The diskette for **FILTER KIT #2** contains the following:

directory of . 15:50:56

Owner	Last modified	attributes	sector	bytecount	name
0	84/07/02 1542	d-ewrewr	A	180	CMDS
0	84/07/02 1541	d-ewrewr	13	A0	MACROS

directory of CMDS 15:51:06

Owner	Last modified	attributes	sector	bytecount	name
0	84/07/02 1542	--e-rewr	1C	11A	append
0	84/07/02 1542	--e-rewr	1F	FB	confirm
0	84/07/02 1542	--e-rewr	21	36	ff
0	84/07/02 1543	--e-rewr	23	6B	forcerror
0	84/07/02 1543	--e-rewr	25	5C4	macgen
0	84/07/02 1543	----rewr	2C	4C	nuldevice
0	84/07/02 1543	--e-rewr	2E	198	rep
0	84/07/02 1543	--e-rewr	31	2F3	size
0	84/07/02 1543	--e-rewr	35	1E9	touch
0	84/07/02 1543	--e-rewr	38	B0	unload

directory of MACROS 15:51:17

Owner	Last modified	attributes	sector	bytecount	name
0	84/06/26 1317	----r-wr	3A	45	cat
0	84/06/27 2154	----r-wr	3C	DA	ed
0	84/06/26 1317	----r-wr	3E	A8	namebak

The CMDS directory contains only executable programs, refer to the documention with this package for their use. You will probably want to copy these to your working CMDS directory on your system disk.

The MACROS directory contains macro text files for example macros given in the printed text for macgen.

If your diskette contains a DOC directory, this will contain text files with documentation updates that may not be in the printed documentation.

ABOUT YOUR ORDER

Before you open the diskette package, examine the documentation to confirm you have received what you ordered. You should also determine from the documentation if the software you ordered will meet your needs. If you have received the wrong item, return the unopened disk package with all documentation and a note stating the problem and we will send you the correct item. If after looking at the documentation you feel there was a misunderstanding as to the function of the software, and it won't meet your needs you may return the UNOPENED disk package and documentation for a refund. No refunds will be given after the diskette package is opened (except for media that is defective according to the terms on the disk package).

ABOUT DOCUMENTATION

No amount of documentation will do you any good if you don't read it. The documentation included with this software assumes you have a basic knowledge of using your system and does not explain in depth information that is covered elsewhere (in your system manuals). We have tried to use terminology consistent with that used in the OS-9 system documentation. If you have not at least read through your OS-9 documentation that was included with your system we strongly urge you to do so. If you do not understand something about our documentation, first see if there is some word you skipped over that you did not understand (like "pipes", "device descriptor", "I/O redirection", etc.) that is explained in the OS-9 COMMANDS or OS-9 TECHNICAL MANUAL, study that manual then reread our documentation, if it still does not make any sense then try giving us a call.

DISKETTES

If you have difficulty reading the diskette supplied try it in more than one drive if you have more than one. If that doesn't work use the test program supplied with the RS OS-9 BOOT diskette to check your drives rotational speed. If the diskette has been exposed to temperature and humidity extremes it may need to sit for a day in your environment after you receive it to achive dimensional stability. Another factor affecting diskette compatability is the track alignment on the disk drives (yours and ours), if they are off a disk becomes unreadable. If after these attempts you can still not read the disk return it for a replacement.

D. P. Johnson
7655 S.W. Cedarcrest St.
Portland, OR 97223
(503) 244-8152

FILTER KIT #2

Filter kit #2 provides an extension of the capabilities of Filter kit #1. Most of the programs in this package are not used as filters however. The two that are, SIZE, and TOUCH are used in the same manner as programs in Kit #1, i.e. usually with a list of names piped from LS.

One generally useful utility provided is the UNLOAD command which does the inverse of the LOAD function, in other words it removes modules from memory.

The NULDEVICE driver provided creates a "bit-bucket" device that can be useful when you want to discard the output of a program.

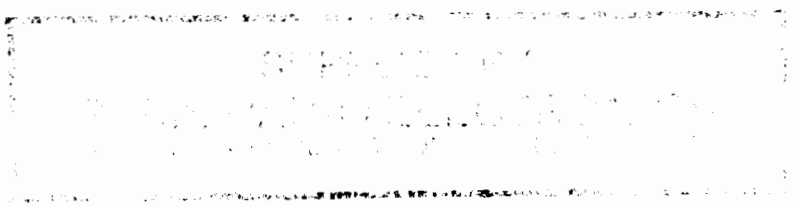
The REP command provides a means of using any OS9 command that was made to operate on a single file (the ident command for example) with a list of files the way the programs in Filter Kit #1 function.

The most complex and versatile command in this package is MACGEN. It provides the ability to create new executable commands by combining any existing OS9 programs (commands) that you have. Its function is similar to a procedure file with the addition of parameter substitution and conditional control. The "MACRO" you generate with macgen however is an executable module that may be loaded in memory.

Many of the examples in this documentation show use of the programs in conjunction with programs contained in FILTER KIT #1. If you do not have these then of course you can not execute the examples as shown. Just for your reference in this case the programs contained in Filter Kit #1 are: LS, INFO, CP, DL, MV, BUF, PAG, FLIST, REMOVE, SELL, SETAT, SORT.

DOCUMENTATION OF PROGRAMS IS ARRANGED IN ALPHABETICAL ORDER

USER INPUT IN EXAMPLES IS SHOWN UNDERLINED



APPEND

SYNTAX: APPEND [-opts] dest-path source-path [source-path..]

or

APPEND -i dest-path [source-path]

Append extends the destination file by copying the source file(s) to the end of the destination file. When the -i option is used the standard input is appended to the destination file after any optional source files listed.

Options have form -A or -An where n is a decimal number associated with that option. Options are single letters A through Z (upper or lower case). Multiple options can be written as -A -B-C or -ABC .

OPTIONS: -I The standard input is appended to the destination file after any source file(s) given on the command line.

EXAMPLES:

OS9: append textl text2 text3

The files text2 and text3 are appended to textl, i.e. file textl is expanded by seeking to the end of textl and writing the contents of text2 followed by the contents of text3.

OS9 date ! append -i textl

The output of the "date" command is appended to textl.

CONFIRM

SYNTAX: CONFIRM [-opts] [text message]

Confirm writes any text_message given to the standard output then waits for a key to be typed on the standard input. If the ´y´ or ´Y´ key is typed then confirm exits without error. If any other key is typed then confirm exits with an error status of 1 or as given by the -e options. Confirm is useful in a macro generated with the macgen command to provide conditional control.

Options have form -A or -An where n is a decimal number associated with that option. Options are single letters A through Z (upper or lower case). Multiple options can be written as -A -B-C or -ABC .

OPTIONS: -En The error number issued by confirm in response to negative confirmation (´Y´ not typed) is set to n.

-N The error issuing logic is inverted, i.e. an error is issued upon exit if ´y´ or ´Y´ WAS typed.

EXAMPLE:

Using confirm within a macro:

```
OS9: macgen formatit
MAC: .repeat
MAC:   format /d1 r "OS9 Data Disk"
MAC:   confirm Format another disk on /d1 ?
MAC: .until
MAC: .clear
MAC:
```

FF

Sends a form feed character to specified device.

syntax: FF devname

Example: FF /P1
Sends a single form feed (hex 0C) to printer P1 .

FORCERROR

SYNTAX: FORCERROR error_number

Causes the error error_number (decimal) to be returned to the shell that invoked the forcerror command. This may be useful in some circumstances inside a macro generated by macgen to control macro flow.

EXAMPLE:

```
OS9: forcerror 211
Error #211
```

SYNTAX: MACGEN [-opts] macro_name

MACGEN is a command macro generator that allows building new commands out of combinations of any old commands (executable program modules).

Options have form -A or -An where n is a decimal number associated with that option. Options are single letters A through Z (upper or lower case). Multiple options can be written as -A -B-C or -ABC .

OPTIONS:

-En The e option sets the edition number of the macro generated to n.

-L The l option causes the macro command to be loaded into memory after it is saved to the execution directory.

-M The m option causes the macro generated to be memory resident only. (The macro is first saved to the execution directory, then it is loaded into memory, then its file in the execution directory is deleted.

-Rn The r option sets the revision number of the macro command generated to n.

-T The T option will cause the macro generated by macgen to display each of the commands it is executing on the standard output in a similar manner as the "t" shell option displays lines being executed from a batch file.

USING MACGEN:

When macgen is invoked it will create a macro with the module name of "macro_name" supplied on the command line which will be saved in a file of the same name in the current execution directory. When macgen is invoked it will send the prompt "MAC:" to the standard output and wait for a macro text line to be input on the standard input after each prompt. Entering a blank line after the prompt (typing only the return key) will end the macro and cause it to be saved to an executable file. (This function is similar to the way the build command works.) Each macro line may contain any valid shell line with optional string parameter substitution symbols $0, $1, $2 ... $9. When the macro is later invoked as a command the actual command line text words will be substituted for the $ parameters in order that they appear on the line. If there are more $parameter numbers present in the macro than on the command line when it is invoked the unused numbers will have a null string value (no characters). A macro line may contain any of the special control words { .iferr .ifnul .else

.endif .for .next .repeat .until .clear }in place of shell commands to conditionally execute depending on the error status of the previous line. Control words must be on a line without any other text (any text after the control word is ignored to the end of line). Macro lines may also contain the parameter setting conditionals {.onnul, .onset} which will be explained in more detail later.

EXAMPLES:

OS9: macgen cat

MAC: LS $0 ! SORT ! PAG -C6 $1 $2 $3 $4
MAC:

The above example will create the executable file "CAT" in the current execution directory. CAT can be used as follows;

CAT *.bak current directory

(In this example "*.bak" will be substituted for the "$0" in the macro text and "current" and "directory" substituted for $1, and $2 respectively. Since there were no other words on the command line $3 and $4 in the macro text will be null strings, so in this example the macro will execute as:

LS *.bak ! SORT ! PAG -C6 current directory

which will give a sorted paginated directory listing of all files in the current directory ending with ".bak". The page heading line will contain the heading "current directory".)

We can also invoke just:

CAT

Which will give a sorted paginated list of ALL files in the current directory, or

cat -t

Which will list all of todays files (the "-t" gets substituted for the $0 and becomes an option to "LS".)

A more complicated example using the .iferr .else .endif control structure follows:


```
OS9: macgen ed
MAC: del $0.bak
MAC: rename $0 $0.bak
MAC: .iferr
MAC:   edit $0 #20k
MAC: .else
MAC:   edit $0.bak $0 #20k
MAC: .endif
MAC:
```

This macro creates a command "ed" to edit a file and maintain a backup file with the suffix ".bak" on the filename. The first line will delete the old backup file ($0.bak , remember the filename you give on the ed command line will be substituted for the $0). The current versions of the del command print a message if the file does not exist and do not return an error, so no error handling is required after the del operation. The next line will attempt to rename the current file being edited to the same name with a ".bak" suffix. If the file does not exist then the rename command will return an error in which case the command(s) between the ".iferr" and the ".else" will be executed to edit a new file. If the rename succeeds then the file did exist in which case the statements between ".else" and ".endif" will be executed which will edit the backup file to a new file of the desired name. An example of using the ed macro command is shown below:

```
OS9: ed junkfile
```

In this example if junkfile did not already exist it would be created by edit. If it did already exist however, then the file junkfile.bak would be deleted (if it existed), then junkfile would be renamed to junkfile.bak, and then the command: edit junkfile.bak junkfile #20k would be performed which would edit junkfile.bak to junkfile, and preserve the .bak file.

It must be kept in mind that when you execute a macro you have generated, the commands composing the macro will be called at run time via shell and must either be present in memory or the current execution directory for the macro to function. The code in the executable macro file you generate will only call the other commands it is composed of, these commands are not included in the macro file automatically. In some cases it might be useful to merge the commands the macro is built of into the same file with the macro so they will all be loaded into memory at once when the macro is run. To do this for the CAT macro in the first example you could do the following:

```
OS9: chx /d0/cmds      Assuming this directory contains cat and the
                       other commands used
OS9: rename cat cat.temp
OS9: merge cat.temp ls sort pag >cat
```


OS9: attr cat e **set execute permission after merge**
OS9: del cat.temp

Merging all these commands into the same file will result in faster execution of cat in the case where ls, sort, and pag are not already resident in memory. In some cases it is better not to merge the commands into the same file with the macro. For example suppose you create a macro to do a compile and link operation where you need to run a compiler on the source file, then run an assembler, then a linker. If the compiler, assembler, and linker are all merged with the macro, then when you execute the macro all three will be loaded into memory at once perhaps leaving insufficient memory available for any of them to operate! If they are not merged with the macro they will each be loaded and run as needed, and when finished their memory will be released making room for the next command.

SPECIAL MACGEN PARAMETERS

The following words and symbols have special meaning to MacGen:

$0	substitute string 0 from command line
$1	substitute string 1 from command line
$2	substitute string 2 from command line
$3	substitute string 3 from command line
$4	substitute string 4 from command line
$5	substitute string 5 from command line
$6	substitute string 6 from command line
$7	substitute string 7 from command line
$8	substitute string 8 from command line
$9	substitute string 9 from command line
$$	Read line from standard input into command line at this location (the carriage return from the line read is not inserted into command line under construction. $9 is set equal to the first word on the line read. If eof is hit on input the current command line being constructed will not be executed, and the status will = 211 (eof error).
.IFERR [number]	Start of conditional section which is executed if error status from previous command equals number given. NOTE: Number is optional, if not given or zero the .IFERR evaluates as true for any error (any non zero value returned from command.
.IFNUL $parameter	Starts a conditional section which is executed if the associated $parameter ($0 ..$9) is null.

.ELSE Optional clause that may appear between .Iferr or .Ifnul and Endif statements. When the .IFERR statement associated with .ELSE evaluates false (no error or not the error number specified) then the statements between .ELSE and the next .ENDIF are executed

.ENDIF Closes the scope of the last .IFERR or .IFNUL statement.

.FOR {literal_number | $parameter}
Starts a loop for fixed number of repetitions given by literal_number or $parameter.

.NEXT Decrements the loop count associated with the last .FOR statement, if count is not zero control flow branches back to the first statement after the .FOR statement.

.REPEAT Marks the beginning of a .REPEAT loop.

.UNTIL [error number]
Tests the current status (from last command executed). If status is NOT equal to error_number, control will branch back to the last .REPEAT statement, otherwise control flow will continue with statement(s) after .UNTIL . If no error_number is given (or if 0) then any error status (status <> 0) will terminate the loop.

.CLEAR Clears the current status (sets it to zero). This statement is useful after the .UNTIL statement to clear the error status before exiting the macro (otherwise the error number will be reported if macro .UNTIL is last statement in the macro.)

.ONNUL $x $y string
Conditional parameter string set statement. The parameter $x {x=0..9} is tested to see if it is null (no characters). If so then the value of parameter $y is set to "string".

.ONSET $x $y string
Conditional parameter string set statement. The parameter $x is tested to see if it is set (not null). If so then the value of parameter $y is set to "string".

.CHD {pathname | $parameter}
Performs the same function as the shell "chd" parameter except that it's scope extends to the end of the macro.

.CHX {pathname | $parameter}
Performs the same function as the shell "chx" except that it's scope extends to the end of the macro.

NOTE: "string" value in .onnul and .onset statement is any group of non-space characters terminated by the first space or end of line. If no nonspace characters are present then the string is "null".

Words delimited by space characters on the command line when a macro is invoked are assigned to the $ string parameters in order of appearance on the line, i.e. $0 will have the value of the first word on the command line after the macro name, $1 the next and so on. When there are no more words on the command line then any remaining $ parameters are assigned a null value. The $parameters can appear in any order and any number of times in the macro text when you build a macro.

NOTE: the characters # ! > < & ; can not be passed as $ parameters since these are intercepted and interpreted by shell as having special meaning. These characters can however be used in the string set by .ONNUL and .ONSET so you can get around this restriction with those statements.

Below we will show a different way of writing the "ed" macro in the previous example so that an optional memory size for the edit operation can be specified.

```
OS9: macgen ed
MAC: .onset $1 $2 #
MAC: del $0.bak
MAC: rename $0 $0.bak
MAC: .iferr
MAC:   edit $0 $2$1
MAC: .else
MAC:   edit $0.bak $0 $2$1
MAC: .endif
MAC:
```

This "ed" macro will function the same as the first example with the exception you can specify a memory size parameter on the command line when you invoke it e.g.

```
OS9: ed junkfile 12k
```

When the ed macro is executed "12K" becomes parameter $1, since $1 is now not null, the value of $2 is set to "#", so when the edit command is called by the macro the $2$1 combination is replaced by "#12k".

If you had invoked ed with OS9: ed junkfile #12k you would not get the desired result because shell when invoking the ed macro would strip the "#12k" from the command line and give ed itself the 12k of memory (which is wasted), when ed is run in this case

it would not find any $1 parameter on the command line! When you are trying to figure out a complicated macro it is useful to define it with the -t option which will cause it to print the actual shell command lines it is using when it later executes.

Example: OS9: macgen -t ed (input to MAC: same as before)

CONTROL NESTING

All control structures may be nested, i.e. .FOR, .NEXT, .REPEAT, .UNTIL, .IFERR, .IFNUL, .ELSE, .ENDIF. If you forget the .next, or .until, or .endif statements to match the opening statement macgen will issue error 69 (Unmatched control structure) when you try to exit. If you try an illegal nesting (such as two .ELSE clauses inside an .iferr statement) macgen will immediately issue error 68, (illegal control structure). The Else clause is optional. If the Else clause is present it always belongs to the last .IFERR statement encountered. Endif is required to close the scope of an .IFERR statement. One .ENDIF is required for each .IFERR. Endif always closes the last open .IFERR statement. If you indent nested .IFERR statements and keep the associated .ELSE and .ENDIF statements aligned vertically it will be very easy to see the proper syntax. (Syntax is the same as for the Basic09 IF statement).

SPECIAL NOTES:

Macros can call other macros. The only limiting factor to how much you can put in a macro, or how many levels of macros can be called from within a macro, is memory usage. Each successive level macro call nested within a macro is guaranteed to use up at least 1k of memory.

You can redirect the input of macgen to a textfile containing the macro text. This will allow you to use EDIT or BUILD commands to create and edit the macro text and also maintain a listing of what is in the macro in case you forget. Macgen will still send the prompt "MAC:" to the standard output for each line in the file when the input is redirected.

Any macro text line beginning with an asterisk (*) is considered a comment only line.

If you are using the LS command to pipe filenames to the $$ parameter in a macro, do not use the -e or -f options with LS.

If you use any of the built in shell commands (called parameters in the shell documentation), i.e. chd, chx, kill, setpr ..etc. on a line alone in a macro, that line of the macro will have the effect of invoking a new shell. The word "shell" will print on the screen and the OS9: prompt will appear and the rest of the macro will appear not to function. If you do a "procs" at this time you will see the the macro is waiting for the new shell to die. If you hit the "eof" character (escape or shift/break on coco) then the macro will continue. The correct way to use the

built in shell commands in a macro is to follow them with the ; separator and then the next line of macro text:

```
e.g.    MAC: chd $0        WRONG
        MAC: dir

        MAC: chd $0;dir    CORRECT
```

When using the chd command as shown above with the ; separator and other commands on the same line, the scope of the chd will be only that one line. That means that the directory you do the chd to will be in effect for that line only and the next line will have a default directory of what ever was in effect before the chd. This is the reason for the ".chd" and ".chx" macro commands, these function the same as the built in chd, chx shell parameters except that their scope extends to the end of the macro.

MORE EXAMPLES

Macro to print a variable number of return address labels:

```
OS9: BUILD labels.mac
? .for $0
?   ECHO D. P. Johnson
?   ECHO 7655 S.W. Cedarcrest St.
?   ECHO Portland, OR  97223
? .next
? * Lines beginning with "*" are comment only lines.
? * space three lines to line up with next label
? .for 3
?   echo
? .NEXT
?

OS9: macgen -m <labels.mac
MAC: MAC: MAC: MAC: MAC: MAC: MAC: MAC: MAC: MAC: MAC:
```

This creates the macro in memory only (-m) using the text in "labels.mac" is the input.

```
OS9: labels 12
```

This will run the macro (12 is substituted for $0, so labels will print 12 labels (on the std output).

```
OS9: LABELS 12 >/P
```

Same thing except labels are directed to /p (printer).

Macro to rename a list of files to "name".bak :

```
OS9: MACGEN NAMEBAK
MAC: .REPEAT
MAC:   RENAME $$ $9.bak
MAC: .UNTIL
MAC: .CLEAR
MAC:
```

Example of use:

```
OS9: ls co* ! namebak
```

The ls command will list filename in the current directory beginning with "co" and pipe this list to namebak which will do a rename to the same name with an extension of ".bak".

You might want to rewrite the "namebak" macro above so it shows the filenames being operated on. E.g.:

```
OS9: macgen namebak
MAC: .repeat
MAC:  rename $$ $9.bak
MAC:  .iferr
MAC:  .else
MAC:    echo $9  renamed to:  $9.bak
MAC:  .endif
MAC: .until
MAC: .iferr 211
MAC:   .clear
MAC: .endif
MAC:
```

This rewrite illustrates several details you should be aware of; first the purpose of the ".iferr .else" after the rename command is so that the echo command will only execute if no error has occurred on the rename command. The $$ will read a name from the standard input. When end of file is encountered on this input we want to end the macro, and are depending on the eof error (#211) to terminate the .repeat .until loop. If we allow the echo command to be executed after eof is hit, the status that the .until statement will be checking will be that of the echo command which will not have an error and we will get an endless loop. Remember also that $9 will have the same value that was read in by $$, so we use this for all successive references to this name since if we repeated the $$ in the macro it would cause another name to be read from the standard input at that point. Finally, the .iferr 211, is used to clear the status (.clear) after the loop terminates at input eof. We are clearing only this code so that if the loop terminated for some other reason (file not found for instance) we will get a report of the error number.

NULDEVICE

The file "nuldevice" contains the device descriptor "nul" and device driver "nuldrv" to implement a nul device or "bit bucket" as it is sometimes called.

TO USE:

OS9: Load nuldevice

Output that you wish to discard may now be directed to the path /nul . (i.e. compiler or assembler output, etc.).

EXAMPLE: dir >/nul

Will cause the output of the dir command to go down the proverbial drain, (nowhere, discarded).

You may add the nuldevice to your bootfile with OS9GEN,

e.g.

OS9: chd /d0
OS9: os9gen /d1
os9boot
/d0/cmds/nuldevice

REP

SYNTAX: REP [-opts] command_text [$]

The REP command is a way of making any OS-9 command repetitive in the way that the commands in Filter Kit #1 are repetitive. This is best shown by an example:

OS9: ls ! rep ident -s $

In this example rep will repetitively invoke the "ident -s" command for each filename passed to it by the ls command. The $ tells rep to read 1 line from the standard input and insert the text at that point (there can be more than one $ on the line if you can think of a use for it, a new line will be read and inserted from path 0 for each $).

Options have form -A or -An where n is a decimal number associated with that option. Options are single letters A through Z (upper or lower case). Multiple options can be written as -A -B-C or -ABC .

OPTIONS:

-Cn The C option causes rep to repeat the command n times where n is between 1 and 65535. If 0 is specified the command will repeat 65536 times (or until aborted).

-T Causes rep to display each expanded command before it is executed in a similar fashion as the shell "T" option in a procedure file.

EXAMPLE:

OS9: rep -c4 list textfile >/p

This command line will list the file "textfile" to the printer "/p", four times.

NOTES: Rep will work properly with single commands. If you try to rep two or more programs directly that communicate with pipes something funny will probably result. You can rep a macro generated with macgen. If you need to repeat a process that consists of more than one command, use macgen to build a macro then rep the macro.

Don't confuse the "$$" used in macgen for this purpose with the "$" in the rep command. The rep substitution parameter is a single dollar sign, so using $$ with the rep command, will cause two lines to be read in from the standard input.

Example: The following shows usage of macgen and rep to run off return address labels on printer /p :

```
OS9: MACGEN LABEL
MAC: ECHO >/P D. P. Johnson
MAC: ECHO >/P 7655 S.W. Cedarcrest St.
MAC: ECHO >/P Portland, OR  97223
MAC: ECHO >/P
MAC: ECHO >/P
MAC: ECHO >/P
MAC:

OS9: REP -C50 LABEL
```

SIZE

SYNTAX: SIZE [-opts]

Reads a list of filenames through the standard input and reports total size of the list of files in number of files, number of bytes and number of sectors. This is useful to find the storage requirements for a group of files you intend to copy to another device to determine if they will fit, also useful to determine how much storage will be freed up by deleting a group of files. The number of sectors reported includes the "fds" file descriptor sector required for each file by RBF, but does not take into account directory space required for the name of each file. (Each file will consume 1/8 sector of directory space in the directory in which it resides.

Options have form -A or -An where n is a decimal number associated with that option. Options are single letters A through Z (upper or lower case). Multiple options can be written as -A -B-C or -ABC .

OPTIONS: -h Report numbers in hexidecimal instead of the default of decimal.

EXAMPLE: OS9: LS -E ! SIZE

```
    64  Files
117,381  Total bytes
   555  Total sectors
```

NOTE: When using the LS command to supply the file list to SIZE the use of the -e option with ls will speed up operation by as much as a factor of 10.

TOUCH

SYNTAX: TOUCH [-opts]

Reads a list of filenames through the standard input and sets the modification date and time of each file to the current date and time.

Options have form -A or -An where n is a decimal number associated with that option. Options are single letters A through Z (upper or lower case). Multiple options can be written as -A -B-C or -ABC .

OPTIONS: -L List each pathname to the standard output as it is touched.

EXAMPLE: OS9: ls -e *.co ! touch -l

The use of the -e or -f switches with the ls command will speed up operation of touch.

NOTE: User 0 may touch any file that has owner or public write permission. All other users may only touch a file owned by another user if the file has public write permission.

UNLOAD

SYNTAX: UNLOAD module_name [module_name ..]

UNLOAD repeatedly unlinks modules in memory until their link count reaches zero, which will cause their memory to be freed.

EXAMPLE: unload dir copy mdir

Removes "dir" "copy" and "mdir" from memory.