BNE     LOOP            ;LOOP UNTIL LINE FEED
        CLRB    (R1)+           ;PUT IN A NULL BYTE
        .CHAIN
CHPTR:  .RAD50  /DK /
        .RAD50  /MYPROG/
        .RAD50  /SAV/
        .END    START
```

.CHCOPY

2.4.3 .CHCOPY (FB and XM Only)

The .CHCOPY request opens a channel for input, logically connecting it to a file that is currently open by the other job for either input or output. This request can be used by either the foreground or the background. .CHCOPY must be issued before the first .READ or .WRITE.

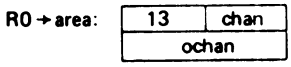
.CHCOPY is legal only on files on disk (including diskette) or DECtape; however, no errors are detected by the system if another device is used. (To close a channel following use of .CHCOPY, use either the .CLOSE or .PURGE request.)

Macro Call: .CHCOPY area,chan,ochan

where:

area	is the address of a two-word EMT argument block
chan	is the channel the current job will use to read the data
ochan	is the channel number of the other job's channel to be copied

Request Format:



Notes:

1. If the other job's channel was opened with .ENTER in order to create a file, the copier's channel indicates a file that extends to the highest block that the creator of the file had written at the time the .CHCOPY was executed.

2. A channel open on a non-file-structured device should not be copied, because intermixture of buffer requests can result.

3. A program can write to a file (that is being created by the other job) on a copied channel just as it could if it were the creator. When the copier's channel is closed, however, no directory update takes place.

Errors:

Code	Explanation
0	Other job does not exist, does not have enough channels defined, or does not have the specified channel (ochan) open.
1	Channel (chan) already open.

Example:

```
.TITLE  CMCOPF.MAC
;THIS IS THE FOREGROUND PROGRAM TO BE RUN IN
;CONJUNCTION WITH CMCOPY.MAC FOR THE EXECUTION OF
;THE CMCOPY EXAMPLE.
        .MCALL  .LOOKUP,.PRINT,.SDATW,.EXIT,.RCVDW

START:  MOV     #AREA,R5
        .LOOKUP R5,#1,#FILE
        BCS     LKERR
        .SDATW  R5,#BUFR,#2     ;PASS BLOCK # AND CHANNEL#
                                ;TO BACKGROUND JOB
        BCS     NJERR           ;NOT THERE
        .RCVDW  R5,#BUF2,#2     ;WAIT FOR RETURN MESSAGE
        .EXIT
NJERR:  MOV     #NJMSG,R0
PMSG:   .PRINT
OK:     .EXIT
LKERR:  MOV     #LKMSG,R0
        BR      PMSG
FILE:   .RAD50  /DK TEST  TMP/
AREA:   .BLKW   5
BUFR:   .WORD   0               ;BLOCK #
        .WORD   1               ;CHANNEL #
BUF2:   .BLKW   3
LKMSG:  .ASCIZ  /LOOKUP ERROR/
NJMSG:  .ASCIZ  /NO BACKGROUND JOB/
        .EVEN
        .END    START

.TITLE  CMCOPY.MAC
;IN THIS EXAMPLE, .CMCOPY IS USED TO READ DATA CURRENTLY
;BEING WRITTEN BY THE OTHER JOB. THE CORRECT BLOCK
;NUMBER AND CHANNEL TO READ IS OBTAINED BY A .RCVDW COMMAND.
;THE CHANNEL NUMBER WILL BE IN MSG+4. THE CMCOPF.MAC PROGRAM
;MUST BE EXECUTED IN THE FOREGROUND.
        .MCALL  .CMCOPY,.RCVDW,.PURGE,.READW,.EXIT,.PRINT
ST:     .PURGE  #0              ;MAKE SURE WE HAVE CLEAR
                                ;CHANNEL
        .RCVDW  #AREA,#MSG,#2   ;READ TWO WORDS, BLOCK #
                                ;AND CHANNEL
        BCS     NOJOB           ;NO JOB THERE
        .CMCOPY #AREA,#0,MSG+4  ;CHANNEL # IS IN THERE
        BCS     BUSY            ;BUT BUSY
        .READW  #AREA,#0,#BUFF,#256.,MSG+2 ;GET THE CORRECT BLOCK
        BCS     RDERR
        .PRINT  #OKMSG
        .EXIT
NOJOB:  .PRINT  #MSG1
        .EXIT
BUSY:   .PRINT  #MSG2
        .EXIT
RDERR:  .PRINT  #MSG3
        .EXIT
AREA:   .BLKW   5
MSG:    .BLKW   5
BUFF:   .BLKW   256.
MSG1:   .ASCIZ  /NO JOB!/
MSG2:   .ASCIZ  /BUSY!/
MSG3:   .ASCIZ  /READ ERROR/
OKMSG:  .ASCIZ  /READ OK/
.EVEN
        .EXIT
        .END    ST
```

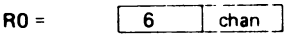
.CLOSE

2.4.4 .CLOSE

The .CLOSE request terminates activity on the specified channel and frees it for use in another operation. The handler for the associated device must be in memory.

Macro Call: .CLOSE chan

Request Format:



A .CLOSE request specifying a channel that is not opened is ignored.

A file opened with .LOOKUP does not require any directory operations when a .CLOSE is issued and the USR does not have to be in memory for the .CLOSE.

A .CLOSE is required on any channel opened for output if the associated file is to become permanent.

A .CLOSE performed on a file opened with .ENTER causes the device directory to be updated to make that file permanent. If the device associated with the specified channel already contains a file with the same name and file type, the old copy is deleted when the new file is made permanent. When an .ENTERed file is .CLOSEd, its permanent length reflects the highest block written since it was entered. For example, if the highest block written is block number 0, the file is given a length of 1; if the file was never written, it is given a length of 0. If this length is less than the size of the area allocated at .ENTER time, the unused blocks are reclaimed as an empty area on the device. In magtape operations, the .CLOSE request causes the handler to write an ANSI EOF1 label in software mode (using MM.SYS or MT.SYS) and to close the channel in hardware mode (using MMHD.SYS or MTHD.SYS).

Errors:

.CLOSE does not return any errors (unless the .SERR system service has been issued). If the device handler for the operation is not in memory, and the .CLOSE request requires updating of the device directory, a fatal monitor error is generated.

Example:

The examples for the .CSISPC and .WRITW requests show typical uses for .CLOSE.

.CMKT

2.4.5 .CMKT (FB and XM Only; SJ Monitor SYSGEN Option)

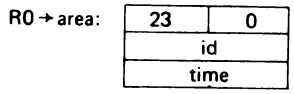
The .CMKT request causes one or more outstanding mark time requests to be cancelled (mark time requests are discussed in Section 2.4.28).

Macro Call: .CMKT area,id,time

where:

- area — is the address of a three-word EMT argument block
- id — is a number used to identify each mark time request to be cancelled. If more than one mark time request has the same id, the request with the earliest expiration time is cancelled. If id = 0, all nonsystem mark time requests (that is, those in the range 1-177377) for the issuing job are cancelled.
- time — is the pointer to a two-word area in which the monitor returns the amount of time remaining in the cancelled request. The first word contains the high-order time, the second contains the low-order. If an address of 0 is specified, no value is returned. If id = 0, the time parameter is ignored and need not be indicated.

Request Format:



Notes:

1. Cancelling a mark time request frees the associated queue element for other uses.
2. A mark time request can be converted into a timed wait by issuing a .CMKT followed by a .TWAIT, and specifying the same time area.
3. If the mark time request to be cancelled has already expired and is waiting in the job's completion queue, .CMKT returns an error code of 0. It does not remove the expired request from the completion queue. The completion routine will eventually be run.

Errors:

Code	Explanation
0	The id was not zero; a mark time request with the specified identification number could not be found (implying that the request was never issued or that it has already expired).

Example:

See the example following the description of the .MRKT request.

.CNTXSW

2.4.6 .CNTXSW (FB and XM Only)

A context switch is an operation performed when a transition is made from running one job to running the other. The .CNTXSW request is used to specify locations to be included in the switching of jobs between background and foreground.

The system always saves the parameters it needs to uniquely identify and execute a job. These parameters include all registers and the following locations:

34,36	Vector for TRAP instruction
40-52	System communication area

If an .SFPA request has been executed with a non-zero address, all floating point registers and the floating point status are also saved.

It is possible that both jobs want to share the use of a particular location that is not included in normal context switch operations. For example, if a program uses the IOT instruction to perform an internal user function (such as printing error messages), the program must set up the vector at 20 and 22 to point to an internal IOT trap handling routine. If both foreground and background wish to use IOT, the IOT vector must always point to the proper location for the job that is executing. Including locations 20 and 22 in the .CNTXSW list for both jobs accomplishes this. In the XM monitor, both IOT and BPT vectors are automatically context switched. The procedure described above is not necessary for jobs running under the XM monitor.

If .CNTXSW is issued more than once, only the latest list is used; the previous address list is discarded. Thus, all addresses to be switched must be included in one list. If the address (addr) is 0, no extra locations are switched. The list cannot be in an area into which the USR swaps, nor can it be modified while a job is running.

In the XM monitor, the .CNTXSW request is ignored for virtual jobs, since they do not share memory with other jobs. The IOT, BPT, and TRAP vectors are simulated for virtual jobs by the monitor. The virtual job sets up the vector in its own virtual space by any of the usual methods (such as a direct move or an .ASECT). When the monitor receives a synchronous trap from a virtual job that was caused by an IOT, BPT, or TRAP instruction, it checks for a valid trap vector and dispatches the trap to the user program in user mapping mode. An invalid trap vector address will abort the job with the following fatal error message:

```
?MON-F-ILL SST (illegal synchronous system trap)
```

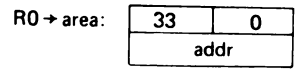
Macro Call: .CNTXSW area,addr

where: area is the address of a 2-word EMT argument block

addr is a pointer to a list of addresses terminated by a zero word. The addresses in the list must be even and:

a. in the range 2-476, or
b. in the user job area, or
c. in the I/O page (addresses 160000-177776).

Request Format:



Errors:

Code	Explanation
0	One or more of the above conditions was violated.

Example:

```
.TITLE  CNTXSW.MAC
;IN THIS EXAMPLE, .CNTXSW REQUEST IS USED TO SPECIFY THAT LOCATION 20
;AND 22 (IOT VECTORS) AND CERTAIN NECESSARY EAE REGISTERS BE CONTEXT
;SWITCHED. THIS ALLOWS BOTH JOBS TO USE IOT AND THE EAE SIMULTANEOUSLY
;YET INDEPENDENTLY.
        .MCALL  .CNTXSW,.PRINT,.EXIT
START:  MOV     #LIST,R0        ;SET R0 TO OUR OWN LIST
        .CNTXSW ,#SWAPLS        ;THE LIST OF ADDRS IS
                                ;AT SWAPLS.
        BCC     1$
        .PRINT  #ADDERR         ;ADDRESS ERROR(SHOULD NOT OCCUR)
        .EXIT
1$:     .PRINT  #CNTOK
        .EXIT
SWAPLS: .WORD   20              ;ADDRESSES TO INCLUDE IN LIST
        .WORD   22
        .WORD   177302
        .WORD   177304
        .WORD   177310
        .WORD   0
LIST:   .BYTE   0,33            ;FUNCTION CODE WORD
        .WORD   0               ;THE MACRO FILLS THIS ONE.
ADDERR: .ASCIZ  /ADDRESSING ERROR/
        .EVEN
CNTOK:  .ASCIZ  /CONTEXT SWITCH O.K./
        .EVEN
        .END    START
```

.CSIGEN

2.4.7 .CSIGEN

The .CSIGEN request calls the Command String Interpreter (CSI) in general mode to process a standard RT-11 command string. In general mode, all file .LOOKUPs and .ENTERs as well as handler .FETCHs are performed. This request gets the program's user command string (dev: output-filespec=dev:input-filespec/options...) into the program, and the following operations occur:

1. The devices specified in the command line are fetched.
2. .LOOKUP and/or .ENTER requests on the files are performed.
3. The option information is placed on the stack.

When called in general mode, the CSI closes channels 0-10 (octal).

.CSIGEN loads all necessary handlers and opens the files as specified. The area specified for the device handlers must be large enough to hold all the necessary handlers simultaneously. If the device handlers exceed the area available, the user program can be destroyed. (The system, however, is protected.)

When control returns to the user program after a call to .CSIGEN, register R0 points to the first available location above the handlers, the stack contains the option information, and all the specified files have been opened for input and/or output. The association is as follows: the three possible output files are assigned to channels 0, 1, and 2(octal); the six input slots are assigned to channels 3 through 10(octal). A null specification causes the associated channel to remain inactive. For example, consider the following string:

```
*,LP:=F1,F2
```

Channel 0 is inactive since the first slot is null. Channel 1 is associated with the line printer, and channel 2 is inactive. Channels 3 and 4 are associated with two files on DK:, while channels 5 through 10 are inactive. The user program can determine whether a channel is inactive by issuing a .WAIT request on the associated channel, which returns an error if the channel is not open.

Options and their associated values are returned on the stack. The first word of the stack contains the number of options. See Section 2.4.8.1 for a description of the way option information is passed.

Macro Call: .CSIGEN devspc,defext,cstrng[,linbuf]

where:

devspc — is the address of the memory area where the device handlers (if any) are to be loaded.

defext — is the address of a four-word block that contains the Radix-50 default file types. These file types are used when a file is specified without a file type.

cstrng — is the address of the ASCIZ input string or a #0 if input is to come from the console terminal. (In an FB environment only, if the input is from the console terminal, an .UNLOCK of the USR is automatically performed, even if the USR is locked at the time.) If the string is in memory, it must not contain a <RET><LF> (octal 15 and 12), but must terminate with a zero byte. If the cstring field is left blank, input is automatically taken from the console terminal. This string, whether in memory or entered at the console, must obey all the rules for a standard RT-11 command string.

linbuf — is the address where the original command string is to be stored. This is a user-supplied area 81 decimal bytes in length. The command string is stored in this area and is terminated with a zero byte instead of <RET> <LF> (octal 15 and 12).

Notes:

The four-word block pointed to by defext is arranged as:

Word 1: default file type for all input channels

Words 2,3,and 4: default file types for output channels 0,1,2, respectively

If there is no default for a particular channel, the associated word must contain 0. All file types are expressed in Radix-50. For example, the following block can be used to set up default file types for a macro assembler:

```
DEFEXT:  .RAD50   "MAC"
         .RAD50   "OBJ"
         .RAD50   "LST"
         .WORD    0
```

In the command string:

```
*DT0:ALPHA,DT1:BETA=DT2:INPUT
```

the default file type for input is MAC; for output, OBJ and LST. The following cases are legal:

```
*DT0:OUTPUT=
*DT2:INPUT
```

In other words, the equal sign is not necessary if only input files are specified.

An optional argument (linbuf) is available in the .CSIGEN format that provides the user with an area to receive the original input string. The input string is returned as an ASCIZ string and can be printed through a .PRINT request.

.CSIGEN automatically takes its input line from an indirect command file if console terminal input is specified (cstring = #0) and the program issuing the .CSIGEN is invoked through an indirect command file.

Errors:

If CSI errors occur and input was from the console terminal, an error message describing the fault is printed on the terminal and the CSI retries the command. If the input was from a string, the carry bit is set and byte 52 contains the error code. The options and option-count are purged from the stack. The errors are:

Code	Explanation
0	Illegal command (bad separators, illegal file name, command too long, etc.).
1	A device specified is not found in the system tables.
2	Unused.
3	An attempt to .ENTER a file failed because of a full directory.
4	An input file was not found in a .LOOKUP.

Example:

```
.TITLE  CSIGEN.MAC
;THIS EXAMPLE USES THE GENERAL MODE OF THE CSI IN A PROGRAM
;TO COPY AN INPUT FILE TO AN OUTPUT FILE. COMMAND INPUT TO THE CSI
;IS FROM THE CONSOLE TERMINAL.
        .MCALL  .CSIGEN,.READW,.PRINT,.EXIT,.WRITW,.CLOSE,.SRESET
ERRWD=52

START:  .CSIGEN #DSPACE,#DEXT   ;GET STRING FROM TERMINAL
        MOV     R0,BUFF         ;R0 HAS FIRST FREE LOCATION
        CLR     INBLK           ;INPUT BLOCK #
        MOV     #LIST,R5        ;EMT ARGUMENT LIST
READ:   .READW  R5,#3,BUFF,#256.,INBLK ;READ CHANNEL 3
        BCC     2$              ;NO ERRORS
        TSTB    @#ERRWD         ;EOF ERROR?
        BEQ     EOF             ;YES
        MOV     #INERR,R0
1$:     .PRINT                  ;ERROR MESSAGE
        CLR     R0              ;HARD EXIT
        .EXIT
2$:     .WRITW  R5,#0,BUFF,#256.,INBLK ;WRITE THE BLOCK
        BCC     NOERR           ;NO ERROR WRITING
        MOV     #WTERR,R0
        BR      1$              ;HARD OUTPUT ERROR
NOERR:  INC     INBLK           ;GET NEXT BLOCK
        BR      READ            ;LOOP UNTIL DONE
EOF:    .CLOSE  #0              ;CLOSE OUTPUT CHANNEL
        .CLOSE  #3              ;AND INPUT CHANNEL
        .SRESET                 ;RELEASE HANDLER FROM MEMORY
        .EXIT                   ;EXIT
DEXT:   .WORD   0,0,0,0         ;NO DEFAULT EXTENSIONS
BUFF:   .WORD   0               ;I/O BUFFER START
INBLK:  .WORD   0               ;RELATIVE BLOCK TO READ/WRITE
LIST:   .BLKW   5               ;EMT ARGUMENT LIST
INERR:  .ASCIZ  /INPUT ERROR/
        .EVEN
WTERR:  .ASCIZ  /OUTPUT ERROR/
        .EVEN
DSPACE:                         ;HANDLER SPACE
        .END    START
```


2.4.8 .CSISPC

The .CSISPC request calls the Command String Interpreter in special mode to parse the command string and return file descriptors and options to the program. In this mode, the CSI does not perform any handler fetches, .CLOSEs, .ENTERs or .LOOKUPs.

Options and their associated values are returned on the stack. However, the optional argument (linbuf) provides the user program with the original command string.

.CSISPC automatically takes its input line from an indirect command file if console terminal input is specified (cstrng = #0) and the program issuing the .CSISPC is invoked through an indirect command file.

Macro Call: .CSISPC outspc,defext,cstrng[,linbuf]

where:

outspc — is the address of the 39-word block to contain the file descriptors produced by .CSISPC. This area can overlay the space allocated to cstring, if desired.

defext — is the address of a four-word block that contains the Radix-50 default file types. These file types are used when a file is specified without a file type.

cstrng — is the address of the ASCIZ input string or a #0 if input is to come from the console terminal. If the string is in memory, it must not contain a <RET><LF> (octal 15 and 12), but must terminate with a zero byte. If cstrng is blank, input is automatically taken from the console terminal.

linbuf — is the address where the original command string is to be stored. This is a user-specified area 81 decimal bytes in length. The command string is stored in this area and is terminated with a zero byte instead of <RET> <LF> (octal 15 and 12).

Notes:

The 39-word file description consists of nine file descriptor blocks (five words for each of three possible output files; four words for each of six possible input files), which correspond to the nine possible files (three output, six input). If any of the nine possible file names are not specified, the corresponding descriptor block is filled with 0s.

The five-word blocks hold Radix-50 four words representing dev:file.type, and one word representing the size specification given in the string. (A size specification is a decimal number enclosed in square brackets [], and following the output file descriptor.) For example:

```
*DT3:LIST.MAC[15]=PC:
```

Using special mode, the CSI returns in the first five-word slot:

```
16101     Radix-50 for DT3
46173     Radix-50 for LIS
76400     Radix-50 for T
50553     Radix-50 for MAC
00017     Octal value of size request
```

In the fourth slot (starting at an offset of 36 octal bytes into outspc), the CSI returns:

```
62170     Radix-50 for PC
0         No file name
0         Specified
0         No file type given
```

Since this is an input file, only four words are returned.

Errors:

Errors are the same as in general mode except that illegal device specifications are checked only for output file specifications with null file names. Since .LOOKUPs and .ENTERs are not done, the valid error codes are:

Code	Explanation
0	Illegal command line
1	Illegal device

Example:

```
.TITLE  CSISPC.MAC
;THIS EXAMPLE ILLUSTRATES THE USE OF THE SPECIAL MODE OF CSI.
;THIS EXAMPLE COULD BE A PROGRAM TO READ A FILE WHICH IS NOT IN
;RT-11 FORMAT TO A FILE UNDER RT-11.
        .MCALL  .CSISPC,.PRINT,.EXIT,.ENTER,.CLOSE

START:  .CSISPC #OUTSPC,#DEXT,#CSTRNG ;GET INPUT FROM A
                                       ;STRING IN MEMORY
        BCC     2$
        MOV     #SYNERR,R0             ;SYNTAX ERROR
1$:     .PRINT                         ;ERROR MESSAGE
        .EXIT
2$:     .ENTER  #LIST,#0,#OUTSPC,#64.  ;ENTER FILE UNDER RT-11
        BCC     3$
        MOV     #ENMSG,R0              ;ENTER FAILED
        BR      1$
3$:     JSR     R5,INPUT               ;ROUTINE INPUT WILL USE
                                       ;THE INFORMATION AT
                                       ;#OUTSPC+36 TO READ INPUT
                                       ;FROM THE NON-RT11 DEVICE.
                                       ;INPUT IS PROCESSED AND
                                       ;WRITTEN VIA .WRITW REQUESTS
        .CLOSE  #0                     ;MAKE OUTPUT FILE PERMANENT.
        .EXIT                          ;AND EXIT PROGRAM
CSTRNG: .ASCIZ  "DT4:RTFIL.MAC=DT2:DOS.MAC"
        .EVEN
DEXT:   .WORD   0,0,0,0                ;NO DEFAULT EXTENSIONS
LIST:   .BLKW   5                      ;LIST FOR EMT CALLS
SYNERR: .ASCIZ  "CSI ERROR"
ENMSG:  .ASCIZ  "ENTER FAILED"
        .EVEN
INPUT:  RTS     R5
OUTSPC: .                              ;CSI LIST GOES HERE
        .END    START
```

2.4.8.1 **Passing Option Information** - In both general and special modes of the CSI, options and their associated values are returned on the stack. A CSI option is a slash (/) followed by any character. The CSI does not restrict the option to printing characters, although it is suggested that printing characters be used wherever possible. The option can be followed by an optional value, which is indicated by a : separator. The : separator is followed by either an octal number, a decimal number or by one to three alphanumeric characters, the first of which must be alphabetic. Decimal values are indicated by terminating the number with a decimal point (/N:14.). If no decimal point is present, the number is assumed to be octal. Options can be associated with files with the CSI. For example:

```
*DK:FOO/A,DT4:FILE.OBJ/A:100
```

In this case, there are two A options. The first is associated with the input file DK:FOO. The second is associated with the input file DT4:FILE.OBJ, and has a value of 100(octal). The format of the stack output of the CSI for options is as follows:

Word #	Value	Meaning
1 (top of stack)	N	Number of options found in command string. If N=0, no options were found.
2	Option value and file number	Even byte = seven-bit ASCII option value. Bits 8-14 = Number (0-10) of the file with which the option is associated. Bit 15 = 1 if the option had a value. = 0 if the option had no value.
3	Option value or next option	If bit 15 of word 2 is set, word 3 contains the option value. If bit 15 is not set, word 3 contains the next option value.

For example, if the input to the CSI is:

```
*FILE/B:20.,FIL2/E=DT3:INPUT/X:SY:20
```

on return, the stack is:

Stack Pointer->	4	Three options appeared (X option has two values and is treated as two options).
	101530	Last option=X; with file 3, has a value.
	20	Value of option X=20 (octal)
	101530	Next option =X; with file 3, has a value.
	075250	Next value of option X=RAD50 code for SY:.
	505	Next option=E; associated with file 1, no value.
	100102	Option=B; associated with file 0 and has a value of 20 (decimal) or 24 (octal).
	24	

As an extended example, assume the following string was input for the CSI in general mode:

```
*FILE[8.],LP:,SY:FILE2[20.]=PC:,DT1:IN1/B,DT2:IN2/M:7
```

Assume also that the default file type block is:

```
DEFEXT:   .RAD50    'MAC'     ;INPUT FILE TYPE
          .RAD50    'OP1'     ;FIRST OUTPUT FILE TYPE
          .RAD50    'OP2'     ;SECOND OUTPUT FILE TYPE
          .RAD50    'OP3'     ;THIRD OUTPUT FILE TYPE
```

The result of this CSI call are:

1. An eight-block file named FILE.OP1 is entered on channel 0 on device DK:; channel 1 is open for output to the device LP:; a 28-block (to show that it is a decimal number) file named FILE2.OP3 is entered on the system device on channel 2.

2. Channel 3 is open for input from paper tape; channel 4 is open for input from a file IN1.MAC on device DT1:; channel 5 is open for input from IN2.MAC on device DT2:.

3. The stack contains options and values as follows:

Contents	Explanation
2	Two options found in string.
102515	Second option is M, associated with Channel 5; has a value.
7	Numeric value is 7 (octal).
2102	Option is B, associated with Channel 4; has no value.

If the CSI were called in special mode, the stack would be the same as for the general mode call, and the descriptor table would contain:

```
OUTSPC:   15270     ;.RAD50   'DK'
          23364     ;.RAD50   'FIL'
          17500     ;.RAD50   'E'
          60137     ;.RAD50   'OP1'
             10     ;LENGTH OF 8 BLOCKS (DECIMAL)
          46600     ;.RAD50   'LP'
              0     ;NO NAME OR LENGTH SPECIFIED
              0
              0
              0
          75250     ;.RAD50   'SY'
          23364     ;.RAD50   'FIL'
          22100     ;.RAD50   'E2'
          60141     ;.RAD50   'OP3'
             24     ;LENGTH OF 20 (DECIMAL)
          62170     ;.RAD50   'PC'
              0
              0
              0
          16077     ;.RAD50   'DT1'
          35217     ;.RAD50   'IN1'
              0     ;.RAD50   '   '
          50553     ;.RAD50   'MAC'
          16100     ;.RAD50   'DT2'
          35220     ;.RAD50   'IN2'
              0     ;.RAD50   '   '
          50553     ;.RAD50   'MAC'
              0
              .
              .
              .
              0     (12 more zero words
                          are returned)
```

Keyboard error messages that can occur from incorrect use of the CSI when input is from the console keyboard include:

Message	Meaning
?CSI-F-Illegal command	Syntax error.
?CSI-F-File not found	Input file was not found.
?CSI-F-Device full	Output file does not fit.
?CSI-F-Illegal device	Device specified does not exist.

Notes:

1. In many cases, the user program does not need to process options in CSI calls. However, the user at the console can inadvertently enter options. In this case, it is wise for the program to save the value of the stack pointer before the call to the CSI, and restore it after the call. In this way, no extraneous values are left on the stack. Note that even a command string with no options causes a word to be pushed onto the stack.

2. In the FB monitor, calls to the CSI that require console terminal input always do an implicit .UNLOCK of the USR. This should be kept in mind when using .LOCK calls.

.CSTAT

2.4.9 .CSTAT (FB and XM Only)

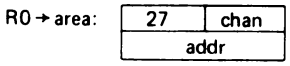
This request furnishes the user with information about a channel. It is supported only in the FB and XM environments; no information is returned by the SJ monitor.

Macro Call: .CSTAT area,chan,addr

where:

area	is the address of a two-word EMT argument block
chan	is the number of the channel about which information is desired
addr	is the address of a six-word block to contain the status

Request Format:



Notes:

The six words passed back to the user are:

1. Channel status word (bit Ø set = hard error;
   bit 13 set = end of file)

2. Starting block number of file (0 if sequential-access device or if channel was opened with a non-file-structured .LOOKUP or .ENTER)

3. Length of file (no information if non-file-structured device or if channel was opened with a non-file-structured .LOOKUP or .ENTER)

4. Highest relative block written since file was opened (no information if non-file-structured device)

5. Unit number of device with which this channel is associated

6. Radix-50 of the device name with which the channel is associated (this is a physical device name, unaffected by any user name assignment in effect)

The fourth word (highest block) is maintained by the .WRITE/.WRITC/.WRITW requests. If data is being written on this channel, the highest relative block number is kept in this word.

Errors:

Code	Explanation
0	The channel is not open.

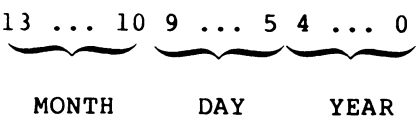
Example:

```
.TITLE  CSTAT.MAC
;IN THIS EXAMPLE, .CSTAT IS USED TO DETERMINE THE .RAD50
;REPRESENTATION OF THE DEVICE WITH WHICH THE CHANNEL IS ASSOCIATED.
        .MCALL  .CSTAT,.CSIGEN,.PRINT,.EXIT
ST:     .CSIGEN #DEVSDC,#DEFEXT ;OPEN FILES
        .CSTAT  #AREA,#0,#ADDR  ;GET THE STATUS
        BCS     NOCHAN          ;CHANNEL 0 NOT OPEN
        MOV     #ADDR+10,R5     ;POINT TO UNIT #
        MOV     (R5)+,R0        ;UNIT # TO R0
        ADD     (PC)+,R0        ;MAKE IT RAD50
        .RAD50  /  0/
        ADD     (R5),R0         ;GET DEVICE NAME
        MOV     R0,DEVNAM       ;DEVNAM HAS RAD50 DEVICE NAME
        .EXIT
AREA:   .BLKW   5               ;EMT ARG LIST
ADDR:   .BLKW   6               ;AREA FOR CHANNEL STATUS
DEVNAM: .WORD   0               ;STORAGE FOR DEVICE NAME
DEFEXT: .WORD   0,0,0,0
NOCHAN: .PRINT  #MSG
        .EXIT
MSG:    .ASCIZ  /NO OUTPUT FILE/
        .EVEN
DEVSDC:.
        .END    ST
```

.DATE

2.4.10 .DATE

This request returns the current date information from the system date word in R0. The date word returned is in the following format:



The year value in bits 4-0 is the actual year minus 72.

NOTE

RT-11 does not support month and year roll-over. The keyboard monitor DATE command must be issued to change the month and year appropriately.

Macro Call: .DATE

Request Format:

R0 = | 12 | 0 |

Errors:

No errors are returned. A zero result in R0 indicates that the user has not entered a date.

Example:

This example is a subroutine that can be assembled separately and linked with a user's program.

```
.TITLE  DATE.MAC
;
; CALLING SEQUENCE:
;
;       JSR     PC,DATE
;
; INPUT: NONE
;
; OUTPUT: R0 = DAY (1-31)
;         R1 = MONTH (1-12)
;         R2 = YEAR -72
;
; ERROR: CARRY SET INDICATES NO DATE SPECIFIED
;
        .MCALL  .DATE
DATE::
        .DATE                   ;GET THE SYSTEM DATE
        MOV     R0,R2           ;COPY THE DATE
        BEQ     10$             ;BRANCH IF NO DATE
        BIC     #^C37,R2        ;ISOLATE THE YEAR
        ASR     R0              ;PUT THE MONTH ON A BYTE BOUNDARY
        ASR     R0              ;
        MOV     R0,R1           ;COPY THE DATE
        SWAB    R1              ;PUT THE MONTH IN THE LOW BYTE
        BIC     #^C37,R1        ;ISOLATE THE MONTH
        ASR     R0              ;SHIFT THE DAY TO THE BYTE BOUNDARY
        ASR     R0              ;
        ASR     R0              ;
        BIC     #^C37,R0        ;ISOLATE THE DAY
        CLC                     ;INDICATE NO ERROR
        BR      20$             ;RETURN
10$:    SEC                     ;NO DATE.  INDICATE ERROR
20$:    RTS     PC

        .END
```

.DELETE

2.4.11 .DELETE

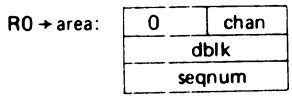
The .DELETE request deletes a named file from an indicated device. This request generates a monitor error if a hard I/O error is detected during directory I/O. The .SERR programmed request can be used to allow the program to process the error. .DELETE is illegal for magtapes.

Macro Call: .DELETE area,chan,dblk,seqnum

where:

area	is the address of a three-word EMT argument block.
chan	is the device channel number in the range 0-377 (octal)
dblk	is the pointer to the address of a four-word Radix-50 descriptor of the file to be deleted.
seqnum	file number for cassette operations: if this argument is blank, a value of 0 is assumed.

Request Format:



Note:

The channel specified in the .DELETE request must not be open when the request is made, or an error will occur. The file is deleted from the device, and an empty (UNUSED) entry of the same size is put in its place. A .DELETE issued to a non-file-structured device is ignored. .DELETE requires that the handler to be used be in memory at the time the request is made. When the .DELETE is complete, the specified channel is left inactive.

Errors:

Code	Explanation
0	Channel is active
1	File was not found in the device directory
2	Illegal operation

Example:

```
.TITLE  DELETE.MAC
;THIS EXAMPLE USES THE SPECIAL MODE OF CSI TO DELETE FILES.
;INSPEC IS THE ADDRESS OF THE FIRST INPUT SLOT IN THE CSI
;INPUT TABLE.
        .MCALL  .SRESET,.CSISPC,.DELETE,.PRINT,.EXIT
START:  .SRESET                 ;MAKE SURE CHANNELS
                                ;ARE FREE
        .CSISPC #OUTSPC,#DEFEXT ;GET COMMAND LINE
                                ;TERMINAL DIALOG WAS
                                ;DT:FILE
        .DELETE #LIST,#0,#INSPC ;USE CHANNEL 0 TO
                                ;DELETE THE FILE
                                ;WHICH IS AT THE
                                ;FIRST INPUT SLOT.
        BCC     1$              ;OK, LOOP AGAIN
        .PRINT  #NOFILE         ;NO SUCH FILE
1$:     .EXIT                   ;EXIT
NOFILE: .ASCIZ  /FILE NOT FOUND/
        .EVEN
DEFEXT: .RAD50  /MAC/           ;.MAC INPUT EXTENSION
        .WORD   0,0,0           ;NO OUTPUT DEFAULTS
LIST:   .BLKW   2               ;EMT ARG LIST
OUTSPC: .
INSPC:  .+36
        .BLKW   39.
        .END    START
```

.DEVICE

2.4.12 .DEVICE (FB and XM Only)

This request allows the user to set up a list of addresses to be loaded with specified values when a program is terminated. Upon an .EXIT or CTRL/C, this list is picked up by the system and the appropriate addresses are filled with the corresponding values. This function is primarily designed to allow user programs to load device registers with necessary values. In particular, it is used to turn off a device's interrupt enable bit when the program servicing the device terminates. Successive calls to .DEVICE are allowed when the user needs to link requested tables. When the job is terminated for any reason, the list is scanned once. At that point, the monitor disables the feature until another .DEVICE call is executed. Thus, background programs that are reenterable should include .DEVICE as a part of the reenter code.

The .DEVICE request is ignored when it is issued by a virtual job running under the XM monitor.

Macro Call: .DEVICE area,addr[,link]

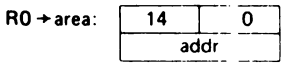
where:

- area — is the address of a two-word EMT argument block.
- addr — is the address of a list of two-word elements, each composed of a one-word address and a one-word value to be put at that address.

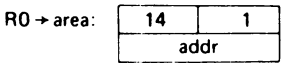
link — is the optional argument, L, that allows linking of tables on successive calls to .DEVICE. If the argument is omitted, the list referenced in the previous .DEVICE request is replaced by the new list. The argument must be supplied to cause linking of lists; however, linked and unlinked list types cannot be mixed.

Request format:

Non-linking



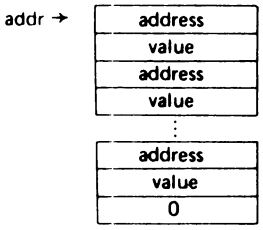
Linking



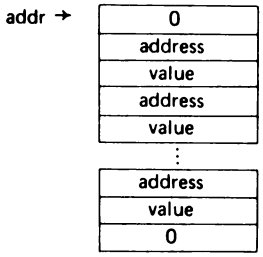
NOTE

The list referenced by addr must be either in linking format or non-linking format. The different formats are shown below. Both formats must be terminated with a separate, zero-value word. Linking format must also have a zero-value word as its first word.

Non-linking



Linking



Errors:

None.

Example:

```
.TITLE  DEVICE.MAC
;THE FOLLOWING EXAMPLE SHOWS .DEVICE IS USED TO DISABLE
;INTERRUPTS FROM THE AFC11 (A-D CONVERTER
;SUB-SYSTEM).
        .MCALL  .DEVICE,.EXIT

START:  .DEVICE #LIST
        .EXIT
LIST:   .BYTE   0,14            ;EMT ARG LIST
        .WORD   ATOD
ATOD:   172570                  ;ADDRESS IS 172570
        0                       ;JAM A 0 INTO IT
        0                       ;THIS 0 TERMINATES THE LIST.
        .END    START
```

.DSTATUS

2.4.13 .DSTATUS

This request is used to obtain information about a particular device.

Macro Call: .DSTATUS retspc,dnam

where: retspc is the four-word space used to store the status information.

dnam is the pointer to the Radix-50 device name.

.DSTATUS looks for the device specified by dnam and, if found, returns four words of status starting at the address specified by retspc. The four words returned are:

1. Status Word

Bits 7-0: contain a number that identifies the device in question. The values (octal) currently defined are:

```
    0 = RK05 Disk
    1 = TC11 DECtape
    2 = Reserved
    3 = Line Printer
    4 = Console Terminal or
        Batch Handler
    5 = RL01 Disk
    6 = RX02 Diskette
    7 = PC11 High-speed paper tape reader and
        punch
   10 = Reserved
   11 = Magtape (TM11, TMA11)
   12 = RF11 Disk
   13 = TA11 Cassette
   14 = Card Reader (CR11,CM11)
   15 = Reserved
   16 = RJ03/4 Fixed-head Disks
   17 = Reserved
   20 = TJU16 Magtape
   21 = RP02 Disk
   22 = RX01 Diskette
   23 = RK06/07 Disk
   24 = Error Log Handler
   25 = Null Handler
26-30 = Reserved (NETWORKS)
31-33 = Reserved (DIBOL LP,LQ,LR,LS)
```

Bit 15: 1= Random-access device (disk, DECtape)
0= Sequential-access device (line printer, paper tape, card reader, magtape, cassette, terminal)

Bit 14: 1= Read-only device (card reader, paper tape reader)

Bit 13: 1= Write-only device (line printer, paper tape punch)

Bit 12: 1= Non RT-11 directory-structured device (magtape, cassette)

Bit 11: 1= Enter handler abort entry every time a job is aborted.
0= Handler abort entry taken only if there is an active queue element belonging to aborted job.

Bit 10: 1= Handler accepts .SPFUN requests (for example, MT, CT, DX).
0= .SPFUN requests are rejected as illegal.

2. Handler size.

   The size of the device handler, in bytes.

3. Load address +6.

   Non-zero implies the handler is now in memory; zero implies it must be .FETCHed before it can be used. The address of the handler is the load address +6.

4. Device size.

   The size of the device (in 256-word blocks) for block-replaceable devices; 0 for sequential-access devices. The last block on the device is the device size minus 1.

The device name can be a user-assigned name. DSTATUS information is extracted from block 0 of the device handler. Therefore, this request requires the handler file for the device to be present on the system volume, unless the device is the system device. The system device handler is always memory resident.

Errors:

Code	Explanation
0	Device not found in tables.

Example:

```
.TITLE  DSTATU.MAC
;THIS EXAMPLE SHOWS HOW TO DETERMINE IF A PARTICULAR DEVICE HANDLER
;IS IN MEMORY AND, IF IT IS NOT, HOW TO FETCH IT THERE.
        .MCALL  .DSTATUS,.PRINT,.EXIT,.FETCH

START:  .DSTATUS #CORE,#FPTR    ;GET STATUS OF DEVICE
        BCC     1$
        .PRINT  #ILLDEV         ;DEVICE NOT IN TABLES
        .EXIT
1$:     TST     CORE+4          ;IS DEVICE RESIDENT?
        BNE     2$
        .FETCH  #HNDLR,#FPTR    ;NO, GET IT
        BCC     2$
        .PRINT  #FEFAIL         ;FETCH FAILED
        .EXIT
2$:     .PRINT  #FECHOK
        .EXIT
CORE:   .BLKW   4       ;DSATUS GOES HERE
FPTR:   .RAD50  /DT0/   ;DEVICE NAME
        .RAD50  /FILE  MAC/ ;FILE NAME
FEFAIL: .ASCIZ  /FETCH FAILED/
ILLDEV: .ASCIZ  /ILLEGAL DEVICE/
        .EVEN
FECHOK: .ASCIZ  /FETCH O.K./
        .EVEN
HNDLR:.                 ;HANDLER WILL GO HERE
        .END    START
```

.ENTER

2.4.14 .ENTER

The .ENTER request allocates space on the specified device and creates a tentative entry for the named file. The channel number specified is associated with the file. (Note that if the program is overlaid, channel 15 is used by the overlay handler and should not be modified.)

Macro Call: .ENTER area,chan,dblk,len,seqnum

where:

area — is the address of a four-word EMT argument block

chan — a channel number in the range 0-377 (octal)

dblk — the address of a four-word Radix-50 descriptor of the file to be operated upon

len — is the file size specification. If the argument is left blank, it is not set to 0 in area. The #0 must be specified to accomplish this. If an argument is left blank, the corresponding location in area is assumed to be set.

The value of this argument determines the file length allocation as follows:

0 - either half the largest empty entry or the entire second-largest empty entry, whichever is larger. (A maximum size for non-specific .ENTERs can be patched in the monitor by changing RMON offset 314).

m - a file of m blocks. The size, m, can exceed the maximum mentioned above.

-1 - the largest empty entry on the device.

seqnum — file number for cassette. If this argument is blank, a value of 0 is assumed.

For magtape it describes a file sequence number that can have the following values:

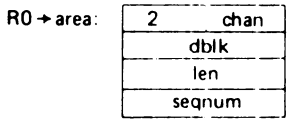
0 - means rewind the magtape and space forward until the file name is found or until logical end-of-tape is detected. If file name is found, delete it and continue tape search.

n - means position magtape at file sequence number n. If the file represented by the file sequence number is greater than two files away from beginning of tape, then a rewind is performed. If not, the tape is backspaced to the beginning of the file.

-1 - means space to the logical end-of-tape and enter file.

-2 - means rewind the magtape and space forward until the file name is found, or until logical end-of-tape is detected. The magtape is now positioned correctly. A new logical end-of-tape is implied.

Request Format:



The file created with an .ENTER is not a permanent file until the .CLOSE on that channel is given. Thus, the newly created file is not available to .LOOKUP and the channel cannot be used by .SAVESTATUS requests. However, it is possible to go back and read data that has just been written into the file by referencing the appropriate block number. When the .CLOSE to the channel is given, any already existing permanent file of the same name on the same device is deleted and the new file becomes permanent. Although space is allocated to a file during the .ENTER operation, the actual length of the file is determined when .CLOSE is requested.

Each job can have up to 256 files open on the system at any time. If required, all 256 can be opened for output with the .ENTER function. .ENTER requires that the device handler be in memory when the request is made. Thus, a .FETCH should normally be executed before an .ENTER can be done. On return, R0 contains the size of the area actually allocated for use.

Notes:

When using the zero-length feature of .ENTER, it must be kept in mind that the space allocated is less than the largest empty space. This can have an important effect in transferring files between devices (particularly DECtape and diskette) that have a relatively small capacity. For example, to transfer a 200-block file to a DECtape on which the largest available empty space is 300 blocks, a zero-length transfer does not work. Since the .ENTER allocates half the largest space, only 150 blocks are really allocated and an output error occurs during the transfer. However, when transferring from A to B and the length is unknown on A, do a .LOOKUP first. This request returns the length and this value can be used to do a fixed-length .ENTER. If a specific length of 200 is requested, however, the transfer proceeds without error. The .ENTER request also generates hard errors when problems are encountered during directory operations. These errors can be detected after the operation with the .SERR request.

Errors:

Code	Explanation
0	Channel is in use.
1	In a fixed length request, no space greater than or equal to m was found, or in a non-specific request, the device or the directory was found to be full.

Example:

```
.TITLE  ENTER.MAC
;.ENTER MAY BE USED TO OPEN A FILE ON A SPECIFIED DEVICE, AND
;THEN WRITE DATA FROM MEMORY INTO THAT FILE AS FOLLOWS:
        .MCALL  .ENTER,.WRITW,.CLOSE,.PRINT
        .MCALL  .SRESET,.EXIT,.FETCH,.SETTOP

START:  .SRESET                 ;MAKE SURE ALL CHANNELS
                                ;ARE CLOSED.
        .SETTOP #-2             ;ASK FOR ALL AVAILABLE MEMORY
        .FETCH  LIMIT+2,#FPRT   ;FETCH DEVICE HANDLER
        BCS     BADFET          ;.FETCH ERROR, PROBABLY
                                ;ILLEGAL DEVICE.
        .ENTER  #AREA,#0,#FPRT,#0 ;OPEN A FILE ON THE DEVICE
                                ;SPECIFIED, LENGTH 0 WILL
                                ;GIVE 1/2 OF LARGEST EMPTY
                                ;SPACE NOW AVAILABLE.
        BCS     BADENT          ;FAILED, CHANNEL PROBABLY BUSY
        .WRITW  #AREA,#0,#BUFF,#END-BUFF/2,#0
                                ;WRITE DATA FROM MEMORY, THE
                                ;SIZE IS # OF WORDS BETWEEN
                                ;BUFF AND END, START AT BLOCK 0.
        BCS     BADWRT          ;WRITE FAILURE.
        .CLOSE  #0              ;CLOSE THE FILE
        .EXIT                   ;AND GO TO KEYBOARD MONITOR.
FPRT:   .RAD50  /DK /           ;FILE WILL BE ON DK
        .RAD50  /FILE  EXT/     ;NAMED FILE.EXT
AREA:   .BLKW   10              ;EMT ARGUMENT LIST
BADFET: .PRINT  #FMSG
        .EXIT
BADENT: .PRINT  #EMSG
        .EXIT
BADWRT: .PRINT  #WMSG
        .EXIT
FMSG:   .ASCIZ  /BAD FETCH/
EMSG:   .ASCIZ  /BAD ENTER/
WMSG:   .ASCIZ  /WRITE ERROR/
        .EVEN
LIMIT:  .LIMIT                  ;PROGRAM LIMITS
BUFF:
        .REPT   400             ;THIS IS BUFFER TO BE WRITTEN OUT
        .WORD   0,1
        .ENDR
END:
        .END    START
```

.EXIT

2.4.15 .EXIT

The .EXIT request causes the user program to terminate. When used from a background job under the FB monitor or XM monitor, or in SJ, .EXIT causes KMON to run in the background area. All outstanding mark time requests are cancelled. Any I/O requests and/or completion routines pending for that job are allowed to complete. If part of the background job resides where KMON and USR are to be read, the user job is written onto the system swap blocks (the file SWAP.SYS). KMON and USR are then loaded and control goes to KMON in the background area. If R0 = 0 when the .EXIT is done, an implicit .HRESET is executed when KMON is entered, disabling the subsequent use of REENTER, START or CLOSE.

The .EXIT request allows a user program to pass command lines to KMON in the chain information area (locations 500-777(octal)) for execution after the job exits. This operation is performed in the following manner:

1. The word (not byte) location 510 must contain the total number of bytes of command lines to be passed to KMON.

2. The command lines are stored beginning at location 512. The lines must be .ASCIZ strings with no embedded carriage return or line feed. For example:

```
        .=510
        .WORD B-A
A:      .ASCIZ /COPY A.MAC B.MAC/
        .ASCIZ /DELETE A.MAC/
B =     .
```

3. The user program must set bit 11 in the JSW immediately prior to doing an .EXIT. The .EXIT must be issued with R0 = 0.

When the .EXIT request is used to input command lines to KMON, the following restrictions are in effect:

1. If the feature is used by a program that is invoked through an indirect file, the indirect file context is aborted prior to executing the supplied command lines. Any unexecuted lines in the indirect file are never executed.

2. An indirect file can be invoked using this mechanism only if a single line containing the indirect file specification is passed to KMON. Attempts to pass multiple indirect files or combinations of indirect command files and other KMON commands yield incorrect results.

EXIT also resets any .CDFN and .QSET calls that were done and executes an .UNLOCK if a .LOCK has been done. Thus, the .CLOSE command from the keyboard monitor does not operate for programs that perform .CDFN requests.

.EXIT from a completion routine is illegal.

NOTE

It is the responsibility of the user program to ensure that the data being passed to KMON is not destroyed during the .EXIT request. Extreme care should be exercised to ascertain that the user stack does not overwrite this data area.

Macro Call: .EXIT

Errors:

None.

Example:

The following example shows how a program can execute a keyboard command after exiting.

```
.TITLE  EXIT.MAC
CHNIFS  = 4000
JSW     = 44
        .MCALL  .EXIT
FINI:   MOV     #510,R0         ;R0 -> COMMUNICATION AREA
        MOV     #CMDSTR,R1      ;R1 -> COMMAND LIST
        MOV     #FINI,SP        ;MAKE SURE THAT THE STACK IS
                                ;NOT IN THE COMMUNICATION AREA
10$:    MOV     (R1)+,(R0)+     ;COPY COMMAND STRING
        CMP     R1,#CMDEND      ;DONE?
        BLO     10$             ;BR IF NOT
        BIS     #CHNIFS,@#JSW   ;SET THE BIT THAT
                                ;TELLS KMON WE LEFT
                                ;A COMMAND LINE FOR IT
        CLR     R0              ;R0 MUST BE ZERO
        .EXIT
CMDSTR: .WORD   CMDEND-CMDST1
CMDST1: .ASCIZ "DIRECT/FULL *.MAC"
CMDEND:
        .EVEN
        .END FINI
```

.FETCH/.RELEAS

2.4.16 .FETCH/.RELEAS

The .FETCH request loads device handlers into memory from the system device.

Macro Call: .FETCH addr,dnam

where:

addr	is the address where the device handler is to be loaded.
dnam	is the pointer to the Radix-50 device name.

The storage address for the device handler is passed on the stack. When the .FETCH is complete, R0 points to the first available location above the handler. If the handler is already in memory, R0 keeps the same value as was initially pushed onto the stack. If the argument on the stack is less than 400(octal), it is assumed that a handler .RELEAS is being done. (.RELEAS does not dismiss a handler that was LOADed from the KMON; an UNLOAD must be done.) After a .RELEAS, a .FETCH must be issued in order to use the device again.

Several requests require a device handler to be in memory for successful operation. These include:

.CLOSE	.READC	.READ
.LOOKUP	.WRITC	.WRITE
.ENTER	.READW	.SPFUN
.RENAME	.WRITW	.DELETE

It is necessary for all handlers to be resident before using a .FETCH in the XM monitor; a fatal error occurs otherwise. In the FB monitor, this is necessary only if the .FETCH is issued from within a foreground job (rather than from a background job).

Errors:

Code	Explanation
0	The device name specified does not exist, or there is no handler for that device in the system.

NOTE

I/O operations cannot be executed on devices unless the handler is resident in memory.

Example:

```
.TITLE  FETCH.MAC
;IN THIS EXAMPLE, THE TT AND PC HANDLERS ARE FETCHED INTO MEMORY
;IN PREPARATION FOR THEIR USE BY A PROGRAM. THE PROGRAM SETS ASIDE
;HANDLER SPACE FROM ITS FREE MEMORY AREA.
        .MCALL  .FETCH,.PRINT,.EXIT,.SETTOP
START:
        MOV     LIMIT+2,FREE    ;SET UP FREE MEMORY POINTER
        .SETTOP #-2             ;ASK FOR ALL AVAILABLE MEMORY
        MOV     R0,LIMIT+2      ;SAVE THE NEW HIGH LIMIT
        .FETCH  FREE,#TTNAME    ;FETCH HANDLERS AT THE 1ST
                                ;FREE LOCATION IN MEMORY
        BCS     FERR            ;FETCH ERROR
        MOV     R2,R2           ;R2 => NEXT FREE LOCATION
        .FETCH  R2,#PCNAME      ;FETCH PC HANDLER
                                ;IMMEDIATELY FOLLOWING
                                ;TT HANDLER. R0 POINTS
                                ;TO THE TOP OF PC
                                ;HANDLER ON RETURN
                                ;FROM THAT CALL.
        BCS     FERR            ;NO PC HANDLER
        MOV     R0,FREE         ;UPDATE FREE MEMORY
                                ;POINTER TO POINT TO
                                ;NEW BOTTOM OF FREE
                                ;AREA(TOP OF HANDLERS).


        .PRINT  #OK
        .EXIT
OK:     .ASCIZ  /FETCH O.K./
        .EVEN
FERR:   .PRINT  #MSG            ;PRINT ERROR MESSAGE
        .EXIT                   ;AND EXIT
TTNAME: .RAD50  "TT "           ;DEVICE NAMES
PCNAME: .RAD50  "PC "
MSG:    .ASCIZ  "DEVICE NOT FOUND" ;ERROR MESSAGE
        .EVEN
FREE:   .WORD   0               ;FREE MEMORY POINTER
LIMIT:  .LIMIT                  ;PROGRAM LIMITS.  2ND
                                ;WORD IS THE HIGH LIMIT
        .END    START
```

The .RELEAS request notifies the monitor that a FETCHed device handler is no longer needed. The .RELEAS request does not modify memory contents nor does it change any free space pointers. The .RELEAS is ignored if the handler is:

1. Part of RMON (that is, the system device), or

2. Not currently resident, or

3. Resident because of a LOAD command to the keyboard monitor

.RELEAS from the foreground job under the FB monitor or from any job under the XM monitor is always ignored, since the foreground job in FB and all jobs in XM can only use handlers that have been LOADed.

Macro Call: .RELEAS dnam

where: dnam is the address of the Radix-50 device name.

Errors:

Code	Explanation
0	Handler name was illegal.

Example:

```
.TITLE  RELEAS.MAC
;IN THIS EXAMPLE, THE DECTAPE HANDLER (DT) IS LOADED INTO MEMORY,
;USED, THEN RELEASED. IF THE SYSTEM DEVICE IS DECTAPE, THE HANDLER IS
;ALWAYS RESIDENT, AND .FETCH WILL RETURN HSPACE IN R0.
        .MCALL  .FETCH,.RELEAS,.EXIT

START:  .FETCH  LIMIT+2,#DTNAME ;LOAD DT HANDLER
        BCS     FERR            ;NOT AVAILABLE

; USE HANDLER

        .RELEAS #DTNAME         ;MARK DT NO LONGER IN
                                ;MEMORY.
        BR      START
FERR:   HALT                    ;DT NOT AVAILABLE
DTNAME: .RAD50  /DT /           ;NAME FOR DT HANDLER
LIMIT:  .LIMIT                  ;PROGRAM LIMITS

        .END    START
```

.FORK

2.4.17 .FORK

.FORK can be used within a standard RT-11 device driver to request a synchronous system process after an interrupt occurs. The request does not use the EMT instruction but issues a subroutine call to the monitor. The .FORK call must be preceded by an .INTEN call, and the address of a four-word block must be supplied with the request. The user program must not have left any information on the stack between the .INTEN and the .FORK call. The contents of registers R4 and R5 are preserved through the call, and on return registers R0-R3 are available for use.

The .FORK request is used when access to a shared resource must be serialized or when a lengthy but non-time-critical section of code must be executed. The .FORK request is linked into a queue and serviced on a first-in/first-out basis. On return to the driver instruction following the call, the interrupt has been dismissed and the driver is executing at priority 0. Therefore, the .FORK request must not be used where it can be reentered using the same fork block by another interrupt, for example. It also should not be used with devices that have continuous interrupts that cannot be disabled. Chapter 1 of this manual has additional information on the .FORK request.

Macro call: .FORK fkblk

where: fkblk is a four-word block of memory allocated within the driver.

Errors:

None.

Note:

For use within a user interrupt service routine, monitor fixed offset 402 (FORK) contains the offset from the start of the resident monitor to the .FORK request processor. A .FORK request can be done by computing the address of the .FORK request processor and using a subroutine instruction. (Under the XM monitor, only privileged jobs can contain user interrupt service routines.) For example:

```
MOV   @#54,R4     ;GET BASE OF RMON
ADD   #402,R4     ;OFFSET TO FORK PROCESSOR
JSR   R5,@R4      ;CALL FORK PROCESSOR
.WORD BLOCK-.     ;FORK BLOCK
```


2.4.18 .GTIM

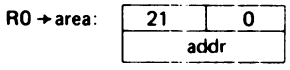
.GTIM allows user programs to access the current time of day. The time is returned in two words, and is given in terms of clock ticks past midnight.

Macro Call: .GTIM area,addr

where: area is the address of a two-word EMT argument block.

addr is a pointer to the two-word area where the time is to be returned.

Request Format:



The high-order time is returned in the first word, the low-order time in the second word. User programs must make the conversion from clock ticks to hours, minutes, and seconds.

The basic clock frequency (50 or 60 Hz) can be determined from the configuration word in the monitor (see Section 2.2.6). In the FB monitor, the time of day is automatically reset after 24:00 when a .GTIM is done; in the SJ monitor, it is not. The month is not automatically updated in either monitor.

The default clock rate is 60-cycle. Consult the RT-11 System Generation Manual if conversion to a 50-cycle rate is necessary.

NOTE

There are also several SYSLIB routines that perform time conversion. They are as follows:

1. CVTTIM (see Section 4.3.5)
2. TIMASC (see Section 4.3.98)
3. TIME (see Section 4.3.99)
4. SECNDS (see Section 4.3.93)

Errors:

None.

Example:

```
.TITLE  GTIM.MAC
        .MCALL  .GTIM,.EXIT

START:
        .GTIM   #LIST,#TIME

        .EXIT
TIME:   .WORD   0,0             ;LOW AND HI ORDER TIME
                                ;RETURNED HERE.

LIST:   .BLKW   2               ;ARGUMENTS FOR THE EMT

        .END    START
```

.GTJB

2.4.19 .GTJB

The .GTJB request passes a job number, the low memory limit and other job parameters back to the user program.

In the SJ monitor, the job number and low memory limit are always 0. In the FB or XM monitor, the job number can either be 0 or 2. If the job number equals 0 (background job), word 3 equals 0.

Word 4 describes where the I/O channel words begin. This is normally an address within the resident monitor. When a .CDFN is executed, however, the start of the I/O channel area changes to the user-specified area.

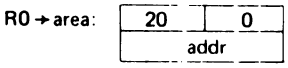
Macro Call: .GTJB area,addr

where: area — is the address of a two-word EMT argument block.

addr — is the address of an eight-word block into which the parameters are passed. The values returned are:

- Word 1 - Job Number. 0=Background 2=Foreground
- 2 - High memory limit of job partition
- 3 - Low memory limit of job partition
- 4 - Beginning of I/O channel space
- 5 - Address of job's impure area in FB and XM monitors
- 6-8 - Reserved for future use

Request Format:



Errors:

None.

Example:

```
.TITLE  GTJB.MAC
;USE .GTJB TO DETERMINE IF THIS PROGRAM IS EXECUTING AS A FOREGROUND
;OR A BACKGROUND JOB.
        .MCALL  .GTJB,.PRINT,.EXIT

START:
        .GTJB   #LIST,#JOBARG   ;R0 POINTS TO 1ST WORD ON
                                ;RETURN FROM CALL.
        MOV     #FMSG,R1
        TST     JOBARG          ;BACKGROUND?
        BNE     1$              ;NO, PRINT FMSG
        MOV     #BMSG,R1
1$:     .PRINT  R1

        .EXIT

FMSG:   .ASCIZ  /PROGRAM IN FOREGROUND/
BMSG:   .ASCIZ  /PROGRAM IN BACKGROUND/
        .EVEN

LIST:   .BLKW   2               ;ARGUMENTS FOR THE EMT
JOBARG: .BLKW   8.              ;JOB PARAMETERS PASSED BACK HERE.

        .END    START
```

.GTLIN

2.4.20 .GTLIN

This request is used to collect a line of input from either the console terminal or an indirect command file, if one is active. This request is similar to .CSIGEN and .CSISPC in that it requires the USR, but no format checking is done on the input line. Normally, .GTLIN collects a line of input from the console terminal and returns it in the buffer specified by the user. However, if there is an indirect command file active, .GTLIN collects the line of input from the command file just as though it were coming from the terminal.

An optional prompt string argument is supported to allow the user to be queried for input at the terminal. (It is similar to the CSI's asterisk.) The prompt string argument is an ASCIZ character string in the same format as that used by the .PRINT request. If input is from an indirect command file and the SET TTT QUIET option is in effect, this prompt is suppressed. If SET TT QUIET is not in effect, the prompt is printed before the line is collected, regardless of whether the input comes from the terminal or an indirect file. The prompt appears only once. It is not reissued if an input line is cancelled from the terminal by CTRL/U or multiple DELETEs.

User programs that require nonstandard command format, such as the UIC specification for FILEX, can use the .GTLIN request to accept the command string input line. .GTLIN tracks indirect command files and the user program can do a pre-pass of the input line to remove the nonstandard syntax before passing the edited line to .CSIGEN or .CSISPC.

Macro Call: .GTLIN linbuf[,prompt]

where: linbuf is the address of the buffer to receive the input line. This is a user-specified area up to 81 decimal bytes in length. The input line is stored in this area and is terminated with a zero byte instead of (RET)(LF) (octal 15 and 12).

prompt is an optional argument and is the address of a prompt string to be printed on the console terminal. The prompt string has the same format as the argument of a .PRINT request.

NOTE

The only requests that can take their input from an indirect command file are .CSIGEN, .CSISPC and .GTLIN. The .TTYIN and .TTINR requests cannot get characters from an indirect command file; their input comes from the console terminal (or from a BATCH file if BATCH is running). The .TTYIN and .TTINR requests are useful for information that is dynamic in nature. For instance, the response to a system query when deleting all files with a .MAC file type or when initializing a disk is usually collected through a .TTYIN so that confirmation can be done interactively, even though the process may have been invoked through an indirect command file. However, the response to the linker's "TRANSFER SYMBOL" query would normally be collected through a .GTLIN, so that the LINK command could be invoked and the start address specified from an indirect file. Note also that if there is no active indirect command file, .GTLIN simply collects an input line from the console terminal by using .TTYINs.

Errors:

None.

Example:

This example prompts the terminal and accepts a line of input. If the first input character is in the range A through M, the example prints the line back at the terminal.

```
.TITLE  GTLIN.MAC
        .MCALL  .GTLIN, .PRINT,.EXIT

START:  .GTLIN  #LINBUF,#PROMPT ;GET A LINE OF INPUT
        MOVB    LINBUF,R0       ;PICK UP FIRST CHARACTER
        BEQ     EXIT            ;IF BLANK LINE-EXIT
        CMPB    R0,#'A          ;NOT BLANK, DOES IT BEGIN WITH A-M?
        BLO     START           ;IF LO, NO, JUST GET ANOTHER LINE
        CMPB    R0,#'M          ;MAYBE, CHECK HIGH LIMIT
        BHI     START           ;IF HI, NOT IN RANGE
        .PRINT  #LINBUF         ;OK, PRINT LINE
        BR      START           ;GO GET ANOTHER
EXIT:   .EXIT                   ;BACK TO MONITOR

PROMPT: .ASCII  /MY PROGRAM>/<200>

LINBUF: .BLKB   82.             ;LINE BUFFER
        .EVEN

        .END    START
```

.GVAL

2.4.21 .GVAL

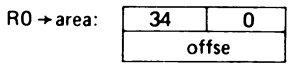
This request returns a monitor fixed offset value in R0 where it can be accessed by the user. This request must be used in the XM monitor to access monitor fixed offset locations, but it should also be used in other RT-11 monitors. The .GVAL request is a read-only operation and provides protection for information obtained from the monitor.

Macro Call: .GVAL area,offse

where: area — is the address of a two-word EMT argument block.

offse — is the displacement from the beginning of the monitor of the word to be returned in R0.

Request Format:



Errors:

Code	Explanation
0	The offset requested is beyond the limits of the resident monitor.

Example:

The following example demonstrates use of the .GVAL request by getting the monitor version number and update number from the resident monitor.

```
.TITLE  GVAL.MAC
        .MCALL  .GVAL,.EXIT
UPDATE  = 276                   ;OFFSET TO MONITOR
                                ;VERSION NUMBER
START:
        .GVAL   #AREA,#UPDATE   ;GET MONITOR VERSION
                                ;NUMBER AND UPDATE
                                ;IN R0
        MOVB    R0,MONVER       ;STORE VERSION #
        SWAB    R0              ;UPDATE TO LOW BYTE
        MOVB    R0,MONUPD       ;STORE UPDATE #
        .EXIT
MONVER: .BLKB                   ;MONITOR VERSION #
MONUPD: .BLKB                   ;MONITOR UPDATE #
AREA:   .BLKW   2
        .END    START
```


2.4.22 .HERR/.SERR

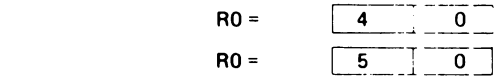
.HERR and .SERR are complementary requests used to govern monitor behavior for serious error conditions. During program execution, certain error conditions can arise that cause the executing program to be aborted (see Table 2-3). Normally, these errors cause program termination with one of the ?MON- error messages. However, in certain cases it is not feasible to abort the program because of these errors. For example, a multi-user program must be able to retain control and merely abort the user who generated the error. .SERR accomplishes this by inhibiting the monitor from aborting the job. Instead, it causes an error return to the offending EMT to be taken. On return from that request, the carry bit is set and byte 52 contains a negative value indicating the error condition that occurred. In some cases (such as the .LOOKUP and ENTER requests), the .SERR request leaves channels open.

.HERR turns off user error interception; it allows the system to abort the job on fatal errors and generate an error message. (.HERR is the default case.)

Macro Calls: .HERR

.SERR

Request Formats:



Errors:

Table 2-3 contains a list of the errors that are returned if soft error recovery is in effect. Traps to 4 and 10, and floating point exception traps are not inhibited. These errors have their own recovery mechanism.

Table 2-3
Soft Error Codes (SERR)

Code	Explanation
-1	Called USR from completion routine.
-2	No device handler; this operation needs one.
-3	Error doing directory I/O.
-4	.FETCH error. Either an I/O error occurred while reading the handler, or tried to load it over USR or RMON.
-5	Error reading an overlay.
-6	No more room for files in the directory.
-7	Illegal address (FB only); tried to perform a monitor operation outside the job partition.
-10	Illegal channel number; number is greater than actual number of channels which exist.
-11	Illegal EMT; an illegal function code has been decoded.

Example:

```
.TITLE  MERR.MAC
;THIS EXAMPLE CAUSES A NORMALLY FATAL ERROR TO GENERATE ERRORS
;BACK TO THE USER PROGRAM. THE ERROR RETURNED IS USED TO PRINT
;AN APPROPRIATE MESSAGE.
        .MCALL  .FETCH,.ENTER,.MERR,.SERR
        .MCALL  .EXIT,.PRINT

ST:     .SERR                   ;TURN ON SOFTWARE ERROR
                                ;RETURNS
        .FETCH  #HDLR,#PTR      ;GET A DEVICE HANDLER
        BCS     FCHERR
        .ENTER  #AREA,#1,#PTR   ;OPEN A FILE ON CHANNEL 1
        BCS     ENERR
        .MERR                   ;NOW PERMIT ?M-ERRORS.
        .EXIT


FCHERR: MOVB    @#52,R0         ;WAS IT FATAL
        BMI     FTLERR          ;YES
        .PRINT  #FMSG           ;NO... NO DEVICE BY THAT NAME
        .EXIT
ENERR:  MOVB    @#52,R0
        BMI     FTLERR
        .PRINT  #EMSG
        .EXIT
FTLERR: NEG     R0              ;THIS WILL TURN POSITIVE
        DEC     R0              ;ADJUST BY ONE
        ASL     R0              ;MAKE IT AN INDEX
        MOV     TBL(R0),R0      ;PUT MESSAGE ADDRESS INTO R0
        .PRINT                  ;AND PRINT IT.
        .EXIT


TBL:    M1                      ;CAN'T OCCUR IN THIS PROGRAM
        M2                      ;NO DEVICE HANDLER IN MEMORY
        M3                      ;DIRECTORY I/O ERROR
        M4                      ;FETCH ERROR
        M5                      ;IMPOSSIBLE FOR THIS PROGRAM
        M6                      ;NO ROOM IN DIRECTORY
        M7                      ;ILLEGAL ADDRESS (F/B)
        M10                     ;ILLEGAL CHANNEL
        M11                     ;ILLEGAL EMT

M1:                             ;CAN'T OCCUR IN THIS PROGRAM
M2:     .ASCIZ  /NO DEVICE HANDLER/
M3:     .ASCIZ  "DIRECTORY I/O ERROR"
M4:     .ASCIZ  /ERROR DOING FETCH/
M5:                             ;NOT APPLICABLE TO THIS PROGRAM
M6:     .ASCIZ  /NO ROOM IN DIRECTORY/
M7:     .ASCIZ  /ADDRESS CHECK ERROR/
M10:    .ASCIZ  /ILLEGAL CHANNEL/
M11:    .ASCIZ  /ILLEGAL EMT/
FMSG:   .ASCIZ  /FETCH FAILED/
EMSG:   .ASCIZ  /ENTER FAILED/
        .EVEN
HDLR:   .BLKW   300             ;LEAVE 300 (OCTAL) FOR HANDLER
PTR:    .RAD50  /DT4/           ;DEVICE AND FILE NAME.
        .RAD50  /EXAMPL/
        .RAD50  /MAC/
AREA:   .BLKW   4               ;EMT AREA
        .END    ST
```

.HRESET

2.4.23 .HRESET

This request stops all I/O transfers in progress for the issuing job, and then performs an .SRESET (see Section 2.4.54). (.HRESET is not used to clear a hard-error condition.) Note that in the SJ environment, a hardware RESET instruction is used to terminate I/O, while in a FB environment, only the I/O associated with the job that issued the .HRESET is affected. All other transfers continue.

Macro call: .HRESET

Errors:

None.

Example:

See the example for .SRESET for format.

.INTEN

2.4.24 .INTEN

This request is used by user program interrupt service routines to:

1. Notify the monitor that an interrupt has occurred and to switch to system state.

2. Set the processor priority to the correct value.

The .INTEN request is not an EMT monitor request but rather a subroutine call to the monitor.

All external interrupts cause the processor to go to priority level 7. .INTEN is used to lower the priority to the value at which the device should be run. On return from .INTEN, the device interrupt can be serviced, at which point the interrupt routine returns with an RTS PC. It is very important to note that an RTI does not return correctly from an interrupt routine that specifies an .INTEN.

Macro Call: .INTEN prio[,pic]

where:

prio	is the processor priority at which the user needs to run the interrupt routine, normally the priority at which the device requests an interrupt.
pic	is an optional argument that should be non-blank if the interrupt routine is written as a PIC (position independent code) routine. Any interrupt routine written as a device handler must be a PIC routine and must use this argument.

Errors:

None.

Example:

See the example for .SYNCH.

2.4.25 .LOCK/.UNLOCK

.LOCK

The .LOCK request is used to keep the USR in memory for a series of operations. If all the conditions that cause swapping are satisfied, the part of the user program over which the USR swaps is written into the system swap blocks (the file SWAP.SYS) and the USR is loaded. Otherwise, the copy of the USR in memory is used, and no swapping occurs. A .LOCK request always causes the USR to be loaded in memory if it is not already in memory. The USR is not released until an .UNLOCK request is given. (Note that in an FB system, calling the CSI can also perform an implicit .UNLOCK.) A program that has many USR requests to make can .LOCK the USR in memory, make all the requests, and then .UNLOCK the USR; no time is spent doing unnecessary swapping.

In a FB environment, a .LOCK inhibits the other job from using the USR. Note that the .LOCK request reduces time spent in file handling by eliminating the swapping of the USR in and out of memory. .LOCK causes the USR to be read into memory or swapped into memory. After a .LOCK has been executed, an .UNLOCK request must be executed to release the USR from memory. The .LOCK/.UNLOCK requests are complementary and must be matched. That is, if three .LOCK requests are issued, at least three .UNLOCKs must be done, otherwise the USR is not released. More .UNLOCKs than .LOCKs can be issued without error.

Macro Call: .LOCK

Notes:

1. It is vital that the .LOCK call not come from within the area into which the USR will be swapped. If this should occur, the return from the .LOCK request would not be to the user program, but to the USR itself, since the LOCK function inhibits the user program from being re-read.

2. Once a .LOCK has been performed, it is not advisable for the program to destroy the area the USR is in, even though no further use of the USR is required. This causes unpredictable results when an .UNLOCK is done.

3. If a foreground job performs a .LOCK request while the background job owns the USR, foreground execution is suspended until the USR is available. In this case, it is possible for the background to lock out the foreground (see the .TLOCK request).

Errors:

None.

Example:

See the example following .UNLOCK.

.UNLOCK

The .UNLOCK request releases the User Service Routine (USR) from memory if it was placed there with a .LOCK request. If the .LOCK required a swap, the .UNLOCK loads the user program back into memory. There is a .LOCK count. Each time the user does a .LOCK, the lock count is incremented. When the user does an .UNLOCK, the lock count is decremented. When it goes to 0, the user program is swapped back in. See note 1.

Macro Call: .UNLOCK

Notes:

1. It is important that at least as many .UNLOCKs are given as .LOCKs. If more .LOCK requests are done, the USR remains locked in memory. It does no harm to give more .UNLOCKs than are required; those that are extra are ignored.

2. The .LOCK/.UNLOCK pairs should be used only when absolutely necessary when running two jobs in the FB system. When a job .LOCKs the USR, the other job cannot use it until it is .UNLOCKed. Thus, the USR should not be .LOCKed unnecessarily, as this can degrade performance in some cases.

3. In an FB system, calling the CSI with input coming from the console terminal performs an implicit .UNLOCK.

4. It is especially important that the .UNLOCK not be in the area that the USR swaps into. Otherwise, the request can never be executed.

Errors:

None.

Example:

The following example tries to obtain as much memory as it can (with the .SETTOP request). Most likely this does, in a background job, make the USR non-resident (unless a SET USR NOSWAP command is done at the keyboard), and swapping must take place for each .LOOKUP given. Using the .LOCK, the USR is brought into memory and remains there until the .UNLOCK is given.

The second .LOOKUP makes use of the fact that the arguments have already been set up at LIST. Thus, it is possible to increment the channel number, put in a new file pointer and then give a simple .LOOKUP, which does not cause any arguments to be moved into LIST.