OS-9 Quick Reference and Programmer's Guide *for* Professional OS-9/68000

Published by
FARNA ***Systems***

the FARNA Fox!

by F.G. Swygert

Distributed by:
FARNA *Systems*
Box 321
Warner Robins, GA 31099
Phone 912-328-7859

Published by FARNA Systems PB
First Edition
First Printing April 1994

OS-9 Quick Reference and Programmer's Guide for OS-9/68000

(Based on Version 2.3)

CONTENTS:

INTRODUCTION

One of the troublesome things about learning OS-9/68000 is that bulky manual. It takes up lots of desk space, and it's sometimes hard to find that one simple command needed to complete a project. And one has to refer to the manual a LOT when first starting out- more than one cares to admit I'm sure!

This little Quick Reference Guide was designed to get that manual off your desk and back on the bookshelf. It isn’t, however, a replacement for the full manual. Only a brief description of commands and codes are given, sometimes no more than the syntax for entering. Enough information is here to jog one’s memory and get back on track, but the manual will still have to be referred to for learning and heavy duty programming chores. Note also that this QRG is based on the generic Microware manuals. Some systems will have special key functions not listed here.

Any of you who have FARNA's QRG for CoCo OS-9 will be familiar with the layout and content of this OSK edition. Indeed, the same template was used for both! I hope you find this edition as helpful as the first. If you have never seen the CoCo edition, well, let's just say enough were sold to make this edition a very good opportunity!

F.G. Swygert

A Note on Redirection

OS-9 commands generally read from the keyboard and write to the current screen. Almost all of them, however, can be sent elsewhere, using the redirection symbols: < (input) > (output) >> (error output).

Some common redirections-

echo Hello>/w7 - would print 'Hello' on window 7
List file>/p - lists file to parallel printer
utility -? >>/p - would print the help file

(note that help info often follows the error path)

1 - The Shell

The OS-9 shell is the built-in command interpreter. It is also user configurable. The wildcards * (representing any string) and ? (representing any single character) may be used with any command (built-in or external).

Built-in Shell Commands

The following commands are built-in the shell. all other commands are external- they are separate utilities.

*	Comment- nothing following on the same line is processed. Used mostly in procedure files.
chd (path)	Changes default data directory to specified (path)
chx (path)	Changes default execution directory to specified (path)
ex (name)	Exit shell and execute program (name)
kill (ID#)	Abort process (ID#)
logout	Terminate current shell
set (options)	Set shell options specified
setenv (var)(val)	Set environment variable (var) to specified value (val)
unsetenv (var)	Remove environment variable (var) from environment
w or wait	Wait for all child processes to terminate

Shell Options

Options can be set by typing on the command line or by using the set command. A dash in front of the command turns it on, no dash turns off.

ex: **set -t** echoes input lines, **set t** does not echo input lines

-e=(file)	Print error messages from (file). If no file given, /dd/sys/errmsg is assumed. If turned off, only error numbers and brief descriptions will be listed.
-ne	Do not print error messages (default)
-l	Terminate login shell with logout command only.
-nl	Terminate login shell with ESC key or CTRL [(default)
-p	Display prompt
-p=(string)	Displays (string) as prompt. Default prompt is $.
-np	Do not display prompt
-v	Verbose mode- display message for each directory searched when executing a command.
-nv	Verbose mode off (default).
-x	Abort process upon error (default).
-nx	Do not abort process upon error.

The Shell Environment

There are eight shell environment variables. These dictate how the shell reacts to subsequent commands. All shells use the parent shells environment unless changed. Only the shell the changes are made in and subsequent shells are affected, not previous shells. The first four (PORT, HOME, SHELL, USER) are automatically set when logging onto a time-share system. They are set with the **set** or **setenv** commands for single user systems. Note that environment variables are case sensitive- use the proper case!

PORT	Name of the terminal, usually /tx (where x is a number)
HOME	Home, or default, directory. This is the users default data directory when logged on. Also the directory used when **chd** command is used with no directory name.
SHELL	First process executed when system is started.
USER	User name typed when prompted by **login**.
PATH	Specifies directories to search through for a command/ program when a path is not given. Directories must be separated by a colon (:). ex: /h0/cmds: /h0/sys: /h1/cmds
PROMPT	Defines the current prompt. Use an @ in the prompt to display the shell number (@ will be replaced by displayed shell sequence number).
_sh	Starting shell number. _sh0 will make the first shell number "0" (no number displayed), the next 1 (@ in prompt replaced with number 1), etc.
TERM	Specifies type of terminal being used. Terminal types are usually specified by manufacturers model number. Types vary with system supplier. Others may be specified using termcap. See manual for details.

1 - System Commands

Commands are given in bold capital letters. Items following in bold lowercase are required. Items enclosed in parentheses ( ) are optional. COMPLETE path lists must be used in paths and names (file and directory) or current is assumed (**path** is synonymous with in this booklet **pathlist**). Examples are in bold lowercase with an explanation. A question mark will display the syntax for that command.

ATTR (options) *filename* (permission) Examine or change file security attributes. filename is the name of the file to be examined or changed, including the complete path. If no options or permissions are given, current attributes for filename are listed. More than one filename can be specified on a single line. Wildcards may be used.

Options:

-a - do not display attributes

-x - search current execution directory only. Execution permission must be set for file to be found.

-z - reads filename from standard input

-z=(file) - reads filename from (file)

(continued on next page)

ATTR *(continued from previous page)*
Permission:
d - file is a driectory
e - only owner can execute
r - only owner can read
s - non-shareable file
w - only owner can write
A “p” in front of e,r, or w means anyone (public) can access file. A minus sign (-) turns permission(s) on. -n turns permission(s) off.
attr file -werpwpepr Gives permission for owner and public (anyone) to read, write, or execute "file". **attr -npwpepr *** turns public permissions off for all files in the current data directory

BACKUP (options)(device1)(device2) Backup data from one drive to another. If no drive specified, /d0 to /d1 assumed. If only device1 specified, single drive assumed. Both disks (source and destination) must be formatted the same.
Options:
-b(#)k - memory used in kilobytes. Default is 4k.
-r - continue backup if read error is encountered.
-v - verify off
backup -v -b40 /d1 /d0 backup, no verify, use 40K buffer, from /d1 to /d0

BINEX (options) (path1) (path2) Converts binary data file in path1 to Motorola S-record file in path2.
Options:
-a(#) - #= load address
-s(#) - #= S-record type number
-x - search for path1 in current execution directory
S-record Types:

1 - Use two byte field address	7 - Terminate blocks of S3 records
2 - Use three byte field address	8 - Terminate blocks of S2 records
3 - Use four byte field address	9 - Terminate blocks of S1 records

BREAK Stops all processes and passes control to the ROM debugger. Used ONLY for system debugging. If called, the system console must be used t ocommunicate with the debugger. Resume operation with debugger g[0] command

BUILD filename Creates an ASCII text file by copying keyboard input to filename. Writes to file after enter is pressed. Press enter with no text or enter EOF character (usually ESC key) to close file.
build /d1/TEXT/textfile Copies every keypress to an ASCII file on /d1, TEXT directory, named "textfile".

CFP (options) (path1) (path2) (etc.) Creates the temporary procedure file path1 in the current data directory, and invokes the shell to execute the procedure. Path2, etc., is the file(s) t obe executed by path1. All asterics (*) are replaced with (path2)(etc.) unless proceeded by a tilde (~).
Options:
-d - delete temporary file when done (default)
-nd - do not delete temporary file when done
-e - execute proceudre file (default)
-ne - do not execute procedure file, dump to standard output
-s=(string) - read (string) instead of procedure file. If string contains shell
commands, entire option should be in quotes.
-t=(path) - create temporary file at path, not current data directory
-z - read file(s) from standard input, not path2
-z=(file) =- read file(s) from (file), not path2
cfp "-s=list * >/p" file1 file2 file3 lists the three named files to the printer.
cfp list.p file1 file2 file3 follows the instructions in procedure file list.p using the three named files.

CHD (path) Changes current data directory to directory named in path. If no file specified, path specified in HOME environment variable is used.

CHX path Changes current execution directory to directory named in path.

CMP file1 file2 (options) Compares binary values of data in the two files specified. Displays address, hexadecimal value, and ASCII character of different bytes if encountered. Summary of differences displayed.
Options:
-b=(#)k - specifies amount of memory to use (4K is default)
-s - Silent mode. Stops when first difference encountered and error message
displayed.
-x - searches current execution directory for both files.
cmp -b=10k -x file1 file2 uses a 10K buffer, searches the current execution directory for file1 and file2, then compares the files (if found)

CODE Prints hex value of all characters input after command execution. Unprintable characters will display as a period. CTRL E or CTRL C abort command.

COMPRESS filename (options) Reads filename and writes a new file of the same name with "_comp" appended to it in compressed format. Use on ASCII text files only! Compressed file is about 30% smaller than original
(See EXPAND on page 14).
Options:
-d - delete original file
-n - create a new output file
-z - read filename from standard input
-z=(file) - read filename from file
compress filename creates a compressed file named "filename_comp". Original file is retained.

COPY path1 path2 (options) Copies from one file or device to another. Path1 is complete path and name of source, path2 complete path and name of destination.
Options:
-a - abort copy on error
-b=(#)k - memory used in kilobytes. Default is 4K
-p - do not print list of multiple files
-r - overwrite existing file
-v - verify new file
-w=(dir) - copy one or more files to directory (dir). Displays "continue (y/n)" on error unless -a also specified.
-x - use current execution directory for path 1
copy /d0/text1 /d1/text2 -b=40k copies text1 on /d0 to text2 on /d1 using a 40K buffer. **copy /d0/*.* -w=/h0/TEXT** copies all files on /d0 to /h0/TEXT.

COUNT (options) filename counts number of characters, lines, and/or words in a file.
Options:
-b - counts and gives breakdown of character occurrences.
-c - count chaacters
-l - count lines
-w - count words
-z - read filename(s) from standard input
-z=(file) - read filname(s) from (file)
count -bclw filename displays the number of times each character occurs, the number of characters, lines, and words in filename.

DATE (options) Displays current system date and time.
Options:
-j - display Julian date and time
-m - display military (24 hour) date and time
date -j displays 359,1995 2:30:00pm (25 DEC, 1995, 2:30pm)
date -m displays December 25, 1995 Monday 14:30:00 (same as above)

DCHECK (options) drive Checks disk file structure in specified drive.
See manual before using -r option!
Options:
-d=(#) - print path to directory (#) deep
-r - maps a cluster found in file structure but not in bitmap in the bit map or the opposite. Prompts off/on for each bit found.
-y - used with -r. Turns off prompts.
dcheck /d1 displays volume name, total sectors, bytes in allocation map, sector per cluster, starting sector of root directory, number of sectors used for id, allocation map, and root directory, number of directories and files, and the number of bytes used and remaining free on disk.

DEINIZ (options) device1 (device2) (etc.) Deinitializes and detaches specified device(s). One should only DEINIZ a device that was initialized with INIZ.
Options:
-z - read device names from standard input
-z=(file) - read device names from (file)
deiniz /p2 removes serial printer from system.

DEL (options) file1 (file2) (etc.) Delete (erase) the specified file(s). Uses current directories unless otherwise specified. Can only DEL files you have permission to write to (see ATTR, page 7).
Options:
-p - prompt before deleting each file
-x - searches for file in current execution directory
-z - read file from standard input
-z=(file) - read file from (file)
del -p * prompts y/n before deleting every file in the current data directory

DELDIR (options) directory Deletes specified directory along with all associated subdirectories and files. Prompts "Delete, List, or Quit (d, l, or q) ?". After listing files, a "delete ? (y/n)" prompt will be displayed.
Options:
-f - delete files even if write permission is not set
-q - deletes all possible files and subdirectories with no prompts
-z - read directory from standard input
-z=(file) - read directory from (file)
deldir TEXT deletes directory TEXT and all subdirectories/files within it.

DEVS Displays the system's device table. Displays device descriptor, driver, file manager, address of static storage, and use count in that order.
The first lines of the **devs** command display may look like this:

```
 Username   OS-9/68K (ver.#)   (max. number of devices)
   Device        Driver        File Mgr      Data Ptr       Links
  -----------   -----------   -------------  -------------  ------------
   term          sc8x30        scf            $003be3f0            3
  (etc.)
```

DIR (options) (path) Display contents of directory named in path. If no options or path specified, current data directory is assumed.
Options:
-a - displays all files, including those starting with a period (.)
-d - adds a slash (/) to end of all directory names
-e - displays size, address, owner, permissions, and last date/time modified.
-n - displays directories only
-r - recursively displays directories with filenames
-r=(#) - recursively displays directories with filenames # directories deep
-s - displays unsorted directory/filename listing
-u - displays unformatted directory/filename listing
-x - current execution directory
-z - read directories to diplay from standard input
Options can be used without path and vice-versa.
dir -e /d0 displays directory of /d0 listing size, address, etc., of each file
dir -n *x /d0 displays only directories on /d0 ending with "x"

DSAVE (options) (path) Copies (backs up) all files from the current data directory to path. Current data directory assumed if no path given.
Options:
-a - do not copy files beginning with a period (.)
-b# - amount of memory for copy to use in kilobytes
-d - copy only files with most recent dates if the same name
-d=(date) - copies only files with date after (date) specified
-e - execute output immediately
-i - indents directory levels
-l - don't process directories below current level
-m - don't include MAKDIR in path
-n - don't load COPY or CMP if -v specified
-o - use OS9GEN to make destination a bootable disk
-o=(name) - use specified (name) for bootfile (see -o, above)
-r - write source file over file in destination with same name
-s - skip file on error
-v - verify copy by forking to CMP after each file
dsave >/d1/makecopy make a procedure file on /d1 called make copy that will copy all files in current data directory to another destination. To copy, chd to the destination path, then run makecopy.
chd /d0; dsave -eb40 /d1 change data directory to source drive; copy using 40K buffer all files on /d0 to /d1 immediately (no file created).

DUMP (options) (path) (address) Displays formatted listing of the physical data content of (path) starting at hexadecimal (address), if address is specified. If no address, beginning of file is used. If no path, keyboard input displays in hexadevimal.
Options:
-c - do not compress duplicate lines
-x - (path) is an execution directory. User must have execute permission
dump /d0/file displays:

```
(starting    data bytes in hexadecimal format)          (data bytes in
address)                                                    ASCII format)
Addr  0 1    2 3   4 5   6 7   8 9   A B  C D  E F       0 2 4 6 8 A C E
 -----  ------  ------  ------ ------ ------ ------  ------  ------
 0000 87CD 0038 002A F181 2800  2E00 3103  FFE0          . m . 8 .*q . (. . . 1. . .
(etc.)
```

ECHO (options) text Prints text to standard output, usually the current screen. Can de redirected to any device). Used to create messages in procedure files or to send initialization strings to a terminal.
Options:
-n - separate text with carriage return
-r - do not send carriage return at end of text (after <enter>)
-z - read text from standard input
-z=(file) - read text from (file)
echo "hello" displays "hello" on screen
echo "hello"; list textfile >/p prints "hello" on screen; prints textfile.

EDT (options) filename Built-in text editor (line oriented). Loads filename and displays last line. If no file found, a new one will be created. A ? prompt will be displayed. First character on a line is interpreted as a command. Use a space as first character if text is to be inserted.
Options:
-b=(#)K - opens editor with buffer (#)K larger than existing file, (#)K size for new file. Default is 2K.
Commands:
# - move cursor to line number #
ESC or Q - close file and exit editor
ENTER - move down one line
+ # - move down # lines (default is 1)
- # - move up # lines (deafult is 1)
SPACE - insert line
d (#) - delete current line. A number deletes # of lines beginning w/ current.
l # - list # of lines. May be positive or negative (up or down list - default=1)
l * - list all lines in file
s (*) / (string) / - search for (string). If asterics (*) used, all occurrences of string will be found, otherwise only first occurrence. Any character may be used for delimiters, not just slash (/).
c (*) / (string1) / (string2) - search for (string1), replace with (string2). If asterics (*) used, all occurrences of string will be found and replaced, otherwise only first occurrence. Any character may be used for delimiters, not just slash (/).
edt -b=40K textfile will create (or load from current data directory) a text file named "textfile" with a 40K buffer.

EVENTS Displays list of active processes currently on system. Gives ID number, name, value of event variable, wait increment, signal increment, and link count.

EX filename (arguments) Terminates current shell then runs filename. If no other shell open and no filename given, OS-9 will crash. Must always be the last command on a line.
ex basic starts Basic without a shell, saves memory.

EXPAND (options) filename Restores compressed files to their original size
Options:
-d - delete compressed file when finished
-n - send output to a file instead of standard output. Expanded file has "_exp" extension.
-z - read filename from standard input (default)
-z=(file) - read filename from (file)
expand -d file_comp will decompress compressed file_comp then delete the compressed file (file_comp). See COMPRESS, page 9.

FIXMOD (options) modname Update and verify module modname parity and CRC. Must have write access to modname. Can also be used to change module attrbutes.
Options:
-u - update module CRC or parity
-ub - fix sys/rev field in packed Basic subroutine module
-x - look for module in current execution directory
-z - read modname from standard input (default)
-z=(file) - read modname from (file)
fixmod xc checks parity and CRC for xc. **fixmod -u xc** checks and updates same.

FORMAT device (options) Formats device using options.
Options:
-c=(#) - format with (#) sectors per cluster, must be power of 2 (default is 1)
-1=(#) - format with interleave value (#)
-np - no physical format
-nv - no physical verification
-r - no prompts
-t=(#) - format (#) of tracks only
-v=(diskname) - format using (diskname). Can specify up to 32 characters. If spaces in diskname, option and diskname must be in quotes.
-sd - single density (floppy only)
-dd - double density (floppy only)
-ss - single sided (floppy only)
-ds - double sided (floppy only)
format /d1 -r -ss -t=35 "-v=boot disk" formats a single sided 35 track disk in /d1 named boot disk without any prompts interrupting.
FREE drive Displays number of unused sectors, name, date created, cluster size, and largest free block of disk in specified drive. Default drive used if none specified.

FRESTORE (options) (path) Restores files from multiple tape or disk backups (see fsave, next). If no path given, /mt0 is assumed. Restore must start with the last backup disk/tape, as that volume has the backup index.
Options:
-b=# - size of buffer in K
-c - verifies files without returning to shell
-d=drive - specifies source drive (default is /mt0)
-e - display all files in index as read from source
-f=file - restore from file
-i - display backup name, creation date, owner number, volume, and if index is on the volume. No restoration is done, frestore terminates after display.
-p - suppress prompt for first volume
-s - restore all files from source without interactive shell
-t=directory - specifies alternate dir for temp index file (default=currentdata)
-v - same as -i except index is not noted
-x=# - specifies memory in K for temporary file.
frestore will restore tape(s) from /mt0 to the current data directory
frestore -d=/d0 -e /h0/NEW will restore from /d0, displaying each file as read, to the NEW directory on /h0

FSAVE (options) (directory) Backsup directories over several disk or tape volumes. If no path given, a level 0 backup of the current directory on /mt0 will be attempted. Logical backup name, date, owner, bytes written, number of files, number of volumes, and index volume will be displayed when finished.
-b=# - size of buffer in K
-d=device - specifies target device (default is /mt0)
-e - do not display path as backed up
-f=file - save to file
-g=# - backup files by group number # only

(continued on next page)

FSAVE *(continued from previous page)*
-l=# - backup level #. A higher level number backsup only files made since the next lower number.
-m=path - specifies the path for the backup log file (default is /h0/sys/backup_dates)
-p - no mount volume prompt for first volume
-s - display path of all files needing backup without backing up files
-t=directory - specifies alternate location for temporary index file
-u=# - backup only files bowned by user #
-v - do not verify disk volume when mounted
-x=# - specifies memory in K for temporary file
fsave -l=0 -d=/d0 does a level 0 backup of the current directory to /d0

GREP (options) string (file1) (file2) (etc.) Searches (file) for string. Special modifiers may be used for string. If multiple files given, file name searched is displayed.
Modifiers:
Period (.) - match any ASCII character (except carriage return); ab.c will find abdc, abxc, etc.
Tilde (~) - match string only at beginning of line; ab will find abcd, abxy, etc.
Square Brackets ([ ]) - define a range of characters for string;
[a-g] will match letters a-g; [h-ma-g] will match h-m and a-g
[~a-g] will match all characters except a-g
Asterics (*) - modifies preceding single character so that zero or more occurrences of the single character; a* will find a, aaa, aaaaa, etc.
Dollar Sign ($) - match string only at end of line; ab$ will find cab, xyzab, etc.
Backslash (\) - allow search for special characters, such as \t (tab), \n (new line), \l (line feed), \b (backspace), \f (form feed).
Options:
-c - count number of matching lines
-e=(string) - same as string in command line
-f=(file) - read string from (file)
-l - print name of file with matching line only
-n - print line number where string found
-s - do not display matching lines
-v - print all lines except those with matching string
-z - read file(s) from standard input
-z=(file) - read file names from (file)
NOTE: -l and -n; -n and -s cannot be used together.
grep abc file1 will find all lines containing the string abc
grep -c abc[~d-g] file1 file2 will find all occurences of abc followed by any characters except defg in file 1 and file2, and will count the number of matching lines

HELP command1 (command2) (etc.) Displays help file for specified command(s). File "helpmsg" must be in /dd/SYS directory. Many third party utilities have a built in help file. Use utilityname -? to view help for most utilities, third party and standard.

IDENT (options) file1 (file2) (etc.) Displays header information for file or module name (size, owner, CRC, parity, edition, type/language, attributes/revision, access permission; for program modules also displays execution offsert, data size,stack size, initialized data offset, offset to data reference list).
Options:
-m - assume name is a module in memory
-q - quick mode- only one line per module
-s - only display bad CRCs
-x - assume file name is in execution directory
-z - read file from standard input
-z=(file) - read file from (file)
ident -m dir displays info for dir command in memory

INIZ (options) device1 (device2) (etc.) Initializes (attaches) the specified device driver(s). Link count will be incremented if device is already attached.
Options:
-z - read device from standard input
-z=(file) - read device from (file)
iniz /p2 initializes a newly attached serial printer

IRQS Displays system's IRQ polling table in the following order: exception vector, priority, hardware port address, driver's static storage address, interrupt routine's entry point, driver name, device descriptor name.

KILL processID1 (processID2) (etc.) Terminates specified processID number. Can only terminate a process with your user number attached. Attached to shell, not in CMDS directory. Process ID of 0 will kill all processes owned by user.

LINK (options) module1 (module2) (etc.) Increases module link count by one. When a module is loaded, link count is 1. Count becomes 2 when module is run. When finished, count drops by 1, but module remains. Modules run from disk only has a count of 1, and will be dropped as soon as it's finished. Can switch to another window and link a module rather than reloading. Once the count is reduced to 0, module "disappears" from memory and must be re-loaded.
Options:
-z - read module from standard input
-z=(file) - read module from (file)
link format format compress increases the link count of format by two, of compress by one. Note that format and compress must already be loaded.
LIST (options) file1 (file2) (etc.) Lists the contents of specified text file(s). Can list to screen or other device. May be redirected.
Options:
-z - read file from standard input
-z=(file) - read file from (file)
list /d1/textfile lists "textfile" on /d1 to screen
list /d1/textfile >/p& lists "textfile" on /d1 to the printer as background process

LOAD (options) module1 (module2) (etc.) Loads module(s) specified into memory. Current execution directory assumed unless specified.
Options:
-d - load module from current data directory
-l - print pathlist of module to be loaded
-z - read module from standard input
-z=(file) - read module from (file)
load format places the format command in memory for fast execution

LOGIN (name) (, password) Provides security for timeshare systems. Requests username and password (if not given with command) and checks against validation file. Automatically sets usernumber, execution and data directories, and executes a program in password file (usually shell). Automatically called by TSMON.

LOGOUT Terminates current shell. If current shell was activated by LOGIN, the ,logout procedure will be executed.

MAKDIR (options) directory Makes directory in current data directory unless path is specified. As a general rule, all OS-9 directories use all uppercase letters in the name, filenames are all lowercase or mixed.
Options:
-x - create directory in current execution directory
-z - read directory from standard input
-z=(file) - read directory from (file)
makdir /d1/CMDS creates a CMDS directory on /d1

MAKE (options) (file1) (file2) (etc.) (macro) Examines date of file(s) and file(s) used to create. If file(s) used to create specified file(s) have newer dates than specified file(s), specified file(s) will be updated. Generally used for compiling high level languages and updating source files, but may be used for any files dependant on other updated files. General file updating assumed here. Case dependant for directory and file names.
-f - read makefile from standard input
-f=(file) - reads makefile from (file)
-i - ignore errors
-n - display commands but do not execute
-s - execute commands without echo to screen
-t - update dates but not files
-u - run make regardless of file dates
-z - read file(s) from standard input
-z=(file) - read file(s) from (file)

MDIR (options) (module1) (module2) (etc.) Displays names of modules currently in memory. If (module#) is used, only that module name will appear if found.
Options:
-a - display language written in instead of type
-e - display extended module directory; lists physical address, size, owner, revision level, user count, and type (language with -a)
-t=(type) - display only modules of specified (type)
-u - display unformatted listing (generally used for piping output, etc.)

(continued on next page)

MDIR *(continued from previous page)*
mdir -e will display:

```
  Addr    Size      Owner    Perm   Type    Revs    Ed#     Lnk     Module name
002600   12136      12.3      ffff      Sys    8000      9          0     kernel
(etc.)
```

MERGE (options) (file1) (file2) (etc.) Copies file(s) to standard output path. Output can be redirected to any device. If no redirection specified, files will be listed to standard output.
Options:
-b=(#) - size of buffer used (default is 4K)
-x - search current execution directory
-z - read file from standard input
-z=(file) - read file from (file)
merge -b=32k file1 file2 >file3 combines files1&2 into file3 in the current data directory using a 32K buffer.

MFREE Displays beginning size of unassigned RAM blocks. -e displays number of free segments, start address and size of each segment.

MODED (options) (module) Used to change Init module and device descriptor modules. moded.fields file must be in sys directory.
Comands:
(E)DIT - edit current module. If no module was specified from command line, the editor will prompt for a module name when invoked. The name of a field and its current value and a prompt for a new value will be displayed. Type in new value or one of the following:
  dash (-) - redisplays last field
  period (.) - leave edit mode
  question (?) - list edit commands
  double question - list description of current field
  enter - leave displayed value unchanged
(F)ILE - open a file of modules
(L)IST - list contents of current module
(M)ODULE - find module in file
(W)RITE - update module CRC and write to file
Q(UIT) - quit moded and return to shell
$ - go to shell to run a command
Options:
-f=(file) - specifies file with one or more modules to be loaded into buffer

OS9GEN device (options) (module1) (module 2) (etc.) Creates a bootable disk by creating and linking OS9Boot file. Device is the drive with the disk to be made bootable. Can make a copy of an existing boot file, add modules to a boot file, or create a new boot file. When called and no options or modules listed, a file called tempboot is made and existing OS9Boot (if any) is renamed OldBoot. If an OldBoot file is present, it will be written over. Any modules listed will be copied to TempBoot then TempBoot is renamed OS9Boot. Should only be used on a newly formatted disk, as an error will occur if there is not enough contiguous space for OS9Boot.

(continued on next page)

OS9GEN *(continued from previous page)*
Options:
-b=# - memory used in kilobytes (default is 4K)
-q=(file) - renames (file) OS9Boot (good for renaming OldBoot)
-s=# - expand TempBoot to #K in size
-x - search execution directory for modules
-z - read module(s) from standard input
-z=(file) - read module(s) from (file)
To copy a boot file from one disk to another: **os9gen /d0 /d1/os9boot** (from /d1 to /d0)
Command line may be used to add devices to an existing boot file: **os9gen /d0 /d1/ os9boot -x newmod.L newmod.2** (copy os9boot from /d1 to /d0 and add newmods... newmods in current execution directory)
To use a bootlist: **os9gen /d0 -z=bootlist** (bootlist in current data directory)

PD Displays path from root directory to current data directory. pd -x displays path from root to current execution directory.

PR (file1) (file2) (etc.) (options) Lists formatted listing of file(s) to standard output. May be redirected. Listing will be separated into pages (with numbers). Page number, name of listing, and time/date listed will be at the top of each page. Default output is 1 line for header, 5 blank lines, 55 lines of text, then 5 blank lines (66 lines per page). Files may be listed in multiple columns on same page (see -c, -k, and -m below).
Options:
-c=(character) - use character as column separator (space is default)
-f - pad page with new lines instead of form feed
-h=(#) - set # of blank lines after header
-k=(#) - set # of columns for multi-column output
-l=(#) - set left margin (default is 0)
-m - print multiple files side by side in columns
-n=(#) - line number increment (default is 0)
-o - truncate lines longer than right margin (default is line-wrap to next line)
-p=(#) - lines per page, not including last 5 lines (default is 61)
-r=(#) - set right margin (default is 79)
-t - don't print header
-u=(name) - print (name) in header rather than file name
-x=(#) - starting page number (default is 1)
-z - read file from standard input
-z=(file) - read file from (file)
pr file1 >/p1 sends file1 to printer using default values
pr file1 file2 -m -k=2 >/p1 prints both files side by side on same page

PRINTENV lists defined environment variables (if any- see Shell, page 5) to standard output.

PROCS (options) Displays list of user's currently running processes at the moment the command is given. Process ID, parent process ID, process owner (group and user), priority, amount of memory being used, number of pending signals, status, CPU time used, elapsedtime since process began, and the process name and I/O paths are shown with no options.

(continued on next page)

PROCS *(continued from previous page)*
Options:
-a - alternate display. Displays process ID, parent ID, name and standard I/O paths, age based on priority and length of time waited for processing, number of service request calls made, number of I/O requests made, last system call made, number of bytes read, number of bytes written.
-b - displays regular and alternate information
-e - displays all processes of all users

QSORT (file1) (file2) (etc.) (options) Quick sort file(s) by specified field (field one is default). If no file(s) given, standard input assumed.
Options:
-c=(character) - (character) separates fileds (default is space). If an asterics(*), comma(,) or period(.) is used option and character must be in quotes.
-f=(#) - sort on field (#). Only one field number may be specified
-z - read file(s) from standard input
-z=(file) - read file(s) from (file)
qsort file1 -f=3 "-c=," sorts file1 by the third field, commas used in file1 as field separators

RENAME file newname Renames the file (or directory) to newname. -x searches for file beginning with the current execution directory.
rename /d1/cmds/util util2 renames util on the cmds directory of /d1 to util2

ROMSPLIT (options) file Divides a 16 or 32 bit ROM image file into eight bit files. Default is 16 bit image into two eight bit files named file.0 (even bytes) and file.L (odd bytes).
Options:
-q - split 32 bit image file into four eight bit files named file.0 (bytes 0,4,8,12, etc.), file.L (bytes 1, 5, 9, 13, etc.), file.2 (bytes 2, 6, 10, 14, etc.), file.3 (bytes 3, 7, 11, 15, etc.)
-x - read input from execution directory

SAVE (options) (module1) (module2) (etc.) Copies specified modules from memory into the current data directory. Created file(s) will have same name as module(s). Each module saved to its own file unless -f option specified, then all modules are saved together in the file given.
Options:
-f=(file) - save module(s) to (file)
-r - rewrite existing file(s)
-x - save file(s) to current execution directory
-z - read module(s) from standard input
-z=(file) - read module(s) from (file)
save dir copy copies the dir and copy command to the current data directory

SETIME (options) (y m d h m (s) (am/pm)) Sets system date and time to year (y), month (m), day (d), hour (h), minutes (m), and optionally seconds (s). Military time (24 hour) may be used or am or pm specified. Date and time can be separated by colons, semicolons, spaces, slashes, or commas. No separaters need be used, except a space between date and time. If no date/time given, a prompt will be displayed.
(continued on next page)

SETIME *(continued from previous page)*
Options:
-d - do not echo date/time when set
-s - read time from real time clock
setime 940501 1330 sets date and time to May 1, 1994, 1:30 pm
setime 940501 130 pm sets same

SETPR processID # Changes processID priority to #. Can set only for processes with your user number attached. Lowest is 1, highest 65535.
setpr 3 65535 gives process 3 highest possible priority

SLEEP # Puts current process to sleep for # of "ticks" or seconds (-s changes count to seconds rather than ticks). Duration of tick is system dependant. Default # is 0, causes process to sleep until signaled to wake up.

TAPE (device) (options) Provides access to tape controller from a terminal. If no device specified, /mt0 assumed. Options are executed in specific order: -z, -f, -b, -w, -e, -r, -o; so tape can be manipulated on one command line.
Options:
-b=(#) - skip (#) of blocks. If (#) is negative, tape skips back (default is 1)
-e=(#) - erase (#) of blocks
-f=(#) - skips (#) of tapemarks. Skips back if (#) is negative (default is 1)
-o - put tape off-line
-r - rewind tape
-s - specify size of tape block
-t - retension tape
-w=(#) - write (#) of tapemarks (default is 1)
-z - read device name(s) from standard input (default is /mt0)
-z=(file) - read device name(s) from (file) (default is /mt0)
tape /mt0 -r -o rewinds /mt0 then puts it off-line
tape -f=6 -e=8 -r skips 6 tapemarks forward, erases 8 blocks, then rewinds tape on default device /mt0

TEE (device 1) (device2) (etc.) Copies all text from standard input to devices listed. Generally redirected through a pipe (!).
echo System Down For Backup ! tee /t1 /t2 /t3 /t4 displays echoed message on all listed terminals. **dir -e ! tee /p1 /dir.text** will print a copy of the extended directory and place a copy in the current data directory as file dir.text

TMODE (-w=path#) (parameter1) (parameter2) (etc.) Displays or temporarily changes (current session only) terminal parameters. If no parameters given, current parameters will be listed to standard output. A parameter given with no value will be re-set to the default value. A parameter set to 0 will be turned off. Type, parity (par), character length (cs), stop bits, and baud parameters cannot be changed by tmode but will be displayed for information purposes. Path numbers are 0 (standard input), 1 (standard output), or 2 (error output). See **xmode**, page 22, for making permanent changes.

(continued on next page)

TMODE *(continued from previous page)*
Parameters:
bsb - backspace erases characters (default)
nobsb - backspace doesn't erase characters
bsl - backspace-space-backspace deletes terminal display line (default)
nobsl - disable backspace over a line
echo - input characters echo to screen (default)
noecho - disable echo
lf - turn on auto line feed to screen (default)
olf - turn off auto line feed
upc - uppercase characters only, converts all lower to upper.
noupc - upper and lower case characters (default)
pause - turn on screen pause when full, press space to resume
nopause - disable screen pause (default)
abort=hex - sets terminate character (default is $03, ctrl C)
baud=# - displays current baud rate
bell=hex - sets bell output character (default is $07)
bse=hex - sets output backspace chararacter (default is $08)
bsp=hex - sets input backspace chararacter (default is $08, ctrl H)
del=hex - sets input delete line character (default is $18, ctrl X)
dup=hex - sets character to duplicate last input line (default is $01, ctrl A)
eof=hex - sets input end-of-file chararacter (default is $1B, escape)
eor=hex - sets end-of-record input character (default is $0D, carriage return)
null=hex - sets number of nulls after carriage return (default is 0)
pag=# - sets # of video display lines, affects pause.
psc=hex - sets pause character (default is $17, ctrl W)
quit=hex - sets quit character (default is $05, ctrl E)
reprint=hex - sets reprint line character (default is $04, ctrl D)
type=hex - displays acia initialization values (parity, character size, number of stop bits)
par=x - displays parity as odd, even, or none
cs=# - displays character length in bits (8, 7, 6, or 5)
stop=# - displays number of stop bits (1, 1.5, or 2)
xon=hex - DC1 resume output character (default is $11, ctrl Q)
xoff=hex - DC2 stop output character (default is $13, ctrl S)
tabc=hex - tsb character (default is $09, ctrl I)
tabs=# - sets # of characters between tab stops (default is 4)
normal - sets all parameters to defaults
tmode xon=0 xoff=0 bell=0 turns xon, xoff, and bell off. tmode normal sets all parameters back to the default settings.

TOUCH (options) file1 (file2) (etc.) Updates the last modification date of file(s) to the current date. If specified file(s) not found, a file will be created with current date.
Options:
-c - do not create file if not found
-q - do not quit on error
-x - search current execution directory
-z - read file(s) from standard input
-z=(file) - read file(s) from (file)

TR (options) string1 (string2) (path1) (path2) Converts characters in string1 to characters in string2. Path1 is input (string1) and path2 is output (string2). If only one path given, input (string1) is assumed and output will be to standard output. If no paths listed, standard input and output is assumed. A dash (-) between characters specifies a range of characters. A back slash allows use of the following special characters: \t = tab, \n = new line, \l = line feed, \b = backspace, \f = form feed.
Options:
-c or -v - convert all ASCII characters to string2 except those listed in string1
-d - delete characters in string1 from string2
-s - squeeze consecutively repeated output characters into single characters
-z - read string(s) from standard input
-z=(file) - read string(s) from (file)
tr abc def /d0/text1 /d0/text2 changes all occurrences of abc in /d0/text1 to def on /d0/text2.
tr a-c d converts evey occurrence of abc to d in standard input, output sent to standard output.

TSMON (options) (terminal) Monitors idle terminals on a timesharing system and initiates a login sequence when an idle terminal is requested. Logoff by sending end-of-file character (usually escape). Up to 28 devices may be specified. More than one tsmon process may be running at once for more than 28 terminals. tsmon generates a logout message stating time this logon and total time for user.
Options:
-d - display ststistics (time, etc.) when ctrl\ ($1C) is typed
-l=program - fork to alternate login program
-p - display "welcome" message to each terminal being monitored
-r=program - fork to alternate shell program
-z - read terminal(s) from standard input
-z=(file) - read terminal(s) from (file)
tsmon -p /term /t1 /t2 /t3 /t4 prints welcome message on each of five listed terminals being monitored.

UNLINK (options) module1 (module2) (etc.) Reduces named module link count by one. Will be unloaded from memory once count reaches 0. If a module is named that wasn't loaded or is being used by another process, that process will crash, usually with module not found error. Modules that are part of a merged file cannot be unlinked except for first module in file, which is the "master" file. Unlink the master and the entire group count will be reduced. All files merged in the group will show a count of 0. The file just before the 0s is the master file (shows count for group).
Options:
-z - read module(s) from standard input
-z=(file) - read module(s) from (file)
unlink -z will wait for modules to be entered from standard input (usually keyboard)
unlink dir copy will reduce link count of dir and copy by one.

W Causes the shell to wait for the last child process to receive I/O before giving a prompt.

WAIT Causes the shell to wait for all child processes to end (terminate) before giving a prompt.

XMODE (options) device (parameter1) (parameter2) (etc.) Displays initialization parameters of any SCF-type device (screen, printer, RS-232 port, etc.) if no options listed. Changes initialization parameters to those listed when parameter list is included. Similar to TMODE, but XMODE updates remain as long as the computer is on during current session. TMODE only works on open paths so effects are gone once current path is closed. Parameters are same as TMODE (see for list, pages 19-20).
Options:
-z - read device(s) from standard input
-z=(file) - read device(s) from (file)
SPECIAL NOTE: Type, parity (par), cs (character length), stop (stop bits), and baud can be changed by xmode. deiniz the module to be changed, use xmode to change, then iniz the module for changes to take affect:
deiniz /p2
xmode baud=2400
iniz /p2

--(end of **System Commands** ***section)--***

2 - Special Keys

KEY(S)	FUNCTION
CTRL	control key
CTRL A	displays last line typed with cursor at end. Press ENTER to execute or edit by backspacing. Repeat to display edited line.
CTRL C	Aborts current program. Some programs intercept CTRL C, stops the current program, and allows a return to a menu or continuation of the program. In the shell, CTRL C converts the foreground process to a background process, provided no terminal I/O has begun.
CTRL D	redisplay command line
CTRL E	halts current program
CRTL H	moves cursor one space left (backspace key can be used)
CTRL W	temporarily halts video (scrolling). Any key restarts.
CTRL X	delete current line
ESCAPE or CTRL [	sends end-of-file to program receiving keyboard input. Must be first on a line.
ENTER	carriage return or execute current command line

4 - µMACS Editor Commands

The uMACS editor is a very powerful screen oriented text editor. Multiple buffers can be opened, allowing one to work on more than one file at once. Portions of one file may even be cut and pasted from one file to another or to another area of the original file. In fact, it is just short of a full fledged word processor. Many programmers use it as just that for short letters, some adding a text formatter for better printed appearance. It is not the purpose of this QRG to teach one to use the editor. For that one must refer to the manual.

To start uMACS type **umacs filename1 (filename2) (etc.) (option)** An unnamed buffer will be opened if filename is not given. It can be named and saved later if necessary. Key sequences are not case sensitive. ctrl = control key. Ctrl commands are executed by holding the ctrl key while pressing the following keys. esc = escape key. Esc commands are executed by pressing and releaseing the esc key and then pressing the following key. Exit with ctrl X ctrl C (prompts to save changed files) or esc M (saves all changed files before exiting).
Options:
-e - all files will be opened in edit mode (default)
-v - all files will be opened in view mode (may not be edited without changing modes)

There are 91 commands. They can be executed by the following key sequences or by typing esc X. upon typing ctrl X, the cursor will then move to the bottom of the window and wait for a command name to be typed. If the first or first few characters are typed and then enter pressed, umacs will dispaly the first part of the command beginning with the specified character(s), display a dash, and wait for the second part, which may be entered in the same fashion (some command have one part, others more than two). When a name is completed, it will be executed upon pressing enter. Commands with no key sequence must be entered with esc X.

Key sequences can be changed with the bind-to-key (esc K) and unbind-key (esc ctrl K) commands. Two character commands must begin with esc or ctrl X. To assign a command to a key sequence, press ctrl K (or use esc X bind-to-key). A ": bind to key" prompt will appear. Type the name of the command, a space, then the key sequence combination. To change an existing key sequence, first unbind the current or default key sequence with unbind-key (esc ctrl K) then use bind-to-key.

A file can be used to change command key sequences and build macros. Each time umacs is executed it looks for the file ".umacsrc" in the home directory. Each line is executed as umacs commands, one command per line. Other file names may be executed with the execute-file command. If a command requires input, the input must be supplied in the file in quotes.

(continued on next page)

uMACS Editor Commands *(continued from previoous page)*

Key and Help Commands:

Name	**Key Sequence**	**Command**
bind-to-key	esc K	define a key sequence
unbind-key	esc ctrl K	undefine a key sequence
execute-named-command	esc X	execute command by name
help	esc ?	open help buffer in view mode in top half of screen
describe-key	ctrl X?	prompts for key sequence, displays name or "not-bound"
describe-bindings	none	displays list of keys and commands
abort	ctrl G	abort command (only before executed)
execute-file (file)	none	executes file as umacs command file
exit-emacs	ctrl X ctrl C	prompts to save changed files then exits
quick-exit	esc Z	save all changed files then exit

Editing Modes:
mMACS has four editing modes besides the -e (edit, default) and -v (view only) options. A buffer may be opened in more than one mode. Modes are:
OVER - Overwrite mode, default is insert.
EXACT - All searches require exact case, default is case insensitive.
WRAP - Word wrap on. Default wraps characters, not words.
CMODE - Auto-indenting for writing C source code. Automatically turned on if buffer name ends in .c or .h.

Name	**Key Sequence**	**Command**
add-mode	ctrl XM	adds mode to current buffer. Prompts for mode
delete-mode	ctrl X ctrl M	delete mode from current buffer. Prompts for mode
add-global-mode	esc M	adds mode to every new buffer
delete-global-mode	esc ctrl M	deletes mode from all buffers

Macro Commands:

Name	**Key Sequence**	**Command**
begin-macro	ctrl X(	marks beginning of a macro
end-macro	ctrl X)	marks end of a macro
execute-macro	ctrl XE	executes a defined macro

(continued on next page)

uMACS Editor Commands *(continued from previoous page)*

File and Shell Commands:

Name	**Key Sequence**	**Command**
insert-file	ctrl X ctrl I	insert file at cursor
read-file	ctrl X ctrl R	read file into current buffer, deleting existing contents
find-file	ctrl X ctrl F	read file into new buffer
change-file-name	ctrl XN	name or rename file in current buffer
save-file	ctrl XS	save changed file
write-file	ctrl X ctrl W	write file to name given at prompt
i-shell	ctrl XC	fork to a shell (esc to return to umacs)
shell-command	ctrl X!	fork to shell, execute command given at prompt, return

Cursor Positioning:

Name	**Key Sequence**	**Command**
backward-character	ctrl B	move cursor 1 character back
forward-character	ctrl F	move cursor 1 char. forward
next-word	esc F	move cursor 1 word forward
previous-word	esc B	move cursor 1 word back
next-line	ctrl N	move cursor down 1 line
previous-line	ctrl P	move cursor up 1 line
next-paragraph	esc N	move cursor to next paragraph
previous-paragraph	esc P	move cursor to previous paragraph
next-page	ctrl V	next screen
previous-page	ctrl Z	previous screen
beginning-of-file	esc <	move cursor to file beginning
end-of-file	esc >	move cursor to file end
beginning-of-line	ctrl A	move cursor to line beginning
end-of-line	ctrl E	move cursor to line end
go-to-line	esc G	go to following line number

Inserting Text:

Name	**Key Sequence**	**Command**
insert-space	ctrl C	insert space to right of cursor
quote-character	esc Q	print following control char.
newline	ctrl M	insert line (same as enter)
open-line	ctrl O	insert new line character to right of cursor
new-line-and-indent	ctrl J or linefeed	insert new line and indent same as previous line
handle-tab	ctrl I	redefine or insert tab at cursor
insert-file	ctrl X ctrl I	insert file from directory at cursor

(continued on next page)

uMACS Editor Commands *(continued from previoous page)*

Deleting Text:

Name	**Key Sequence**	**Command**
delete-next-character	ctrl D	delete character at cursor
delete-previous-character	ctrl H, back-space, or delete	delete character to left of cursor
delete-next-word	esc ctrl D	delete word beginning at cursor
delete-previous-word	esc ctrl H or esc backspace	delete word from left of cursor to cursor
delete-blank-lines	ctrl X ctrl O	delete blank lines between text
kill-paragraph	esc ctrl W	delete paragraph at cursor
kill-region	ctrl W	delete marked block
kill-to-end-of-line	ctrl K	delete line from cursor
yank	ctrl Y	put last deleted item(s) in kill buffer

Search and Replace:

Name	**Key Sequence**	**Command**
search-forward	esc S (text)	move cursor forward to following text
search-reverse	esc R (text)	move cursor back to following text
hunt-forward	none	move cursor forward to next occurence of last text
hunt-backward	none	move cursor backward to next occurence of last text
replace-string	ctrl R (text1) (text2)	replace all occurences of text1 with text2
query-replace-string	esc ctrl R (text1) (text2)	prompt before replacing text1 with text 2

Text Blocks (regions):

Blocks marked with a beginning marker and the cursor (cursor marks end).

Name	**Key Sequence**	**Command**
set-mark	esc (period) or esc (space)	mark beginning of block
exchange-point-and-mark	ctrl X ctrl X	swap beginning of block with cursor position
copy-region	esc W	copy block to kill buffer
kill-region	ctrl W	delete block
case-region-lower	ctrl X ctrl L	change all marked letters to lower case
case-region-upper	ctrl X ctrl U	change all marked letters to upper case
yank	ctrl Y	paste kill buffer to cursor position

(continued on next page)

uMACS Editor Commands *(continued from previoous page)*

Text Formatting:

Name	**Key Sequence**	**Command**
case-word-capitalize	esc C	change letter at cursor to upper case
case-word-lower	esc L	change letters from cursor to end of word to lower case
case-word-upper	esc U	change letters from cursor to end of word to upper case
set-fill-column	esc (#) ctrl XF	set right margin to (#) spaces
fill-paragraph	esc O	reformat paragraph using new right margin
transpose-characters	ctrl T	transpose (swap) character at cursor with character to left of cursor

Buffer Commands:

Name	**Key Sequence**	**Command**
list-buffers	ctrl X ctrl B	list umacs buffers
select-buffer	ctrl XB	select buffer to be edited. Prompts for buffer. Will create new buffer if named doesn't exist.
name-buffer	esc ctrl N	prompts for new buffer name (change name)
next-buffer	ctrl XX	move to next buffer in list (first if in last)
buffer-position	ctrl X=	display current line number
delete-buffer	ctrl XK (buffer)	delete (buffer) from memory. Does not delete from disk.
execute-buffer	none	execute buffer as umacs procedure file

Window Commands:

Each buffer is displayed in a separate window consisting of one line to the entire screen (default is entire screen).

Name	**Key Sequence**	**Command**
split-current-window	ctrl X2	copy current window in a new window
next-window	ctrl XN	move cursor to next window
previous-window	ctrl XP	move cursor to previous window
move-window-up	ctrl X ctrl P	scroll current window up 1 line
move-window-down	ctrl X ctrl N	scroll current window down 1 line
scroll-next-up	esc ctrl Z	scroll next window up one page (one screen)
scroll-next-down	esc ctrl V	scroll next window down one page (one screen)
shrink-window	ctrl X ctrl Z	decrease size of current window
grow-window	ctrl XZ or ctrl X ctrl	increase size of current window
delete-other-windows	ctrl X1	delete all windows except current (where cursor is)

5 - OS-9 System Calls

System calls communicate between the OS-9 operating system and machine language programs. There are three categories of system calls: User-state, System-state, and I/O. The user-state is the normal program environment. User-state calls do not normally deal with system hardware. The system-state is the environment where system calls and interrupts are normally executed. System-state calls often deal with system hardware. I/O calls perform various I/O functions.

In the following listings, the system call is listed in bold followed by a brief description. If there is no OUTPUT or ERROR OUTPUT listed, then there are no such functions for that call.

User-State System Calls

F$Alarm - send a signal to calling process when specified time has elapsed.

INPUT:	**OUTPUT:**
D0.L=ID (or 0)	D0.L=ID
D1.W=function code	
D2.L=signal code	**ERROR OUTPUT:**
D3.L=time interval or time	CC=carry set
D4.L=date	D1.W=error code

A$Delete - removes any alarm that has not expired. If ID=0 all pending cancelled.

INPUT:	**ERROR OUTPUT:**
D0.L=ID (or 0)	CC=carry set
	D1.W=error code

A$Set - send signal after specified time has elapsed. Time specified in system clock ticks or 256ths of a second.

INPUT:	**OUTPUT:**
D0.L=reserved, must be 0	D0.L=ID
D1.W=function code	**ERROR OUTPUT:**
D2.W=signal code	CC=carry set
D3.L=time interval	D1.W=error code

A$Cycle - sends a recurring signal every set interval (system clcock ticks, 256ths/sec).

INPUT:	**OUTPUT:**
D0.L=reserved, must be 0	D0.L=ID
D1.W=function code	**ERROR OUTPUT:**
D2.L=signal code	CC=carry set
D3.L=time interval	D1.W=error code

A$AtDate - sends a signal at the specified date/time (to nearest second).

INPUT:	**OUPUT:**
D0.L=reserved, must be 0	D0.L=ID
D1.W=function code	
D2.L=signal code	**ERROR OUTPUT:**
D3.L=time (00hhmmss)	CC=carry set
D4.L=date (yyyymmdd)	D1.W=error code

A$AtJul - sends a signal at the specified Julian date/time (to nearest second).

INPUT:	**OUTPUT:**
D0.L=reserved, must be 0	D0.L=ID
D1.W=function code	
D2.L=signal code	**ERROR OUTPUT:**
D3.L=time (secs after midnight)	CC=carry set
D4.L=Julian day number	D1.W=error code

F$AllBit - set bits in allocation bit map. Bit numbers from 0 to one less than number of bits in map.

INPUT:	**ERROR OUTPUT:**
D0.W= # of first bit	CC=carry set
D1.W=number of bits to set	D1.W=error code
(A0)=base address of bit map	

F$CCtl - changes system instruction/data caches (if any). D0.L set to zero flushes cache. Only system-state processes and super-group processes may change cache(s). If bits are set, the following will occur (unset bits have no effect): 0 - enable data cache, 1 - disable data cache, 2 - flush data cache, 4 - enable instruction cache, 5 - disable instruction cache, 6 - flush instruction cache. All other bits are reserved and remain unset.

INPUT:	**ERROR OUTPUT:**
D0.L=reserved, must be 0	CC=carry set
	D1.W=error code

F$Chain - load & execute new primary module, no new process created.

INPUT:	**ERROR OUTPUT:**
D0.W=language/type code	CC=carry set
D1.L=additional memory size	D1.W=error code
D2.L=parameter size	
D3.W= # of I/O paths to copy	
D4.W=priority	
(A0)=name pointer	
(A1)=parameter pointer	

F$CmpNam - compare two names. Case insensitive. Two wildcards: ? matches single character, * matches string. Target name must be terminated with null byte.

INPUT:	**OUTPUT:**
D1.W=length of source name	CC=carry clear if match
(A0)=pointer to source	**ERROR OUTPUT:**
(A1)=pointer to target	CC=carry set
	D1.W=error code

F$CpyMem - copy external memory into user's buffer.

INPUT:	**ERROR OUTPUT:**
D0.W=process ID of owner	CC=C bit set
D1.L= # of bytes to copy	D1.W=error code
(A0)=address of memory to copy	
(A1)=destination buffer pointer	

F$CRC - calculate new or check existing module CRC.

INPUT:	**OUTPUT:**
D0.L=data byte count	D1.L=updated CRC accumulator
D1.L=CRC accumulator address	**ERROR OUTPUT:**
(A0)=pointer to data	CC=carry set
	D1.W=error code

F$DatMod - create data module.

INPUT:	**OUTPUT:**
D0.L=data size	D0.W=type/language
D1.W=attr/revision	D1.W=attr/revision
D2.W=access permission	(A0)=new name string pointer
D3.W=type language	(A1)=module data pointer (exec entry)
D4.L=memory color type	(A2)=module header pointer
(A0)=module name string pointer	**ERROR OUTPUT:**
(A2)=process descriptor to put	CC=carry set
module in	D1.W=error code

F$DelBit - clear allocation map bits.

INPUT:	**ERROR OUTPUT:**
D0.W= # of first bit to clear	CC=carry bit
D1.W= # of bits to clear	D1.W=error code
(A0)=address of bit map	

F$DExec - suspends process and executes a debugged child process.

INPUT:	**OUTPUT:**
D0.W=child process ID	D0.L= # of instructions executed
D1.L= # of instructions to exec.	D1.L= # of instructions not executed
D2.W= # of breakpoints	D2.W=exception occurred (offset if not 0)
(A0)=breakpoint list	D3.W=classification of word
ERROR OUTPUT:	D4.L=access address
CC=carry set	D5.W=instruction register
D1.W=error code	

**F$DExit -** terminates suspended child process created with F$DFork.

INPUT:	**ERROR OUTPUT:**
D0.W= ID of child to terminate	CC=carry set
	D1.W=error code

F$DFork - creates suspended child process for debugger control.

INPUT:	**OUTPUT:**
D0.W=module type/revision	D0.W=child ID
D1.L=additional stack space	(A0)=updated past module string
D2.L=parameter size	(A2)=child's registers in buffer
D3.W= # of I/O paths for child	**ERROR OUTPUT:**
D4.W=priority	CC=carry set
(A0)=name pointer or path	D1.W=error code
(A1)=parameter pointr	
(A2)=copy of child's register buffer	

F$Event -create, delete, and manipulate events.
INPUT:
D1.W=event function code
Output and **Error Output** depends on function code.
F$Event Function Codes:

Ev$Link - link to an existing event by name.

INPUT:	*OUPTUT:*
(A0)=name string pointer	D0.L=ID #
Di.W=0 (function code)	(A0)=updated past event name
ERROR OUTPUT:	
CC=carry set	
D1.W=error code	

Ev$UnLnk - unlink an event.

INPUT:	*ERROR OUTPUT:*
D0.L=ID #	CC=carry set
D1.W=1 (function code)	D1.W=error code

Ev$Creat - create a new event.

INPUT:	*OUTPUT:*
D0.L=initial event variable	D0.L=ID #
D1.W=2 (function code)	(A0)=updated past event name
D2.W=auto-increment for Ev$Wait	*ERROR OUTPUT:*
D3.W=auto-increment for Ev$Signal	CC=carry set
(A0)=name string pointer	D1.W=error code

Ev$Delet - delete an existing event.

INPUT:	*ERROR OUTPUT:*
(A0)=name string pointer	CC=carry set
D1.W=3 (function code)	D1.W=error code
OUPUT:	
(A0)=updated past event name	

Ev$Wait - wait for an event to occur.

INPUT:	*OUTPUT:*
D0.L=ID #	D1.L=event value
D1.W=4 (function code)	*ERROR OUTPUT:*
D2.L=minimum activation value	CC=carry set
D3.L=maximum activation value	D1.W=error code

Ev$WaitR - wait for a relative event to occur.

INPUT:	OUTPUT:
D0.L=ID #	D1.L=event value
D1.W=5 (function code)	D2.L=minimum activation value
D2.L=minimum activation value	D3.L=maximum activation value
D3.L=maximum activation value	
ERROR OUTPUT:	
CC=carry set	
D1.W=error code	

(continued on next page)

F$Event Function Codes: *(continued from previous page)*

Ev$Read - read an event value without waiting.

INPUT:	*ERROR OUTPUT:*
D0.L=ID #	CC=carry set
D1.W=6 (function code)	D1.W=error code
OUTPUT:	
D1.L=event value	
D1.L=event value	

Ev$Info - returns event information.

INPUT:	*OUTPUT:*
D0.L=ID # to begin search	D0.L=ID # found
D1.W=7 (function code)	(A0)=data returned in buffer
(A0)=pointer to buffer for event info	
ERROR OUTPUT:	
CC=carry set	
D1.W=error code	

Ev$Signl - signals an event occurrence.

INPUT:	*ERROR OUTPUT:*
D0.L=ID #	CC=carry set
D1.W=MS bit set	D1.W=error code
LS bit=8 (function code)	

Ev$Pulse - signals an event occurrence.

INPUT:	*ERROR OUTPUT:*
D0.L=ID #	CC=carry set
D1.W=MS bit set	D1.W=error code
LS bit=9 (function code)	
D2.L=event pulse value	

Ev$Set - set event variable and signal event occurrence.

INPUT:	*OUTPUT:*
D0.L=ID #	D1.L=previous event value
D1.W=MS bit set	*ERROR OUTPUT:*
LS bit=A (function code)	CC=carry set
D2.L=new event value	D1.L=error code

Ev$SetR - set relative event variable and signal an event.

INPUT:	*OUTPUT:*
D0.L=ID #	D1.L=previous event value
D1.W=MS bit set	*ERROR OUTPUT:*
LS bit=B (function code)	CC=carry set
D2.L=increment for event variable	D1.L=error code

(end of F$Event function codes)

F$Exit - terminates the calling process (process terminates itself).

INPUT:
D1.W=status to be returned to parent

OUTPUT:
Process terminated

ERROR OUTPUT:
CC=carry set
D1.W=error code

F$Fork - creates a new process which becomes a child of the calling process.

INPUT:
D0.W=module type/revision
D1.L=additioanl memory size
D2.L=parameter size
D3.W= # of I/O paths to copy
D4.W=priority
(A0)=module name pointer
(A1)-parameter pointer

OUTPUT:
D0.W=child ID
(A0)=updated module name

ERROR OUTPUT:
CC=carry set
D1.W=error code

F$GBlkMp - get copy of system free block map.

INPUT:
D0.L=start address
D1.L=buffer size (bytes)
(A0)=buffer pointer

ERROR OUTPUT:
CC=carry set
D1.W=error code

OUTPUT:
D0.L=minimum allocation size
D1.L= # of memory fragments
D2.L= total RAM found
D3.L=total free RAM
(A0)=fragment information

F$GModDr - get copy of system module directory.

INPUT:
D1.L= Max bytes to copy
(A0)=buffer pointer

OUTPUT:
D1.L= # of bytes copied

ERROR OUTPUT:
CC=carry set
D1.W=error code

F$GPrDBT - get a copy of the process descriptor block table.

INPUT:
D1.L= Max bytes to copy
(A0)=buffer pointer

OUTPUT:
D1.L= # of bytes copied

ERROR OUTPUT:
CC=carry set
D1.W=error code

F$GPrDsc 103F 18 - get copy of process descriptor.

INPUT:
D0.W=process ID
D1.W=bytes to copy

ERROR OUTPUT:
CC=carry set
B=error code

F$Gregor - converts Julian date to Gregorian date.

INPUT:
D0.L=time (secs since midnight)
D1.L=Julian date

ERROR OUTPUT:
CC=carry set
D1.W=error code

OUTPUT:
D0.L=time (00hhmmss)
D1.L=date (yyyymmdd)

F$Icpt - set signal intercept trap.

INPUT:
(A0)=address of intercept routine
(A6)=address passed to routine

OUTPUT:
Signal sent to process causes intercept to be called, process not killed.

F$ID - get process ID and user ID

INPUT:
None

ERROR OUTPUT:
CC=carry set
D1.W=error code

OUTPUT:
D0.W=process ID
D1.L=user ID
D2.W=priority

F$Julian - converts Gregorian date to Julian date

INPUT:
D0.L=time (00hhmmss)
D1.L=date (yyyymmdd)

ERROR OUTPUT:
CC=carry set
D1.W=error code

OUTPUT:
D0.L=time (secs since midnight)
D1.L=Julian date

F$Link - link to named memory module.

INPUT:
D0.W=type/language
(A0)=module name string pointer

ERROR CODE:
CC=carry set
D1.W=error code

OUTPUT:
D0.W=type/language
D1.W=attribute/revision label
(A0)=updated past module name
(A1)=module execution entry point
(A2)=module pointer

F$Load - load module(s) from file.

INPUT:
D0.B=access mode
D1.L=memory color type
(A0)=path strin pointer

ERROR OUTPUT:
CC=carry set
D1.W=error code

OUTPUT:
D0.W=type/language
D1.W=attributes/revision level
(A0)=updated beyond path name
(A1)=exec entry point of first module
(A2)=module pointer

F$Mem - change process data memory.

INPUT:
D0.L=memory size in bytes

ERROR OUTPUT:
CC=carry set
B=error code

OUTPUT:
D0.L=memory size in bytes
(A1)=pointer to new end of data +1

F$PErr - writes error message to standard path.

INPUT:
D0.W=error message path (0=none)
D1.W=error code

F$PrsNam - scan input string for valid OS-9 path name.

INPUT:	**OUTPUT:**
(A0)=name of string pointer	D0.B=path delimiter
ERROR OUTPUT:	D1.W=length of path
CC=carry set	(A0)=path pointer updated past "/"
B=error code	(A1)=address of last name char. +1

F$RTE - terminate a signal intercept routine and continue main program execution.
NO INPUT OR OUTPUT

F$SchBit - search memory allocation bit map for free memory block of specified size.

INPUT:	**OUTPUT:**
D0.W=start bit to search for	D0.W=first bit # found
D1.W= # of bits to find	D1.W= # of bits found
(A0)=bit map pointer	**ERROR OUTPUT:**
(A1)=end of bit map pointer +1	CC=carry bit
	D1.W=error code

F$Send - send signal to process.

INPUT:	**ERROR OUTPUT:**
D0.W=process ID	CC=carry set
D1.W=signal code	D1.W=error code

F$SetCRC - update the header parity and CRC of a module in memory.

INPUT:	**ERROR OUTPUT:**
(A0)=module pointer	CC=carry set
	D1.W=error code

F$SetSys - chnge or examine a system global variable.

INPUT:	**OUTPUT:**
D0.W=offset of variable to examine	D2.L=original variable value
D1.L=size of variable	**ERROR OUTPUT:**
D2.L=new value if change	CC=carry bit
	D1.W=error code

F$Sigmask -enable/disable signal mask.

INPUT:	**ERROR OUTPUT:**
D0.L=reserved, must be 0	CC=carry bit
D1.L=signal level	D1.W=error code

F$Sleep - temporarily turn process off.

INPUT:	**ERROR OUTPUT:**
D0.L=sleep time (ticks)	CC=carry set
OUPUT:	D1.W=error code
D0.L=remaining time if started early	

F$SPrior - change process priority.

INPUT:	**ERROR OUTPUT:**
D0.W=process ID	CC=carry set
D1.W=priority (0-65535)	D1.W=error code

F$SRqCMem - allocate block of specific memory type.

INPUT:	**OUTPUT:**
D0.L= # of bytes requested	D0.L= # of bytes given
D1.L=memory type	(A2)=pointer to memory block
ERROR OUTPUT:	
CC=carry bit	
D1.W=error code	

F$SrqMem - allocate a block of memory from top of available system memory.

INPUT:	**OUTPUT:**
D0.L= # of bytes requested	D0.L= # of bytes given
ERROR OUTPUT:	(A2)=pointer to memory block
CC=carry bit	
D1.W=error code	

F$SRtMem - returns a block of memory to the system.

INPUT:	**ERROR OUTPUT:**
D0.L= # of bytes being returned	CC=carry bit
(A2)=address of returned block	D1.W=error code

F$SSpd - suspend a process.

INPUT:	**ERROR OUTPUT:**
D0.W=process ID	CC=carry bit
	D1.W=error code

F$STime - set system date and time and start real-time clock.

INPUT:	**ERROR OUTPUT:**
D0.L=time (00hhmmss)	CC=carry bit
D1.L=date (yyyymmdd)	D1.W=error code
OUTPUT:	
clock is set	

F$STrap - set process error trap routine.

INPUT:	**ERROR OUTPUT:**
(A0)=exception stack to use	CC=carry bit
(A1)=pointer to service request initialization table	D1.W=error code

F$SUser - set group or user ID number.

INPUT:	**ERROR OUTPUT:**
D1.L=group/user ID #	CC=carry bit
	D1.W=error code

F$SysDbg - starts system level debugger.

ERROR OUTPUT:
CC=carry bit
D1.W=error code

F$Time - get system time and date.

INPUT:	**OUTPUT:**
D0.W=format	D0.L=time
0=Gregorian	D1.L=date
1=Julian	D2.W=day of week (0=Sun, 6=Sat)
2=Gregorian w/ticks	D3.L=tick rate/current tick
3=Julian w/ticks	
ERROR OUTPUT:	
CC=carry bit	
D1.W=error code	

F$TLink - link or load named user trap handler module

INPUT:	**OUTPUT:**
D0.W=trap #	(A0)=updated past module name
D1.L=memory override	(A1)=trap library entry point
(A0)= module name pointer	(A2)=trap module pointer
(0 to unlink)	
ERROR OUTPUT:	
CC=carry bit	
D1.W=error code	

F$Trans - translate a memory block address to/from external bus address.

INPUT:	**OUTPUT:**
D0.L=size of block	D0.L=size of translated block
D1.L=mode:0 - local to external	(A0)=translated block address
1 - external to local	**ERROR OUTPUT:**
(A0)=block address	CC=carry bit
	D1.W=error code

F$UAcct - user accounting. Helps keep track of system/user activity.

INPUT:	**ERROR OUTPUT:**
D0.W=function code	CC=carry set
(A0)=process descriptor pointer	D1.W=error code

F$UnLink - decrements module link count (by header address), removes if result is 0.

INPUT:	**ERROR OUTPUT:**
(A2)=address of module header	CC=carry set
	D1.W=error code

F$UnLoad - decrements module link count (by name), removes if count=0.

INPUT:	**ERROR OUTPUT:**
D0.W=type/language	CC=carry set
(A0)=name pointer	D1.W=error code
OUTPUT:	
(A0)=updated past module name	

F$Wait - temporarily turn off calling process until child terminates.

OUTPUT:	**ERROR OUTPUT:**
D0.W=child process ID	CC=carry set
D1.W=child exit status code	D1.W=error code

I/O System Calls

I$Attach - attach or verify a device to system.

INPUT:
D0.B=access mode
(A0)=device name pointer

OUTPUT:
(A2)=device table entry address

ERROR OUTPUT:
CC=carry bit
D1.W=error code

I$ChgDir - change working directory.

INPUT:
D0.B=access mode
(A0)=path address

ERROR OUTPUT:
CC=carry set
B=error code

OUTPUT:
(A0)=updated path

ACCESS MODE PARAMETERS:
1=read only
2=write only
3=update
4=execute

I$Close - terminate I/O path.

INPUT:
D0.W=path #

ERROR OUTPUT:
CC=carry set
D1.W=error code

I$Create - create and open a file.

INPUT:
D0.B=access mode (S,I,E,W,R)
D1.W=attributes
D2.L=allocation size
(A0)=path pointer

ERROR OUTPUT:
D1.W=error code
CC=carry set

OUTPUT:
D0.W=path #
(A0)=update past pathlist

ATTRIBUTE BITS:

0=read	4=public write
1=write	5=public exec.
2=execute	6=shareable
3=public read	

I$Delete - delete a file.

INPUT:
D0.B=access mode
(A0)=pathname pointer

OUTPUT:
(A0)=updated past pathlist

ERROR OUTPUT:
CC=carry set
B=error code

I$Detach - remove device from system.

INPUT:
(A2)=device table entry address

ERROR OUTPUT:
CC=carry set
D1.W=error code

I$Dup - second path no. for same (duplicate) path (used to redirect I/O).

INPUT:
D0.W= # of path to copy

ERROR OUTPUT:
CC=carry set
D1.W=error code

OUTPUT:
D0.W=new path #

I$GetStt - get status of file or device.

INPUT:	**ERROR OUTPUT:**
D0.W=path	CC=carry set
D1.W=function code	D1.W=error code

Function Codes:

SS_DevNm (return device name)

INPUT:	*OUPUT:*
D0.W=path #	device name in storage area
D1.W=#SS_DevNm function code	
(A0)=address of storage area	

SS_EOF (test for end of file)

INPUT:	*ERROR OUTPUT:*
D0.W=path #	CC=carry set
D1.W=#SS_EOF function code	D1.W=error code
OUTPUT:	
D1.L=0 if not EOF	

SS_CDFD (return file descriptor)

INPUT:	*ERROR OUTPUT:*
D0.W=path #	CC=carry set
D1.W=#SS_CDFD function code	D1.W=error code
D2.W= # of bytes to copy	
(A0)=pointer to descriptor buffer area	

SS_FD (read file descriptor sector)

INPUT:	*OUTPUT:*
D0.W=path #	descriptor copied to buffer
D1.W=#SS_FD function code	
D2.W= # of bytes to copy	
(A0)=address of buffer area	

SS_DFInf (get specific file descriptor sector)

INPUT:	*OUTPUT:*
D0.W=path #	descriptor copied to buffer
D1.W=#SS_FDInf function code	
D2.W= # of bytes to copy	
D3.L=FD sector address	
(A0)=address of buffer area	

SS_Free (return amount of free space on device)

INPUT:	*OUTPUT:*
D0.L=path #	D0.L=size of free space in bytes
D1.W=#SS_Free function code	

(continued on next page)

I$GetStt Function Codes: *(continued from page 47)*

SS_Opt (read path descriptor option section)

INPUT:	*ERROR OUTPUT:*
D0.W=path #	CC=carry set
D1.W=#SS_Opt function code	D1.W=error code
(A0)=128 byte status area	*OUTPUT:*
	Status packet copied to status area

SS_Pos (get current file position)

INPUT:	*ERROR OUTPUT:*
D0.W=path #	CC=carry set
D1.W=#SS_Pos function code	D1.W=error code
OUTPUT:	
D2.L=current file position	

SS_Ready (check for data ready)

INPUT:	*ERROR OUTPUT:*
D0.W=path #	CC=carry set
D1.W=#SS_Ready function code	D1.W=error code
OUTPUT:	
D1.L= # of input characters available	

SS_Size (return current file size)

INPUT:	*ERROR OUTPUT:*
D0.W=path #	CC=carry set
D1.W=#SS_Size function code	D1.W=error code
OUTPUT:	
D2.L=file size	

(end of I$GetStt *function codes)*

I$MakDir - create and initialize directory.

INPUT:	**OUTPUT:**	
D0.B=mode	(A0)=updated past pathname	
D1.W=attributes	**ATTRIBUTE BITS:**	
D2.L=initial allocation size	0=read	4=public write
(A0)=path pointer	1=write	5=public exec.
ERROR OUTPUT:	2=execute	6=single user
CC=carry set	3=public read	7=any user/type
D1.W=error code		

MODE BITS:

0=read	2=execute	7=directory
1=write	5=beginning directory size	

I$Open - open path to existing file or device.

INPUT:	**OUTPUT:**
D0.B=access mode (D,S,E,W,R)	D0.W=path
(A0)=pathname pointer	(A0)=updated past pathname
ERROR OUTPUT:	**ACCESS MODE BITS:**
CC=carry set	0=read 1=write 2=execute
D1.W=error code	6=open non-shareable file 7=open dir file

I$Read - read number of bytes from path.

INPUT:	**ERROR OUTPUT:**
D0.W=path	CC=carry set
D1.L=maximum # of bytes to read	D1.W=error code
(A0)=storage address	
OUTPUT:	
D1.L= # of bytes read	

I$ReadLn - read line of text and activate line editing.

INPUT:	**ERROR OUTPUT:**
D0.W=path	CC=carry set
D1.L=maximum # of bytes to read	D1.W=error code
(A0)=input buffer address	**OUTPUT:**
	D1.L= # of bytes read

I$Seek - reposition file pointer.

INPUT:	**ERROR OUTPUT:**
D0.W=path	CC=carry set
D1.L=new position	D1.W=error code

I$SetStt - set status of file or device.

INPUT:	**ERROR OUTPUT:**
D0.W=path	CC=carry set
D1.W=function code	D1.W=error code

Function Codes:

SS_Attr (set file attributes)

INPUT:

D0.W=path	D2.W=new attributes
D1.W=#SS_Attr function code	

SS_Close (let driver know path is closed)

INPUT:

D0.W=path	D1.W=SS_Close function code

SS_DCOff (send signal when Data Carrier Detect=false)

INPUT:

D0.W=path	D2.W=signal code to be sent
D1.W=SS_DCOff function code	

SS_DCOn (send signal when Data Carrier Detect=true)

INPUT:

D0.W=path	D2.W=signal code to be sent
D1.W=SS_DCOn function code	

SS_DsRTS (disable Ready to Transmit)

INPUT:

D0.W=path	D1.W=SS_DsRTS function code

(continued on next page)

I$SetStt Function Codes: *(continued from previous page)*

SS_EnRTS (enable Ready to Transmit)
INPUT:
D0.W=path
D1.W=SS_EnRTS function code

SS_Feed (erase tape)
INPUT:
D0.W=path
D2.L= # of tape blocks to erase
D1.W=SS_Feed function code

SS_FD (write floppy disk sector)
INPUT:
D0.W=path
(A0)=floppy disk sector image address
D1.W=#SS_FD function code

SS_Lock (lock out a record)
INPUT:
D0.W=path
D2.L= # of bytes to lock out
D1.W=#SS_Lock function code

SS_Open (let driver know a path is open)
INPUT:
D0.W=path
D1.W=SS_Open function code

SS_Opt (write option section of path descriptor)
INPUT:
D0.W=path
(A0)=status packet address
D1.W=#SS_Opt function code

SS_Relea (release device from a request)
INPUT:
D0.W=path
D1.W=SS_Relea function code

SS_Reset (restore disk drive head to track 0 or rewind tape)
INPUT:
D0.W=path
D1.W=#SS_Reset function code

SS_RFM (skip tape marks)
INPUT:
D0.W=path
D2.L= # of marks to skip
D1.W=SS_ function code

SS_Size (set file size)
INPUT:
D0.W=path
D2.L=file size in bytes
D1.W=#SS_Size function code

(continued on next page)

I$SetStt Function Codes: *(continued from previous page)*

SS_Skip (skip tape blocks)
INPUT:

D0.W=path	D2.L= # of blocks to skip
D1.W=SS_Skip function code	

SS_SSig (send signal when device has data ready)
INPUT:

D0.W=path	D2.W=signal code
D1.W=SS_SSig function code	

SS_Ticks (wait # of ticks for record release)
INPUT:

D0.W=path	D2.L= # of ticks to wait
D1.W=#SS_Ticks function code	

SS_WFM (write tape marks)
INPUT:

D0.W=path	D2.L= # of tape marks to write
D1.W=SS_WFM function code	

SS_WTrk (format disk track)
INPUT:

D0.W=path	D3.W= Bit0=side (0 or 1)
D1.W=SS_WTrk function code	Bit1=density (0=sgl, 1=dbl)
(A0)=track buffer address	Bit2=track density (0=sgl, 1=dbl)
(A1)=interleave table address	D4=interleave value
D2=track # to format	

(end of I$SetStt *function code)*

I$Write - write to file or device.

INPUT:	**OUTPUT:**
D0.W=path	D1.L= # of bytes written
D1.L= # of bytes to write	**ERROR OUTPUT:**
(A0)=buffer address	CC=carry set
	D1.W=error code

I$WritLn - write to file or device until carriage return.

INPUT:	**OUTPUT:**
D0.W=path	D1.L= # of bytes written
D1.L=maximum # of bytes to write	**ERROR OUTPUT:**
(A0)=buffer address	CC=carry set
	D1.W=error code

System-State System Calls

F$Alarm - set alarm.

INPUT:
D0.L=alarm ID
D1.W=function code
D2.L=reserved, must be 0
D3.L=time or interval
D4.L=date (if absolute time)
(A0)=register image

OUTPUT:
D0.L=alarm ID

ERROR OUTPUT:
CC=carry set
D1.W=error code

Function Codes:
A$Delete (delete pending alarm)
A$Set (execute system-state subroutine after set time interval)
A$Cycle (execute system-state subroutine every interval)
A$AtDate (execute system-state subroutine on Gregorian date/time)
A$AtJul (execute system-state subroutine on Julian date/time)
See manual for info on subroutines.

F$AllPD - allocate process/path descriptor storage area.

INPUT:
(A0)=process/path table pointer

ERROR OUTPUT:
CC=carry set
D1.W=error code

OUTPUT:
D0.W=process/path #
(A1)=process/path descriptor pointer

F$AllPrc - allocate and initialize process descriptor.

OUTPUT:
(A2)=descriptor pointer

ERROR OUTPUT:
CC=carry set
D1.W=error code

F$AProc - insert a process into the active process queue for execution.

INPUT:
(A0)=process desciptor address

ERROR OUTPUT:
CC=carry set
D1.W=error code

F$DelPrc - deallocate process descriptor storage are (caller must return resources to system).

INPUT:
D0.W=process ID

ERROR OUTPUT:
CC=C bit set
D1.W=error code

F$FindPD - find address of process or path descriptor.

INPUT:
D0.W=process/path #
(A0)=process/path descriptor ptr

OUTPUT:
(A1)=process/path descriptor pointer

ERROR OUTPUT:
CC=carry set
D1.W=error code

F$IRQ - add or remove device from IRQ (system) polling table.

INPUT:	**ERROR OUTPUT:**
D0.B=vector #	CC=carry set
D1.B=priority	D1.W=error code
(A0)=IRQ entry point (0=delete)	
(A2)=device static storage	
(A3)=port address	

F$IRQ service routine register:

INPUT:	**ERROR OUTPUT:**
(A2)=global static pointer	Carry set if device din't cause interrupt
(A3)=port address	
(A6)=system global data pointer	
(A7)=system stack	

F$Move - block-move data from one address to another.

INPUT:	**ERROR OUTPUT:**
D2.L= # of bytes to copy	CC=carry set
(A0)=source pointer	D1.W=error code
(A2)=destination pointer	

F$NProc - execute next process in active queue.

OUTPUT:	**ERROR OUTPUT:**
control not returned to caller	CC=carry set
	D1.W=error code

F$Panic - kill system when a catastrphic occurrence is detected.

INPUT:	**OUTPUT:**
D0.L=panic code	does not usually return
ERROR OUTPUT:	**Defined Panic Codes:**
CC=carry set	K$Idle=no processes to execute
D1.W=error code	K$PFail=power failure detected

F$RetPD - deallocate process or path descriptor.

INPUT:	**ERROR OUTPUT:**
D0.W=process/path #	CC=carry set
(A0)=process/path table pointer	D1.W=error code

F$SSvc - add or replace a request in user & priveleged system service request table.

User-State System Service Requests:

INPUT:	**ERROR OUTPUT:**
(A1)=service request init. table ptr	CC=carry set
(A3)=user defined (usually global static storage)	D1.W=error code

System-State System Service Requests:

INPUT:	**ERROR OUPUT:**
D0-D4=user's values	CC=carry set
(A0)-(A2)=user's values	D1.W=error code
(A4)=current process descriptor pointer	
(A5)=user register's image pointer	
(A6)=system global data pointer	

F$VModul - check header parity and CRC of a module.

INPUT:	**ERROR OUTPUT:**
D0.L=beginning of module group	CC=carry set
D1.L=module size	D1.W=error code
(A0)=module pointer	

(end of system calls)

6 - Standard Math Module Function Subroutines

OS-9 is supplied with math subroutines for systems without a math coprocessor. The software based modules can be easily replaced by coprocessor modules with no application software changes. Calls are made in the format: TCALL T$Math,(function). Functions are listed.

T$Acs - returns arc cosine (x) in radians.

INPUT:	**CONDITION CODE:**
D0:D1 = x	C=set on error
D2:D3 = precision	
OUPUT:	**POSSIBLE ERROR:**
D0:D1 =ArcCos(x)	Illegal Argument

T$Asn - returns arc sine (x) in radians.

INPUT:	**CONDITION CODE:**
D0:D1 = x	C=set on error
D2:D3 = precision	
OUPUT:	**POSSIBLE ERROR:**
D0:D1 =ArcSin(x)	Illegal Argument

T$Atn - returns arc tangent (x) in radians.

INPUT:	**CONDITION CODE:**
D0:D1 = x	C=set on error
D2:D3 = precision	
OUPUT:	**POSSIBLE ERROR:**
D0:D1 =ArcTan(x)	Illegal Argument

T$AtoD - converts an ASCII string to a double-precision floating point number.

INPUT:	**CONDITION CODE:**
(A0) = pointer to ASCII string	N or Z = undefined
(sign)(digits).(digits) E (sign)(digits)	V = set on under/over flow
	C = set on error
OUPUT:	**POSSIBLE ERROR:**
(A0) = updated pointer	Not Number
D0:D1 = double-precision FP#	Format Error

T$AtoF - converts an ASCII string to a single precision floating point number.

INPUT:	**CONDITION CODE:**
(A0) = pointer to ASCII string	N or Z = undefined
(sign)(digits).(digits) E (sign)(digits)	V = set on under/over flow
	C = set on error
OUPUT:	**POSSIBLE ERROR:**
(A0) = updated pointer	Not Number
D0:D1 = single-precision FP#	Format Error

T$AtoL - converts and ASCII string to a signed long integer.

INPUT:	**CONDITION CODE:**
(A0) = pointer to ASCII string	N or Z = undefined
(sign)(digits)	V = set on under/over flow
OUPUT:	C = set on error
(A0) = updated pointer	**POSSIBLE ERROR:**
D0.L = signed long integer	Not Number

T$AtoN - returned results depend on condition codes.

INPUT:	**CONDITION CODE:**
(A0) = pointer to ASCII string	V=0 & N=1 = signed integer
OUPUT:	V=0 & N=0 = unsigned integer
(A0) = updated pointer	V=1 = double-precision FP number
D0 = # (if long signed/unsigned integer)	**POSSIBLE ERROR:**
D0:D1 = # (if floating point)	TrapV

T$AtoU - converts an ASCII string to an unsigned long integer.

INPUT:	**CONDITION CODE:**
(A0) = pointer to ASCII string	N or Z = undefined
(digits)	V = set on under/over flow
OUPUT:	C = set on error
(A0) = updated pointer	**POSSIBLE ERROR:**
D0.L = unsigned long integer	Not Number

T$Cos - returns cosine (x) of an angle in radians.

INPUT:	**CONDITION CODE:**
D0:D1 = x	C = always clear
D2:D3 = precision	**OUTPUT:**
	D0:D1 = Cos(x)

T$DAdd - add two double-precision floating point numbers.

INPUT:	**CONDITION CODE:**
D0:D1 = addend	N = set if result negative
D2:D3 = augend	Z = set if result zero
OUTPUT:	V = set on under/over flow
D0:D1 = result	**POSSIBLE ERROR:**
C = always clear	TrapV

T$DCmp - compare two double precision floating point numbers.

INPUT:	**CONDITION CODE:**
D0:D1 = first operand	N = set if second larger than first
D2:D3 = second operand	Z = set if equal
OUTPUT:	V = always clear
D0.L-D3.L = unchanged	C = always clear

T$DDec - subtract 1.0 from a double precision floating point operand.

INPUT:	**CONDITION CODE:**
D0:D1 = operand	N = set if result negative
OUTPUT:	Z = set if result zero
D0:D1 = result	V = set on underflow
POSSIBLE ERROR:	C = always clear
TrapV	

T$DDiv - divide two sdouble precision floating point numbers.

INPUT:	**CONDITION CODE:**
D0:D1 = dividend	N = set if result is negative
D2:D3 = divisor	Z = set if result is 0
OUTPUT:	V = set on under/over flow, divide by 0
D0:D1 = result	C = set on divide by 0
POSSIBLE ERROR:	
TrapV	

T$DInc - add 1.0 to a double precision floating point operand.

INPUT:	**CONDITION CODE:**
D0:D1 = operand	N = set if result negative
OUTPUT:	Z = set if result zero
D0:D1 = result	V = set on]overflow
POSSIBLE ERROR:	C = always clear
TrapV	

T$DInt - round floating point number to nearest integer.

INPUT:	**OUTPUT:**
D0:D1 = number	D0:D1 = rounded integer

T$DMul - multiply two double precision floating point numbers.

INPUT:	**CONDITION CODE:**
D0:D1 = multiplicand	N = set if result negative
D2:D3 = multiplier	Z = set if result 0
OUPUT:	V = set on under/over flow
D0:D1 = result	C = always clear
POSSIBLE ERROR:	
TrapV	

T$DNeg - negate a double precision floating point number.

INPUT:	**CONDITION CODE:**
D0:D1 = operand	N = set if result negative
OUTPUT:	Z = set if result 0
D0:D1 = result	V & C = always clear

T$DNrm - convert 64 bit binary number to double precision format.

INPUT:
D0:D1 = 64 bit number
D2.L = exponent
OUTPUT:
D0:D1 = double precision #

CONDITION CODE:
N & Z = undefined
V & C = always clear

T$DSub - subtract two double precision floating point numbers.

INPUT:
D0:D1 = minuend
D2:D3 = subtrahend
OUPUT:
D0:D1 = result
POSSIBLE ERROR:
TrapV

CONDITION CODE:
N = set if result negative
Z = set if result 0
V = set on under/over flow
C = always clear

T$DtoA - convert double precision floating point number to an ASCII string.

INPUT:
D0:D1 = double precision #
D2.L - low word = digits desired in result
high word = digits desired after decimal
(A0) = pointer to buffer

CONDITION CODE:
N = set if negative number
Z, V, C = undefined
OUTPUT:
(A0) = ASCII string
D0.L = 2's comp. exponent

T$DtoF - convert double precision floating point number to single precision floating point number.

INPUT:
D0:D1 = double precision #
OUTPUT:
D0.L = single precision #

CONDITION CODE:
N, Z, C = undefined
V = set on under/over flow
POSSIBLE ERROR:
TrapV

T$DtoL - convert integer portion of a double precision floating point number to a signed long integer (truncates fraction).

INPUT:
D0:D1 = double precision #
OUTPUT:
D0.L = signed long integer
POSSIBLE ERROR:
TrapV

CONDITION CODE:
N = undefined
Z = undefined
V = set on under/over flow
C = undefined

T$DtoU - convert integer portion of a double precision floating point number to an unsigned long integer (truncates fraction).

INPUT:
D0:D1 = double precision #
OUTPUT:
D0.L = unsigned long integer
POSSIBLE ERROR:
TrapV

CONDITION CODE:
N = undefined
Z = undefined
V = set on under/over flow
C = undefined

T$DTrn - separate double precision floating point integer and fraction.

INPUT:	**OUTPUT:**
D0:D1 = double precision #	D0:D1 = integer
CONDITION CODE:	D2:D3 = fraction
All = undefined	

T$Exp - exponential function. Raises e (2.718282) to the x power.

INPUT:	**OUTPUT:**
D0:D1 = x	D0:D1 = exp(x)
D2:D3 = precision	**CONDITION CODE:**
	C = always clear

T$FAdd - add two single precision floating point numbers.

INPUT:	**CONDITION CODE:**
D0.L = addend	N = set if result negative
D1.L = augend	Z = set if result 0
OUPUT:	V = set on under/over flow
D0.L = result	C = always clear
POSSIBLE ERROR: TrapV	

T$FCmp - compare two single precision floating point numbers.

INPUT:	**CONDITION CODE:**
D0.L = first operand	N = set if second larger than first
D1.L = second operand	Z = set if equal
OUTPUT:	V = always clear
D0.L-D1.L = unchanged	C = always clear

T$FDec - subtract 1.0 from a single precision floating point operand.

INPUT:	**CONDITION CODE:**
D0.L = operand	N = set if result negative
OUTPUT:	Z = set if result zero
D0.L = result	V = set on underflow
POSSIBLE ERROR:	C = always clear
TrapV	

T$FDiv - divide two single precision floating point numbers.

INPUT:	**CONDITION CODE:**
D0.L = dividend	N = set if result is negative
D1.L = divisor	Z = set if result is 0
OUTPUT:	V = set on under/over flow, divide by 0
D0.L = result	C = set on divide by 0
POSSIBLE ERROR: TrapV	

T$FInc - add 1.0 to single precision floating point operand.

INPUT:	**CONDITION CODE:**
D0.L = operand	N = set if result negative
OUTPUT:	Z = set if result zero
D0.L = result	V = set on]overflow
POSSIBLE ERROR:	C = always clear
TrapV	

T$FInt - round floating point number to nearest integer.

INPUT:	**OUTPUT:**
D0.L = number	D0.L = rounded integer

T$FMul - multiply two single precision floating point numbers.

INPUT:	**CONDITION CODE:**
D0.L = multiplicand	N = set if result negative
D1.L = multiplier	Z = set if result 0
OUPUT:	V = set on under/over flow
D0.L = result	C = always clear
POSSIBLE ERROR:	
TrapV	

T$FNeg - negate a single precision floating point number.

INPUT:	**CONDITION CODE:**
D0.L = operand	N = set if result negative
OUTPUT:	Z = set if result 0
D0.L = result	V & C = always clear

T$FSub - subtract two single precision floating point numbers.

INPUT:	**CONDITION CODE:**
D0.L = minuend	N = set if result negative
D1.L = subtrahend	Z = set if result 0
OUPUT:	V = set on under/over flow
D0:D1 = result	C = always clear
POSSIBLE ERROR:	
TrapV	

T$FtoA - convert single precision floating point number to an ASCII string.

INPUT:	**CONDITION CODE:**
D0.L = single precision #	N = set if negative number
D2.L - low word = digits desired in result	Z, V, C = undefined
high word = digits desired after decimal	**OUTPUT:**
(A0) = pointer to buffer	(A0) = ASCII string
	D0.L = 2's comp. exponent

T$FtoD - convert single precision floating point number to double precision floating point number.

INPUT:	**CONDITION CODE:**
D0.L = single precision #	N, Z, C = undefined
OUTPUT:	V = set on under/over flow
D0.L = single precision #	**POSSIBLE ERROR:**
	TrapV

T$FtoL - convert integer portion of a single precision floating point number to a signed long integer (truncates fraction).

INPUT:	**CONDITION CODE:**
D0.L = single precision #	N, Z, C = undefined
OUTPUT:	V = set on under/over flow
D0.L = signed long integer	**POSSIBLE ERROR:** TrapV

T$FtoU - convert integer portion of a single precision floating point number to an unsigned long integer (truncates fraction).

INPUT:
D0.L = single precision #
OUTPUT:
D0.L = unsigned long integer

CONDITION CODE:
N, Z, C = undefined
V = set on under/over flow
POSSIBLE ERROR:
TrapV

T$FTrn - separate single precision floating point integer and fraction.

INPUT:
D0.L = single precision FP #
CONDITION CODE:
All = undefined

OUTPUT:
D0.L = integer

T$LDiv - divide two long signed 32 bit integers.

INPUT:
D0.L = dividend
D1.L = divisor
OUTPUT:
D0.L = result

CONDITION CODE:
N = set if result negative
Z = set if result 0
V = set on divide by 0
C = always clear

T$LMod - divide two long signed 32 bit integers, return remainder.

INPUT:
D0.L = dividend
D1.L = divisor
OUTPUT:
D0.L = result (remainder)

CONDITION CODE:
N = set if result negative
Z = set if result 0
V = set on divide by 0
C = always clear

T$LMul - multiply two long signed 32 bit integers.

INPUT:
D0.L = multiplicand
D1.L = multiplier
OUTPUT:
D0.L = result

CONDITION CODE:
N = set if result negative
Z = set if result 0
V = set on divide by 0
C = always clear

T$Log - natural logarithm of x.

INPUT:
D0:D1 = x
D2:D3 = precision
POSSIBLE ERROR:
Illegal Argument

OUTPUT:
D0:D1 = log(x)
CONDITION CODE:
C = set on error

T$Log10 - common logarithm of x.

INPUT:
D0:D1 = x
D2:D3 = precision
POSSIBLE ERROR:
Illegal Argument

OUTPUT:
D0:D1 = log10(x)
CONDITION CODE:
C = set on error

T$LtoA - convert signed long integer to ASCII string (10 digits, leading zeroes used if less than 10 digits).

INPUT:
D0.L = signed long integer
(A0) = pointer to buffer
OUTPUT:
(A0) = ASCII string

CONDITION CODE:
N = set if negative
Z, V , C = undefined

T$LtoD - convert a signed long integer to a double precision floating point number.

INPUT:
D0.L = signed long integer
CONDITION CODE:
All undefined

OUTPUT:
D0:D1 = double precision FP #

T$LtoF - convert a signed long integer to a single precision floating point number.

INPUT:
D0.L = signed long integer
CONDITION CODE:
All undefined

OUTPUT:
D0.L = single precision FP #

T$Power - raise x to the y power.

INPUT:
D0:D1 = x
D2:D3 = y
D4:D5 = precision
POSSIBLE ERROR:
Illegal Argument

OUTPUT:
D0:D1 = x raised to y power
CONDITION CODE:
C = set on error

T$Sin - sine of an angle specified in radians.

INPUT:
D0:D1 = angle (in radians)
D2:D3 = precision

OUTPUT:
D0:D1 = sine
CONDITION CODE:
C = always clear

T$Sqrt - square root of x.

INPUT:
D0:D1 = x
D2:D3 = precision
POSSIBLE ERROR:
Illegal Argument

OUTPUT:
D0:D1 = square root of x
CONDITION CODE:
C = set on error

T$Tan - tangent of an angle specified in radians

INPUT:
D0:D1 = angle (in radians)
D2:D3 = precision
CONDITION CODE:
C = always clear

OUTPUT:
D0:D1 = tangent

T$UDiv - divide two 32 bit unsigned integers.

INPUT:	**CONDITION CODE:**
D0.L = dividend	N = undefined
D1.L = divisor	Z = set if result is 0
OUTPUT:	V = set on divide by 0
D0.L = result	C = always clear

T$UMod - divide two 32 bit unsigned integers, return remainder.

INPUT:	**CONDITION CODE:**
D0.L = dividend	N = undefined
D1.L = divisor	Z = set if result is 0
OUTPUT:	V = set on divide by 0
D0.L = result	C = always clear

T$UMul - multiply two single precision floating point numbers.

INPUT:	**CONDITION CODE:**
D0.L = multiplicand	N = undefined
D1.L = multiplier	Z = set if result 0
OUPUT:	V = set on overflow
D0.L = result	C = always clear

T$UtoA - convert unsigned long integer to ASCII string (10 digits, leading zeroes used if less than 10 digits).

INPUT:	**OUTPUT:**
D0.L = unsigned long integer	(A0) = ASCII string
(A0) = pointer to buffer	

CONDITION CODE:
All undefined

T$UtoD - convert unsigned long integer to double precision floating point number.

INPUT:	**OUTPUT:**
D0.L = unsigned long integer	D0:D1 = double precision FP #

CONDITION CODE:
All undefined

T$UtoF - convert unsigned long integer to single precision floating point number.

INPUT:	**OUTPUT:**
D0.L = unsigned long integer	D0.L = single precision FP #

CONDITION CODE:
All undefined

7 - System/Basic Error Codes

Only built-in error codes are listed. Programs and programming languages may define their own codes, which will not appear in the following listing. Entries are decimal number (mnemonic, if available) followed by name. Numbers may have leading zeroes.

Signal Error Codes
02 KEYBOARD ABORT- CTRL E was pressed.
03 KEYBOARD INTERRUPT- CTRL C was pressed.

Basic Error Codes
10 UNRECOGNIZED SYMBOL
11 EXCESSIVE VERBIAGE
12 ILLEGAL STATEMENT CONSTRUCTION
13 I-CODE OVERFLOW- need more workspace memory
14 ILLEGAL CHANNEL REFERENCE- bad path number
15 ILLEGAL MODE- read,write, update; directory only
16 ILLEGAL NUMBER
17 ILLEGAL PREFIX
18 ILLEGAL OPERAND
19 ILLEGAL OPERATOR
20 ILLEGAL RECORD FIELD NAME
21 ILLEGAL DIMENSION
22 ILLEGAL DIMENSION
23 ILLEGAL RELATIONAL
24 ILLEGAL TYPE SUFFIX
25 TOO-LARGE DIMENSION
26 TOO-LARGE LINE NUMBER
27 MISSING ASSIGNMENT STATEMENT
28 MISSING PATH NUMBER
29 MISSING COMMA
30 MISSING DIMENSION
31 MISSING DO STATEMENT
32 MEMORY FULL- need more workspace memory
33 MISSING GOTO
34 MISSING LEFT PARENTHESIS
35 MISSING LINE REFERENCE
36 MISSING OPERAND
37 MISSING RIGHT PARENTHESIS
38 MISSING THEN STATEMENT
39 MISSING TO
40 MISSING VARIABLE REFERENCE
41 NO ENDING QUOTE
42 TOO MANY SUBSCRIPTS
43 UNKNOWN PROCEDURE
44 MULTIPLY-DEFINED PROCEDURE
45 DIVIDE BY ZERO
46 OPERAND TYPE MISMATCH
47 STRING STACK OVERFLOW

48 UNIMPLEMENTED ROUTINE
49 UNDEFINED VARIABLE
50 FLOATING OVERFLOW
51 LINE WITH COMPILER ERROR
52 VALUE OUT OF RANGE FOR DESTINATION
53 SUBROUTINE STACK OVERFLOW
54 SUBROUTINE STACK UNDERFLOW
55 SUBSCRIPT OUT OF RANGE
56 PARAMETER ERROR
57 SYSTEM STACK OVERFLOW
58 I/O TYPE MISMATCH
59 I/O NUMERIC INPUT FORMAT BAD
60 I/O CONVERSION number out of range
61 ILLEGAL INPUT FORMAT
62 I/O FORMAT REPEAT ERROR
63 I/O FORMAT SYNTAX ERROR
64 ILLEGAL PATH NUMBER
65 WRONG NUMBER OF SUBSCRIPTS
66 NON-RECORD-TYPE OPERAND
67 ILLEGAL ARGUMANT
68 ILLEGAL CONTROL STRUCTURE
69 UNMATCHED CONTROL STRUCTURE
70 ILLEGAL FOR VARIABLE
71 ILLEGAL EXPRESSION TYPE
72 ILLEGAL DECLARATIVE STATEMENT
73 ARRAY SIZE OVERFLOW
74 UNDEFINED LINE NUMBER
75 MULTIPLY-DEFINED LINE NUMBER
76 MULTIPLY-DEFINED VARIABLE
77 ILLEGAL INPUT VARIABLE
78 SEEK OUT OF RANGE
79 MISSING DATA STATEMENT
80 PRINT BUFFER OVERFLOW
(81-101 undefined for Basic or System)

Math Trap Handler Error Codes (64-67 also defined by Basic)
64 (E$IllFnc) ILLEGAL FUNCTION CODE
65 (E$FmtErr) FORMAT ERROR
66 (E$NotNum) NUMBER NOT FOUND
67 (E$IllArg) ILLEGAL ARGUMENT

Processor Exception Error Codes (100-155)
102 (E$BusErr) BUS ERROR- exception occured.
103 (E$AdrErr) ADDRESS ERROR- exception occured.
104 (E$IllIns) ILLEGAL INSTRUCTION- exception occured.
105 (E$ZerDiv) ZERO DIVIDE- can't divide by zero.
106 (E$Chk) CHECK- CHK instruction exception occured.
107 (E$TrapV) TRAPV- TrapV instruction exception occured.
108 (E$Violat) PRIVILEGE VIOLATION- exception occured.
109 (E$Trace) UNINITIALIZED TRACE EXCEPTION- exception occured.
110 (E$1010) 1010 TRAP- A line emulator exception.
111 (E$1111) 1111 TRAP- F line emulator exception.

113 COPROCESSOR PROTOCOL VIOLATION
114 FORMAT ERROR
115 UNINITIALIZED INTERRUPT OCCURRED
(116-123 undefined)
124 SPURIOUS INTERRUPT OCCURRED
(125-132 undefined)
133-147 (E$Trap) TRAP* uninitialized user TRAP* (*=1-15) executed

Floating Point Coprocessor (FPCP) Errors (148-155)
148 (E$FPUnordC) FPCP ERROR- branch or set on unordered condition
149 (E$FPInxact) FPCP ERROR- Inexact results
150 (E$FPDivZer) FRCP ERROR-divide by zero
151 (E$FPUndrFl) FPCP ERROR- underflow error
152 (E$FPOprErr) FPCP ERROR- operand error
153 (E$FPOverFl) FPCP ERROR- overflow error
154 (E$FPNotNum) FPCP ERROR- not a number (NAN) signaled

Processor Memory Management Unit (PMMU) Errors (156-163)
156 CONFIGURATION ERROR
157 ILLEGAL OPERATION
158 ACCESS LEVEL VIOLATION

Miscellaneous Error Codes (164-199)
164 (E$Permit) NO PERMISSION- user doesn't have permission to perform function.
165 (E$Differ) DIFFERENT ARGUMENTS- F$ChkNam arguments don't match.
166 (E$StkOvf) STACK OVERFLOW- pattern string to complex.
167 (E$EvntID) ILLEGAL EVENT ID- illegal ID number.
168 (E$EvNF) EVENT NAME NOT FOUND- name not in event table.
169 (E$EvBusy) EVENT BUSY- link count not 0.
170 (E$EvParm) IMPOSSIBLE EVENT PARAMETER- bad parameters passed to F$Event.
171 (E$Damage) SYSTEM DAMAGE- data structure corrupted.
172 (E$BadRev) INCOMPATIBLE REVISION- software incompatible with current OS revision.
173 (E$PthLost) PATH LOST- path no longer available.
174 (E$Bad Part) BAD PARTITION- partition data bad or not active.

General System Error Codes (200-239)
200 (E$PthFul) PATH TABLE FULL- can't track any more files.
201 (E$BPNum) ILLEGAL PATH NUMBER- number to large or doesn't exist.
202 (E$Poll) INTERRUPT POLLING TABLE FULL- no room for more entries.
203 (E$BMode) ILLEGAL MODE- device can't perform function.
204 (E$DevOvf) DEVICE TABLE FULL- no more devices can be added.
205 (E$BMID) ILLEGAL MODULE HEADER- bad sync code, header parity, or CRC.
206 (E$DirFul) MODULE DIRECTORY FULL- modules can't be entered.
207 (E$MemFul) MEMORY FULL- no more available memory.
208 (E$UnkSvc) ILLEGAL SERVICE REQUEST- issued system call has illegal code.
209 (E$ModBsy) MODULE BUSY- non-shareable module in use.
210 (E$BPAddr) BOUNDARY ERROR- memory allocation/deallocation not on page boundary.
211 (E$EOF) END OF FILE- read terminated.
212 (E$VctBsy) VECTOR BUSY- IRQ vector currently in use.
213 (E$NES) NON-EXISTING SEGMENT- file structure of device bad.

214 (E$FNA) FILE NOT ACCESSIBLE- user doesn't have access to perform specified operation.
215 (E$BPNam) BAD PATHNAME- syntax error in path.
216 (E$PNNF) PATH NAME NOT FOUND- can't find path.
217 (E$SLF) SEGMENT LIST FULL- file to fragmented to be expanded.
218 (E$CEF) FILE ALREADY EXISTS- file exists in current directory
219 (E$IBA) ILLEGAL BLOCK ADDRESS- device file structure bad.
220 (E$HangUp) PHONE HANGUP - DATA CARRIER LOST- no carrier on RS-232 port.
221 (E$MNF) MODULE NOT FOUND- module not in directory.
222 (E$NoClk) NO CLOCK- system has no clock running.
223 (E$DelSP) SUICIDE ATTEMPT- attempt to return to stack.
224 (E$IPrcID) ILLEGAL PROCESS NUMBER- non-existant process.
225 (E$Param) BAD POLLING PARAMETER- impossible vector number passed to IRQ.
226 (E$NoChld) NO CHILDREN- wait service issued but no dependants.
227 (E$ITrap) ILLEGAL TRAP CODE- unavailable or invalid trap code.
228 (E$PrcAbt) PROCESS ABORTED- current process terminated.
229 (E$PrcFul) PROCESS TABLE FULL- no more processes can be run.
230 (E$IForkP) ILLEGAL PARAMETER AREA- fork passed bad boundaries.
231 (E$KwnMod) KNOWN MODULE- module already in memory.
232 (E$BMCRC) INCORRECT MODULE CRC- bad module CRC.
233 (E$USigP) UNPROCESSED SIGNAL PENDING- receiving process has signal pending.
234 (E$NEMod) NON-EXECUTABLE MODULE- module can't be executed.
235 (E$BNam) BAD NAME- illegal name used.
236 (E$BMHP) BAD PARITY- module parity header bad.
237 (E$NoRAM) RAM FULL- no system RAM available.
238 (E$DNE) DIRECTORY NOT EMPTY
239 (E$NoTask) NO TASK NUMBER AVAILABLE- all in use.

Device Driver Error Codes (240-255)

240 (E$Unit) ILLEGAL DRIVE NUMBER
241 (E$Sect) BAD ERROR- sector # out of range or bad.
242 (E$WP) WRITE PROTECT- device write protected.
243 (E$CRC) CRC ERROR- bad CRC on read/write verify.
244 (E$Read) READ ERROR- disk read data error or terminal input overrun.
245 (E$Write) WRITE ERROR- error during device write.
246 (E$Ready) NOT READY- device not ready.
247 (E$Seek) SEEK ERROR- seek attempted on non-existant sector.
248 (E$Full) MEDIA FULL- not enough free disk space.
249 (E$BTyp) WRONG TYPE- attempt to read incompatible disk.
250 (E$DevBsy) DEVICE BUSY- non-shareable device in use.
251 (E$DIDC) DISK ID CHANGE- disk changed with files still open.
252 (E$Lock) RECORD IS LOCKED OUT- record is being used.
253 (E$Share) NON-SHAREABLE FILE BUSY- file being used.
254 (E$DeadLk) I/O DEADLOCK- two processes attempting to use same disk area.
255 (E$Format) DEVICE IS FORMAT PROTECTED- cannot format disk (check descriptor).

Do you want to learn more about
OS-9/68000 (OSK) or OS-9000
and systems designed specifically for OSK?

If so, you need a subscription to:

the world of
68' micros
Supporting Motorola Processors

"the world of 68' micros" offers support for ALL Motorola processors. Current support is predominantly for the Tandy Color Computer (MC6809E), OS-9, and OS-9/68000 running on BlackHawk/IMS, Computer Design Services, Hazelwood, and Peripheral Technology 680x0 based systems (MM/1, Delmar Systems, and FHL). Other Motorola processors and microcontrollers also covered. Receive eight issues per year (approx. every six weeks) for only $23*, four issues over six months for $12*.

Send Check or Money order to:
FARNA Systems
Box 321
Warner Robins, GA 31099

* Canada/Mexico: $30/$16; Overseas , $40/$20 Air. All prices inU.S. funds. Canadians may send Canadian Postal Money Orders. Checks accepted if drawn on a U.S. bank with U.S. address (sorry, but Canadian U.S. dollar accounts cannot be processed by our bank). All others must pay by International Money Order or cash sent registered mail. Not responsible for cash payment sent via normal mail.

```
Professional OS-9 V2.3
Copyright 1994 by Microware Systems Corp.
All Rights Reserved

*     Welcome to OS-9     *
*         on the          *
*      Motorola 68030     *

$
```

FARNA Systems
Box 321
Warner Robins, Georgia
31099-0321

Support for OS-9, OS-9/68000, and the Tandy Color Computer