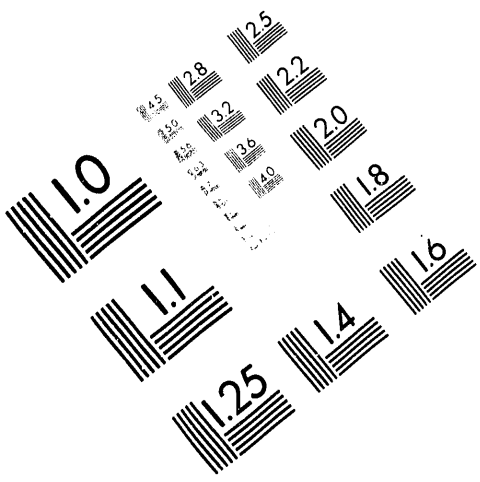
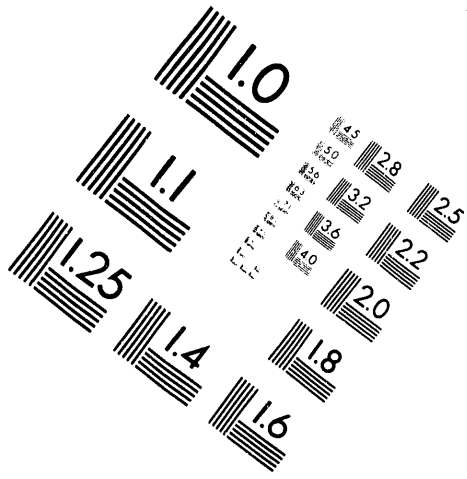
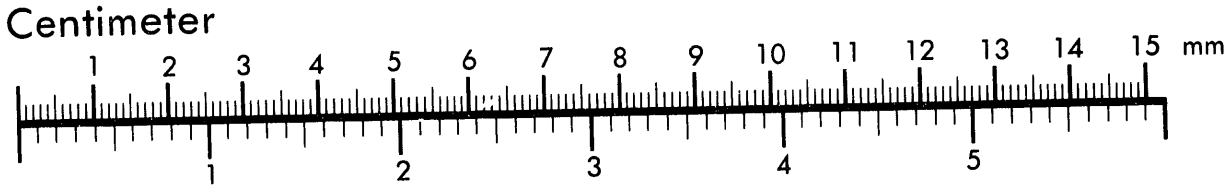
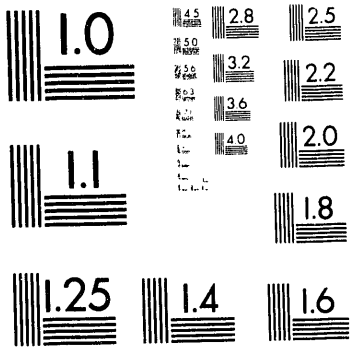
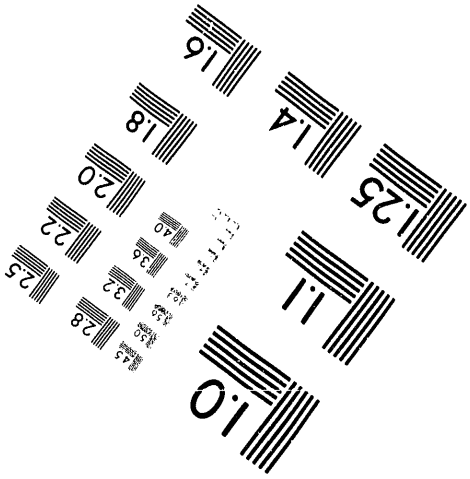


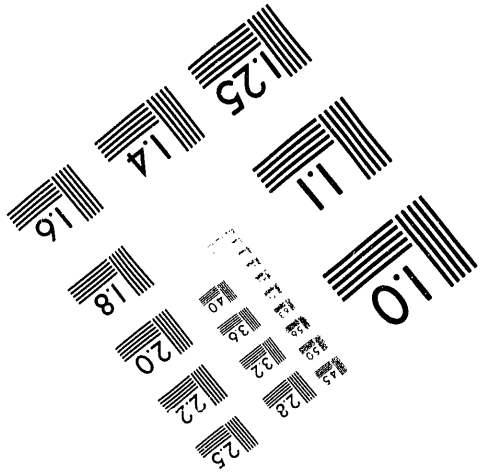
Association for Information and Image Management
1100 Wayne Avenue, Suite 1100
Silver Spring, Maryland 20910
301/587-8202











1 of 1

2

BNL-48955
INFORMAL REPORT

ACE PROGRAM/UNIX USER MANUAL

S. Kate Feng-Berman

January, 1993

Research Supported by the
OFFICE OF BASIC ENERGY SCIENCES

NATIONAL SYNCHROTRON LIGHT SOURCE

BROOKHAVEN NATIONAL LABORATORY
ASSOCIATED UNIVERSITIES, INC.

Under Contract No DE-AC02-76CH00016 with the
UNITED STATES DEPARTMENT OF ENERGY

ACE PROGRAM/UNIX

USER MANUAL

S. Kate Feng-Berman, NSLS, BNL
Jan. 12, 1993

Table of Contents

CHAPTER 0

RELEASE NOTE for version 2.2a

The version 2.2a program is called **ace** which stands for **Automatic Control Environment.** It's an updated version of the old **go** program, which includes a C-like syntax and some other new features. In this C-like syntax, users do not have to declare any variable used in the macro definition. The updates are the followings:

1 The line editor command is 'edit' instead of 'def' which is used to define a macro. Please reference section 1.2 : 'command edit' for the line editor 'edit', and section 1.1.2 and section 1.2: 'command def' about defining a macro.

2 The maximum number of arguments for a macro definition is ten which is represented by $1, $2, $3, $4, $5, $6, $7, $8, $9, and $10 respectively. The number of argument in a macro call is $#, and the whole line of arguments is $*. Please read section 1.2 for command 'def'.

3 There are three options (-f, -c, -m) when running **ace** program. They are : -f to load in a macro file, -m to load in a motor database file, and -c to load in a counter database file. Besides, you can use a "dofile" command to load in macro files up to the maximum of five files at a time (please reference section 1.2 for the command 'dofile').

4 There are several new reserved variables : POS[], OUTNAME, CNT[]. You can use a 'print' or 'printf' command to print out expression lists to the desired output. The default output is the console. However, you can use '> "filename"' or '>> "filename"' to overwrite or append the result into a specified file instead of the standard output. Furthermore, you can use '> OUTNAME' or '>> OUTNAME' to print anything to the data output filename specified by a 'newfile' or 'oldfile' command, or "disk.dat" file if the output filename is not specified by a 'newfile' or 'oldfile' command. This way, you can print a formatted string with variables such as motor position(POS[ ]), counts (CNT[]), a result of arithmetician operations, or even a comment to any output file. Please read section 1.2 command 'print' for details.

5 The 'datetime' command is replaced by the 'date()' command now. Please see command 'date()' for details.

6 The command 'tempfile' does not exist anymore since we can use the print command to print the result in any format we want into any file.

7 The format of a 'for' statement is changed to be C-language-like (i.e. nested statement is allowed). Please read section 1.2 command 'for' for details.

8 As I promised, now **ace** provides **if-else , while,** and **do-while** statements as in the C programming language (i.e. nested statements are allowed). Please read section 1.2 command 'if-else' for the **if-else** statement, and command 'do-while' for a **while** statement.

9 Since users can define their own variables (except those reserved variables mentioned above), the UNIX system command calls have to be preceded by a '$' symbol or use 'unix' command to avoid conflicts. Please read section 1.2 command **unix.**

10 In the 'PO' command, users can toggle between a linear(default) and log scale using the function key F4.

11 The online help command is 'h' instead of '?'.

12 The macro commands can be executed not only from a macro but also from the online input, like other online commands **bu, cu, po,** and so on. This is mentioned in section 1.1.

14 MCA support is much more user friendly after updating the MCA interactive graphical function, macro commands, and counter database. Please see section 1.3.1, 1.3.2 and 1.3.4 for those MCA features update.

15 Please do not forget to set the minimum speed on the E500 motors.

16 Macro command **setup** can be used to change the counter database also. Please see section 1.2 command **setup** and section 1.3.2 for counter database.

17 If a macro returns a value, please remember to use the '()' brackets to delimit the call by reference parameters which are seperated by comma. Please reference section section 1.1.2, and section 1.2: command **return.**

18 The format for **ca_read, ca_write, ca_str, gpib_read, gpib_write** are different now. Please reference section 1.2 for those commands.

Excuse me, the following is the 'to do' list.

1 Print out motor position in a different window.

2 Macro commands for MCA with (?) mark are not implemented yet!

RELEASE NOTE for version 2.0g

1 If you want to rescale during a scan, type F4. Then, the program will ask you if you want the auto scaling. The auto scaling will optimize the plot automatically according to the real data.

2 Any macro filename with an extension '.mac' will not have to type the extension

3 The commands 'mvc', 'mvf', and 'mvp' support multi-motor alignment to move those motor to the center, curve-fitted peak, and peak position. Please read section 1.2 about command **mvc, mvf, mvp.**

RELEASE NOTE for version 2.0a

The following is changed in the new release.

1 The graphics will stay opened once it is opened until the user quits the program. When the graphics is opened, it can be focused in/out, moved and resized by the mouse in the same way as the other windows controlled by the window manager.

2 The feedback counters can be mixed in with any scan counter in any order in the counter database file. The maximum number of counters in the database file is ten. Any feedback counter can be one of the scan counter. (i.e. They could be using the same counter.)

3 The macro file commands 'countdwell','countdelay','countlimit', and 'countpreset' update not only the current counting but also the counter database file.

4 The macro file commands are case sensitive. Most of them should be lowercase.

5 The movr command and read negg4.cnf command will not print the updated position and counts as a comment in the output file. If you want to print the update, you can add in a print statement right after the movr command or read counter command such as 'printf "pos %12.4f cnts %12.4f %12.4f ", POS[in1], CNT[a], CNT[b]'.

6 The move commands for on line commands and macro file commands are identical. All the move commands of on line commands are changed from 'm' to 'mv' (i.e. mr,ma, mm, mp, mc, mf, and mt is changed to mvr, mva, mvm, mvp, mvc, mvf, and mvt, respectively). All the move commands for a macro file are changed form 'mov' to 'mv' (i.e. like movr, mova, movm, movp, movc, movf, and movt is changed to mvr, mva, mvm, mvp, mvc, mvf, and mvt, respectively).

7 The on line command for alignments or scans is the same as the macro file command. Please use 'align' or 'scan' for a single motor alignment or scan, and use 'alignm' or 'scanm' for a multi motor alignment or scan. Also, you can change the align(scan) parameters (i.e. start position, end position, and step size) from the 'align' or 'scan' command. The sign of the step size determines the alignment

direction, but not the scan direction which is determined by the start and end position. Please see commands **align, alignm, scan, scanm** in section 1.2 for details.

8 In order to print a plot from 'PO' command, users must make sure the plotter number is set right in the counter database file. Please see section 1.3.2 for counter database item 26 which setup the plotter number.

CHAPTER 1

How to use the ACE program ?

1.0 Introduction to the ACE program

First, you need an account in the UNIX. If you do not have one, please see your system administrator.

The program **ace,** which stands for **Automatic Control Environment,** is an updated version of **go** program, and incorporates most of its user interface. The users define all the macros with a C-like syntax in one or several files, and then load the files with a 'dofile' command (section 1.2: command dofile), or load them with the '-f' option while running the 'ace' program. To create those files, you can use any editor or the **ace** builtin line-editor-command 'edit' (section 1.2 : command edit). Please read section 1.1.2 and section 1.2: command def about defining a macro.

After login to your account, please type 'xinit' to start the X window, and type 'cd xexp' after X window showing. This will bring you to the subdiretory 'xexp' under your account. Then you can type 'ace' with or without the '-m','-c', and '-f' options to start the program. When you type 'ace -m motor.cnf -c counter.cnf -f doall.mac', you automatically load a motor database (-m option), a counter database (-c option), and a macro definitions file (-f option). The **ace** program must be run under an X window for graphics. There are two database files required, one is the motor database file (discussed in section 1.2 command setup) and another is the counter database file (discussed in section 1.3.2). It is suggested that you name thoses files with the extension '.cnf'. As to the data output file, you can give a filename without an extension, and then **ace** will create a new file with a lowest extension higher than the existing files extensions (available form .000 to .999 only). If you give a filename with an extension, the data will be appended to the file if the filename already exists.

1.1 Online Command

You can type 'h' for help at command input line to get the command summary. The online commands are the followings, plus the macro commands described in section 1.2 and section 1.3.4.

align:	a motor ALIGNment scan. (align motor [leftoffset rightoffset stepsize filename)
alignm:	Multi-motors ALIGNment scan. (alignm motor motor .....)
ba:	motor dataBase file Append
bm:	motor dataBase file Make (overwrite/create)

bo:	Open another motor dataBase file
bu:	motor dataBase file Update
ca:	Counter database file Append
ca_read:	CAMAC Read data (example: ca_read(10, 17, 0))
ca_write:	CAMAC Write data (example: ca_write(10, 25, 8, 4500))
ca_str:	CAMAC String data
cm:	Counter database file Make (overwrite/create)
co:	Open another counter dataBase file
cu:	Counter database file Update
dofile:	To read in macro definition file(s)
edit:	A simple line editor to create/overwrite/append a macro file.
feedback:	FeedBack by a certain number of times specified in the counter database.
feedbackc:	Feedback Continuously.
gpib_read	GPIB Read data (example: print gpib_read(6,"IP"))
gpib_write:	GPIB Write data (example: gpib_write(6,"P123"))
hi:	HIstory listing of commands.
mva:	Move to Absolute position (mva motorcode position)
mvm:	Move Multiple motors at the same time. (mvm motor motor ....)
mvc:	Move to beam intensity center position. (mvc motor [motor....] counter)
mvf:	Move to beam intensity curve fit position. (mvf motor [motor ....] counter)
mvp:	Move to beam intensity Peak position. (mvp motor [motor....] counter)
mvr:	Move Relati ely to current position (mvr motorcode distance)
scan:	A motors Scan from one position to another. (scan motor [start end step-size filename])
scanm	Multi-motors Scan from one position to another.
setup	Setup motor or counter database file by the item number.
p :	move the same distance as last MR in (+) direction
n :	move the same distance as last MR in (-) direction
pa:	Plot All scans of one data file in one plot
pca:	MCA interactive graphical data control/collection/analysis with original data on all channels
pci:	MCA interactive graphical data control/collection/analysis with all channels' data to be zero
po:	Plot Old data
ri:	Read Intensity from data source

sp:	Set/read unit position and pulse position
wa:	Write the motor/counter information in ASCII file -> motor_cnt.asc
q :	Quit
!!:	repeat last command

1.1.2 Define a macro file

A macro is like a function in a C language, which consists of one to several commands. The command 'def' is used in a macro file to define a macro. Each macro file can have many macro definitions (each defined by a 'def' command). The format for the 'def' command is:

def macro_name '.....'

The first line of the definition must include **def macro_name '** followed with several lines of C-like-syntax commands, one command one line, or several commands in one line each separated by a **semicolon** (';') , and the macro definition is ended with another ' at the end. The C-like-syntax command could be a C-like statement (i.e.:'if', 'while', 'for', etc.), a C-like variable assignment (i.e. energy=12.4), a C-like arithmetic operation (i.e. energy/=2*cos(1)), a C-like input-output (i.e.: 'printf','getline'), a C-like string operation (i.e. 'sprintf', 'split'), an 'ace' builtin command (i.e.: 'mvr', 'scan'), or another defined macro command. All of them are described in section 1.2 for details.

Example:

```
def do_test 'unix("cd /usr/u1/feng/xexp")
if ($#!=2) { printf("Usage: do_test distance counter\n"); exit}
mvr theta $1; set_it 2.5
for (i=0; i<2; i++) {
read $2
mvr axe 2*i
}'
```

The first line starts the definition of 'do_test' macro, and changes the current directory to /usr/u1/feng/xexp so that the input and output file will be at /usr/u1/feng/xexp directory (this is a UNIX system command). The second line checks if the macro is called with the right arguments. The third line has two commands. The first command moves motor theta relatively according to the first argument, followed by a ';' as a command separator and the second command calls another defined macro command 'set_it' with one argument '2.5'. The line four to line seven consist of a loop which reads the counter specified by the second argument, and moves motor axe relatively two units in first loop and four units in second loop. When a function is called, the passed parameters can be a defined variable like the above example in line six, which uses loop number 'i' to decide how much distance to move motor 'axe'. Usually, the macro name does not have an extension, while the macro files have a '.mac' extension.

If a macro returns a value, please remember to use the '()' brackets to delimit the call by reference parameters which are seperated by comma.

Example:

```
def calc_it '
# This function retunrs a value
return(cos($1)+$2)'
```

In the above example, the fucntion **calc_it** is defined as another arithmatic operation which can be called like other builtin arithmatic operation (i.e. printf("cos(1.2)+3.5= %12.4f\n", calc_it(1.2, 3.5));). In the above function, '#' is used to start a line with a comment.

1.2 macro commands

The following commands are macro commands appearing in an alphabetical order, and they are case sensitive. Regarding the macro commands for the MCA devices, please reference section 1.3.4 for details. In the format description, all arguments enclosed by [ ] are optional, but they have to be given in the specified order if used.

Command : align

Purpose : If the step size is greater than zero, it performs an alignment scan from the [current position - scan left] to the [current position + scan right], and then returns to the current position. If the step size is smaller than zero, it reverses the alignment from the [current position + scan right] to the [current position - scan left], and then returns to the current position.

Format : align MotorCode [leftoffset rightoffset stepsize filename]

Example : align theta
: align theta 3 2 0.5 (It changes align left, right, and step size in motor database file first, then does alignment with the new argument)

Note: If the options within [] are specified, they have to be given in the above order, and the database file will be updated on the specified motor.

Command : alignm

Purpose : It performs a multi-motors' alignment

Format : alignm MotorCode MotorCode .....

Example : alignm theta ome pm

Command : break

Purpose : It terminates the innermost enclosing 'while' or 'for' loop. It is like a C language 'break' statement.

Format : break

Command : ca_read

Purpose : It gets data from a CAMAC crate.

Format : ca_read(slot# , function, address)

Example 1 : printf("The data read from CAMAC slot 2, func 0,addr 0 is %ld\n",ca_read(2, 0, 0));

Example 2 ret_value=ca_read(10, 25, 8)

Command : ca_str

Purpose : It sends a command string to a CAMAC crate.

Example 1 : ca_str("z")

Example 2 : ca_str("i")

Note : Only the following three commands are supported:

z : initialization
i : inhibit
c : clear inhibit

Command : ca_write

Purpose : It sends data to a CAMAC crate

Format : ca_write(slot#, function, address, data)

Example : ca_write(2 ,16, 0, 100)

Command : continue

Purpose : It passes control to the loop-test portion of the innermost enclosing 'while' or 'for' loop.

Format : continue

Command : countdelay

Purpose : It delays counting after motor finishing moving.

Format : countdelay DelayTime (in seconds)

Example : countdelay 10

Note It not only changes the counting delay time but also updates the counter database file.

Command : countlimit

Purpose : It setups the counter low limit

Format : countlimit #

Example : countlimit 1000

Note : It not only changes the counter low limit but also updates the counter database file.

Command : countdwell

Purpose : It setups the count dwell time in second.

Format : countdwell Time Base (dwell= Time * 10 (Base))

Example : countdwell 1 2 (dwell= 100)

Note : : It not only changes the counting time but also updates the counter database file.

Command : countpreset

Purpose : It setups the counter preset number

Format : countpreset preset#

Example : countpreset 1000

Note : It not only changes the counting preset value but also updates the counter database file.

Command : date()

Purpose : The date() function returns the current date and time as a string.

Format : date()

Example : print "#"date() > O[illegible] NAME (print current date and time as a comment in the current specified output file).

Command : def

Purpose : It defines a macro with arguments. As of today, the maximum number of arguments for a macro is ten, which is represented by $1, $2, $3, $4, $5, $6, $7, $8, $9, and $10 respectively. The number of argument in a macro call is $#, and the whole line of arguments is $*.

Format : def macro_name '

Note : A macro is defined within a pair of apostrophe ('....').

Example def scanloop '
if ($#!=5)

```
{
  printf("Input is %s\n",$*);
  printf("The number of input arguments must be five\n");
}
newfile $1
for (i=0; i<$2; i++) {
mvr in1 0.5
scan in2 $3 $4 $5
} '
```

After building the macro file called 'scanloop', you can call it with varied arguments like 'scanloop scan_test 2 5.0 10.0 1.0'.

Command : do-while

Purpose : It is a C-language-like 'while' statement to handle conditional events in a loop.

Format 1 : while (expression-list) { statement ..... }

Format 2 : do { statement ..... } while (expression-list)

Example 1 :

```
while ( POS[mon]>0.0)
{  mvr mon -2.5;  mvr the -2
read negg4.cnf
mvr the 2; mva mon 0
}
```

Example 2 :

```
do
{  mvr mon -2.5;  mvr the -2
read negg4.cnf
mvr the 2; mva mon 0;
} while ( POS[mon]>0.0)
```

Note : Nested do-while statements are allowed.

Command : dofile

Purpose : The macro defined in files are read in line by line.

Format : dofile file1 [file2 file3 file4 file5]

Note : You can load in macro files up to a maximum of five files at a time

---

Command : edit

Purpose : It is a simple line editor which can create, append, or overwrite a macro file without using the other complicated editor. It is a line editor. That is to say, you can only modify the current line, but not the previous lines.

Format : edit filename

Note : After you type **edit filename** followed with an ENTER, if the filename already existed, it will ask you whether you want to quit, overwrite or append the file. Then you have to start the editing with **def macro_name** ' , and end the editor with another '.

Command : exit

Purpose : It terminates the execution of the current macro, and jumps control back to command level. It is a C-language-like **exit** statement.

Format : exit

Command : feedback

Purpose : It performs a feedback function to tune up a system. It will work only if the feedback preset counter and up counter are setup and enabled for feedback. Please read section 1.3.3 for feedback counter setups.

Format 1 : feedbackc (It performs a feedback continuously. Type ^C to exit.)

Format 2 : feedback (It performs a feedback by a certain number of times which is specified in the feedback up counter database item 29: tuneup times/feedback.)

Command : for

Purpose : It is a C-language-like 'for' statement to do things in a loop.

Format
```
for (var=0; var<10; var+=2)
{
xxxx
}
```

Example
```
: for (i=1; i<6; i++)
{ mvr mon 2*i;  mvr in1 4.6;
read negg4.cnf
```

```
for (x=1; x<3; x++) mvr in3 cos(x);
}
```

Note : Nested 'for' statement is allowed.

Command : gpib_read

Purpose : It sends a read command through GPIB to a device, and then print the read string on the stdout.

Format : gpib_read(dev, command)

Example 1 : printf("The motor pulse position is %ld\n", gpib_read(6,"IP"));

Example 2 : read_pos=gpib_read(6,"IP");

Command : gpib_sread

Purpose : It does the same thing as gpib_read except it uses slow handshake sequence to read string one byte by one byte. It is used for a device with a slow handshake timing.

Format : gpib_sread(dev#, command)

Example : read_value=gpib_sread(3,"RC")/3.0;

Command : gpib_write

Purpose : It sends a write command through GPIB to a device

Format : gpib_write(dev#, command)

Example : gpib_write(10, 099999)
: gpib_write(6, "P20 X-1000 Y1000")

Command : if-else

Purpose : It is a C-language-like 'if' statement to handle conditional events.

Format 1 if (expression-list) { statement ..... }

Format 2 if (expression-list) { statement ..... } else { statement ....}

Example : distance=5.0; if (POS[the]>10.0)

```
{  mvr mon 2.5;  distance-=2.4;
read negg4.cnf; printf("counter red counts %12.4f\n",CNT[red])
}
```

```
else
{
distance +=2.4; mva mon 0;
}
mvr the distance;
```

Note : Nested if-else statements are allowed.

Command : index
Format index(s,t)
Purpose It returns position of string t in s, 0 if not present.
Example : index("banana","an")
(It returns 2.)

Command : length
Format length(s)
Purpose It returns length of s.
Example : length("bananna")
(It returns 7.)

Command : mva
Purpose : It performs an absolute move.
Format : mva MotorCode Position
Example : mva theta 0

Command : mvr
Purpose : It performs a relative move.
Format : mvr MotorCode Distance
Example : mvr theta 45

Command : mvm
Purpose : It performs a multi-motors' relative motion at the same time. (All the motors are moved relatively to their current positions)

Format : mvm MotorCode Distance MotorCode Distance ..... MotorCode Distance

Example : mvm t0 10 t2 5 a1 10

Command : mvp

Purpose : It does alignment scans and moves those scan motors to the beam Peak position. The number of alignment motors could be more than one.

Format : mvp MotorCode [MotorCode .....] CounterCode [filename]

Example : mvp the mon ind roi

Note: The **filename** is an option. If it is specified, a full filename must be given. If it is not specified, it is the default output file specified by 'oldfile' or 'newfile'.

Command : mvc

Purpose : It does alignment scans and moves those scan motors to the beam peak CENTER position. The number of alignment motors could be more than one.

Format :mvc MotorCode [MotorCode ....] CounterCode

Example : mvc the cnt
: mvc the ome cnt

Command : mvf

Purpose : It does alignment scans and moves those scan motors to the beam curve-fitted peak position. The ..umber of alignment motors could be more than one.

Format : mvf MotorCode [MotorCode.....] CounterCode

Example : mvf theta cnt

Command : newfile

Purpose : It specifies a data filename for the output.
It is required before data scan to specify the data output filename, otherwise data will be written to the programmed default data file 'disk.dat'. If the specified filename has no extension, it will open a new file with an extension whose number is one higher than the existing highest

extension. Otherwise, the new data will be written to the end of the specified file.

Format : newfile Filename

Example : newfile test.002 (The output file is test.002)

Example : newfile test (If the highest extension in the current directory for test is .010, then the output file is test.011)

Command : oldfile

Purpose : It specifies an old data filename for the output.
The given filename should not come with an extension. It will look for the current existing files with the same filename and with the highest extension number. The new data will be appended to the old file.

Format : oldfile Filename

Example : oldfile test1

Command : print (or printf)

Purpose : The command 'print' prints an expression list on an output, while the command 'printf' prints a formatted expression list on an output.
The default output is the console. You can use '> OUTNAME' or '>> OUTNAME' to print anything to the data output filename specified by 'newfile' or 'oldfile' command, or "disk.dat" file if the output filename is not specified by 'newfile' or 'oldfile' command. You can further use '> "filename"' or '>> "filename"' to overwrite or append the result into a specified file instead of the standard output. This way, you can print a formatted string with variables such as motor position(POS[ ]), counts (CNT[]), a result of arithmetician operations, or even a comment to any output file. Please remember to quote a string and the filename with " ". There are several new builtin variables : POS[](for motor position), OUTNAME (for data output filename), CNT[] (for counter data), the motor code and counter code read from the database file. All the builtin variables except 'OUTNAME' can not be specified a value by user. The motor-code variable or counter-code variable whose numeric value is the order of the motor or counter appearing in the database file, and motors, counters must not have the same name. In order to get the right numeric value for the motor-code variable or the counter-code variable, it is forbidden to use a number as the first character of the code, and the maximum number of characters for a code is three. The index for POS[] is the motor-code variable and the index for CNT[] is the counter-code variable. The arithmetic functions are as the followings:

acos(x)	returns the arccosine of x, in the range [0, PI]
asin(x)	returns the arcsine of x, in the range [-PI/2, PI/2]
atan(x)	returns the arctangent of x, in the range [-PI/2, PI/2].
atan2(y,x)	returns the arctangent of y/x in the range [-PI, PI].
cos(x)	returns the cosine of the argument, x, measured in radians.
exp(x)	returns exponential value of x. (e**x)
fabs(x)	returns the absolute value of x.
int(expr)	truncate to integer
log(x)	returns the natural logarithm of x. The value of x must be positive.
log10(x)	returns the logarithm base ten of x. The value of x must be positive.
pow(x,y)	returns x**y. If x is zero, y must be positive. If x is negative, y must be an integer.
rand()	returns random number between 0 and 1.
sin(x)	returns the sine of the argument, x, measured in radians.
sqrt(x)	returns the non-negative square root of x. The value of x may not be negative.
srand(expr)	returns new seed for random number generator; use time of day if no expr.
tan(x)	returns the tangent of the argument, x, measured in radians.
	The operators in increasing precedence are as the followings:
?:	conditional expression
|	logical OR
&&	logical AND
~, !~	regular expression match, negated match
+, -	add, subtract
*, /, %	multiply, divide, mod
^	exponentiation (** is a synonym)
++, --	increment, decrement
Format 1	: print expr-list >> "filename" (It prints expressions.)
Format 2	: printf(formatted expr-list) >> "filename" (It formats and prints like a C language 'printf' statement)
Example	: print "#this is a comment" > "/dev/lp1" (print a string to a printer) : printf("ome pos. %10.4f,fed counts %8ld\n",POS[ome],CNT[fed]) : print sin(1)+0.5*cos(1)

Command : read
Purpose : It reads intensities from a counter source.
Format : read CounterSource
Example : read egg4.cnf

Command : return
Purpose : It returns a value from a defined macro.
Format : return(value)
Example : return(sin(3.0)/2.0)

Command 9 : scan
Purpose : It performs a scan from a start position to an end position. Unlike an alignment, the direction of a scan is not determined by the sign of the step size, but the start position and end position. scan.
Format : scan MotorCode [start end stepsize filename]
Example : scan theta
: scan theta 12.0 2.0 1.0 (It changes scan start position, end position, and step size in motor database file first, then does alignment with the new argument)
Note: If the option within [] is specified, they have to be given in order, and the database file will be updated on the specified motor.

Command : scanm
Purpose : It performs a multi-motors' scan
Format : scanm MotorCode MotorCode .....
Example : scanm theta pm

Command : setup
Purpose : It updates the database parameters of the specified motor or counter.
Format 1 : setup MotorCode item# value item# value.....,e.t.c (update motor database)
Format 2 : setup CounterCode item# value item# value.....,e.t.c (update counter database)

Example : setup theta 5 0.01 9 1000

Please reference section 1.3.2 for counter database parameters. As to the item# for each motor database parameter, it is listed as followings.

1: Driver Type (ASCII string): The current supported motor driver type are 'MMC' for a GPIB system, and 'E500', 'DAC0' ([-2.5,2.5] volts : 12 bits), 'DAC1' ([0,5] volts : 12 bits), 'DAC2' ([0,10] volts : 12 bits), 'DAC3' ([-5,5] volts : 12 bits), 'DAC4' ([-10,10] volts : 12 bits), 'DAC5' ([0,10] volts : 16 bits) for a CAMAC system. The drivers 'DAC0' to 'DAC4' are designed for Joerger model DAC-16, which is a 16 channel, 12-bit-resolution digital to analog converter. The driver 'DAC5' is designed for Joerger model D/A-16, which is a dual channel 16-bit-resolution digital to analog converter. Monochrometer motor with nonlinear function between pulse position and unit (eV) position can be implemented by appending '_x25m' (for beamline X25) or '_x19m' (for beamline X19) after the driver type like 'mmc' or 'e500' (i.e.: e500_x25m, or mmc_x25m) .The scale factor and calibration nummber vary from different kinds of crystals used.

2: Motor Code (1-3 char.): A short name used as a code in the motor commands.

3: Motion Name (1-20 char.): A description of the motor function.

4: Slot Number (Integer): The slot number on the CAMAC crate, or the device number on the GPIB board.

5: Motor Address (Integer): Motor channel number or DAC channel module.

6: Unit Name (ASCII string): e.x. mm,inch,degree,...,etc. For DAC module, it is volts or voltages.

7: Minimum Speed (pulses/sec): The starting speed.

8: Maximum Speed (pulses/sec): The peak speed. If a DAC module is used, it is the number of bit pulses sent out on each ramp step (i.e. voltage change/ramp step).

9: Acc. Rate (pulses/S/S): The acceleration rate. If a DAC module is used, it is the delay time in milliseconds between each voltage change (ramp step).

10: Pulse Position (long int): The motor position in unit of pulses.

11: Backlash (Integer): The number of pulses for backlash. If the backlash value is positive, a backlash occurs only when the motor is moving in a positive direction. If the backlash value is negative, a backlash occurs only when the motor is moving in a negative direction.

12: BacklashOn	(Y/N): Enable/disable the backlash occurrence.
13: ActiveOn	(1/0=on/off): Set the motor to be active or not. If the motor is set active, its position will be printed each time before a command input prompt, and its database will be in the data output file.
14: Disabled	(1/0=yes/no): Disable or enable the motor motion by software.
15: Scale Factor	(pulses/unit): It is the number of pulses per unit. If DAC module is used, it is hard-coded (set by the computer), and the unit is bits/voltage.
16: Lower Limit	(Units): Software low limit.
17: Upper Limit	(Units): Software high limit.
18: Current Position	(Units): The motor position in unit.
19: Align Left	(Units): The alignment distance from the current motor position to the left. It is always a positive value.
20: Align Right	(Units): The alignment distance from the current motor position to the right. It is always a positive value.
21: Step Size	(Units): The step size in unit for the motor alignment and motor scan. The sign of the step size determines the direction of the alignment, but not the direction of the scan. If it is a negative value, then the motor aligns from the right to left.
22: scan Start Pos.	(Units): The motor scan start position in unit.
23: Scan End Pos.	(Units): The motor scan end position in unit.
24: Data Source	(ASCII string): The name of a counter database file which was set with all the counters' parameters for the motor scan.
25: Calibration no.	(float): The calibration constant to be used in the monochrometer motor.
26: Feedback on scan	(1/0=yes/no): Enable/disable the feedback during scan or alignment points.

Command	: sprintf
Format	sprintf(fmt,expr-list)
Purpose	It prints expr-list according to fmt, and returns resulting string.
Example	str= sprintf("Mo %s pos %12.4f",th2, POS[th2])

Command	: substr
Format	substr(s,i,n)
Purpose	It returns n-char substring of s starting at i; if n omitted, use rest of s.
Example	str=substr("North American",1,5) (It returns "North".)

Command	: unix
Purpose	: It is used to call a unix system command.
Format 1	: unix("command")
Format 2	: $command
Note 1	: You can define the often used system command in a macro file.
Note 2	To avoid conflicts, please do not define any variable starting with '$' or a digit.
Example 1	: unix("ls -l")
Example 2	: $ls -l
Example 3	: def l 'unix("ls -l " $*)' (make sure 'l' command is not defined yet)

1.3 Counters and MCA

MCA is a multi channel analyzer. In the **ace** program, we can use the **Region Of Interest** (called ROI) of the MCA as a counter. Besides, there are interactive graphical functions and macro commands for the MCA device to do spectrum anslysis and acquisition. The 1.3 subsections discussed all the details about the counters and MCA.

1.3.1 Counters usage

The data source in the motor database is actually a counter database filename. You can have different counter setups with different counter database files. There are three kinds of counter database supported. They are called egg4.cnf which is a setup for a EG&G 874 counter module, negg4.cnf which is a setup for a EG&G 974 counter module, and kcnt.cnf which is a setup for a Kinetic counter module 3640, or module 3610-L2A. Under each setup, you can have some counters set up from the ROI (Region Of Intreset) of an 'MCA'.

To use the kinetic counter, you need to use model 3640, or model 3610-L2A, a real time clock, and/or a clock generator. Then you should connect END signal of real time clock to INHIBIT signal of counter, and loop the PRESET OUT of real time clock to its START signal. If you want an external clock for the real time clock, take the signal from a clock generator to IN-A of real time clock.

When you set up the ROI of the 'MCA' as a counter, you have to gate the 'GATE' input on the back of MCA module with either the 'INTERVAL' output from an EG&G counter or the 'BUSY' output from a Kinetic counter to make MCA counting synchronized with either EG&G counters or Kinetic counters. Hence, the MCA ROI counters never exist alone in a counter database file. They are one of counters in either an EG&G counter database file or a Kinetic counter database file. Besides, to get the sum of serveral ROI, please do the following: 1. Make sure the LTB jumper is at the 'ROI OUT' position, 2. Feed the 'LTB' signal from the back panel of the 'mca' card into a EG&G counter or a Kinetic counter, 3. set those ROI number to be between 128 and 255. Furthermore, if you set the ROI number to be 0 in one of the ROI counter, you will get the counts of all channels in current group.

So far, this program supports up to ten counters in the database file. Any counter could be setup either for a scan or for a feedback. The feedback counter setup is discussed in section 1.3.3.

1.3.2 Counter database

The scan counters in the database file has the following parameters:

1: Driver Type — (ASCII string): The scan counters' driver type could be **egg4.cnf, negg4.cnf, kcnt.cnf,** and **mca.** When the driver type is **egg4.cnf, negg4.cnf,** or **kcnt.cnf,** it is also a counter database filename,

which is the data source parameter in the motor database file. When **mca** is a driver type of a counter, other counters in the same database file must include the other driver type like 'egg4.cnf', 'negg4.cnf', or 'kcnt.cnf', because 'mca' device is gated by other type of counter for counting synchronization. The follwing database parameters which are marked with a '*' at the end are those needed for 'mca' driver type. They are the slot number (group number), counter channel (ROI number), counter time (ROI's start channel), counter base (ROI's end channel), and so on.

2: Counter Code* (1-3 char.): A short name used as a code in the counter command.

3: Counter Name* (1-20 char.): A description of the counter function.

4: Slot Number* (Integer): The counter slot number on the CAMAC, or the device number on the GPIB board. If 'mca' is the driver type, it is the MCA group number.

5: Counter Channel* (Integer): The counter channel number. If 'mca' is the driver type, it is the ROI's number.

6: Unit Name (ASCII string): It is counts for the count-preset mode, or unit of time (i.e. second) for the timer mode.

7: Feedback?/Scan counter? (1/0 : Y/N) : It specifies whether this counter is used in a feedback or a scan. The following parameters are for scan counters, and the parameters for feedback counters will be explained in the following section 1.3.3.

8: Counting in a scan * (1/0 : Y/N): It is a flag for a counter's counting. If it is '0', the counter's counts will not be outputted.

9: Plot counter data * (1/0 : Y/N): It specifies whether or not to plot a counter's data.

10: Plot Data Symbol ? (1/0 : Y/N): It specifies whether or not to plot each counter data point with the associated symbol.

11: Timer scan/Preset count? (1/0): It specifies whether it is a timer mode or preset-count mode.

12: Preset counter ? (1/0 : Y/N): You should specify one of the counters as a preset counter which specifies the value of preset counts if you are using a preset-count mode for Kinetic module. If you did not give one, the default is the first counter for the scan and none for the feedback.

13: Beam monitor ? (1/0 : Y/N): If the counter's count is used as a beam monitor, then please input '1' here.

14: Count low limit (Integer): It is used as a value to monitor beam intensity so that the counter will not take the counts if there is a beam dump or a low beam. It is valid only on the beam monitor counter.

15: Counter Preset (long/counts): It is used if the counter is set for a preset mode. Usually a negative value, it is valid only on the preset counter. The preset value for the Kinetic counter has the following restrictions. If the absolute value of preset is smaller than 65536 (2**16), it could be any value. If the absolute value of preset is in the range of [65536,524,288(2**19)], it must be a multiple of 8. If the absolute value of preset is in the range of [524,288(2**19),(2**22)], it must be a multiple of 64. If the absolute value of preset is in the range of [(2**22),(2**25)], it must be a multiple of 512. If the absolute value of preset is in the range of [(2**25),(2**28)], it must be a multiple of 4096. If the absolute value of preset is in the range of [(2**28),(2**32)), it must be a multiple of 32768.

16: Counter Time* (Integer): It is used for a EG&G counter in timer mode (time=M *(10**N), where M is the counting time, N is the counting base). If 'mca' is the driver type, it is the start channel of the ROI number.

17: Counter Base* (Integer): It is used for a EG&G counter in timer mode (time=M *(10**N), where M is the counting time, N is the counting base). If 'mca' is the driver type, it is the end channel of the ROI number.

18: Counter Delay (seconds): The delay time in counting after motor's motion.

26: Plotter ? (0,1: Epson. 2,3,4,5: laser)

It specifies a plotter to be an Epson printer(0,1), a local laser printer(2,3), or the laser printer located at the hall of R&D group (4,5). It should be set on the first counter. For the laser printer, plotter 2 or 4 generates a landscape (horizontal) output, while plotter 3 or 5 generates a portrait (vertical) output. If a Laser printer is chosen to print an image, please click the mouse in the desired window when the cursor becomes a cross.

28: Real time clock slot — It is the CAMAC slot number of the real time clock (module 3640, or 3610-L2A) if a Kinetic counter is used.

1.3.3 Feedback Counter Database

The feedback counters could be one of the scan counters. To do the feedback, you have to set up a preset down counter, an up counter, and optionally a normalizing counter. Besides, if you want to enable the feedback tuneup during a scan, you have to set the enable flag to be one on the feedback preset down counter. The normalizer is used to normalize the beam intensity in case there is too much change in the beam intensity as time goes on. Also the normalizer counter can be used to monitor the beam intensity. If the normalizing counter is used, the actual counts from the up counter and the normalizing counter will be normalized by the previous normalizer's counts. The previous normalizer's counts can be setup by giving any reference value in the up counter. The feedback counters in the database file has the following parameters:

1: Driver Type — (ASCII string): It is the counter database filename.

2: Counter Code — (1-3 char.): A short name used as a code in the counter command.

3: Counter Name — (1-20 char.): A description of the counter function.

4: Slot Number — (Integer): The counter slot number on the CAMAC, or the device number on the GPIB board.

5: Counter Channel — (Integer): The counter channel number.

6: Unit Name — (ASCII string): It is counts for the count-preset mode, or unit of time (i.e. second) for the timer mode.

7: Feedback?/Scan counter? — (1/0 : Y/N) : It specifies whether this count is used in a feedback or a scan.

8: Feedback up counter?	(1/0 : Y/N): This is the counter which compares its counts with the feedback reference value used for tuneup. If this counter is not set in the database file, the feedback will do nothing.
9: Feedback enabled?	(1/0 : Y/N): Specified in the preset counter, it enables (1) or disables (0) the feedback capability of the program. To further enable a feedback during a scan, you have to enable the feedback in the scan motor database. (item 26 of the motor database)
10: Feedback mode?	(0:ref, 1:max,2:min): This is specified in the preset counter. In the reference mode, the up counter is compared with the multiplication of reference value (parameter 14) and offset (parameter 23). In the max. mode, the up counter is compared with its last count to get the maximum counts. In the min. mode, the up counter is compared with its last count to get the minimum counts.
11: Timer mode/Preset count?	(1/0): It specifies whether it is a timer mode or preset-count mode.
12: Preset down counter ?	(1/0 : Y/N): Used as a preset counter for tuneup counting period, it has to be set in the counter database file to do the feedback.
13: Feedback Normalizer?	(1/0 : Y/N): If you want to normalize the beam intensity, you should set a counter as a normalizer. After set, it will start counting in the preset counting period to get the normalized counts.
14: Count low limit	(Integer): Specified in the feedback normalizer counter, it sets the normalizer's low count limit so that the feedback will not happen if the normalizer counts is lower the low limit.
15: Feedback preset	(long/counts): It is the preset value for the feedback preset counter. Usually a negative value, it is specified in the preset counter for preset-count mode.
16: Counter Time	(Integer): Specified in the preset down counter, it is used for a EG&G counter in timer mode. (time=M *(10**N), where M is the counter time, N is the counter base)
17: Counter Base	(Integer): Specified in the preset down counter, it is used for a EG&G counter in timer mode. (time=M *(10**N), where M is the counter

time, N is the counter base)

18: Feedback loop delay — (seconds): Specified in the feedback up counter, it is the number of delaying seconds in the feedback loop between each tuneup.

23: Feedback offset — (0.0 - 1.0): Specified in the feedback up counter, it is the offset factor (between 0.0 and 1.0) from the reference value.

24: Feedback motor code — (up counter): Specified in the feedback up counter, it is the code of the motor which controls the feedback function.

27: Feedback reference — (up counter): Specified in the feedback up counter, it is the absolute value of feedback-up-counter's counts for tuneup. Whenever the reference value is setup in the up counter, it will change the normalizing counts also (if the normalizer exists!).

28: Scan points/feedback — (Integer): Specified in the feedback up counter, it is the number of scan points for each feedback action.

29: Tuneup times/feedback — (Integer): Specified in the feedback up counter, it is the number of tune up in each feedback loop.

1.3.4 MCA functions and macro command

The MCA interactive graphical functions will become available if you type 'pca' or 'pci' at the command line. Besides, if you set up one of the counters using MCA's ROI (section 1.3.2) , the interactive graphical function will appear automatically when you start the **ace** program unless all the counters using MCA's ROI are disabled for plotting. The interactive graphical functions are quite compatiable with those under the DOS Operating System except some functions are optimized for being user friendly.

The followings are macro file commands to control MCA (Multi-Channel Analyzer) device:

Command : MCA ACQUIRE OFF

Purpose : It stops MCA acquiring

Format : mca acquire off

Example : mca acquire off

Command : MCA ACQUIRE <mode>

Purpose : It setups MCA to be in PHA mode or MCA mode

Format : mca acquire <mode>

Example : mca acquire pha
: mca acquire mcs

Command : MCA ERASE

Purpose : It erases MCA memory

Format : mca erase <all>

Example : mca erase (erase the current memory group)
: mca erase all (erase all memory group)

Command : MCA ROI (?)

Purpose : It sets the MCA's Region Of Interest.

Format : mca roi <mode> [<region>] [all ROI#]
The option [all] is used only if <mode> is clear or set.
: If [all] is used for <clear>, it will clear all ROIs in all memory groups.
: If [all] is used for <set>, it will set the ROI number in all memory groups, and the specified region is an offset from the first channel of each group.

Example : mca roi on (display regions of interest)
: mca roi off (remove ROIs from display)
: mca roi clear (clear all ROIs in current memory group)
: mca roi set 100 120 (mark region <100 200> as a ROI)

Command : MCA SAVE

Purpose : It stores MCA's current memory group data in a binary file

Format : mca save <filename> [OVERWRITE] [ALL]
If the option [OVERWRITE] is used, a previously existing file of the same name is overwritten. Otherwise, the existing files will be protected.
If the option [all] is used, it will save all memory group data.

Example : mca save data.bin

Command : MCA SCAN (?)

Purpose It decides whether the mca counter should advance one group between each scan or alignment point. The default is off.

Format : mca scan <mode>

Example 1 : mca scan on

Example 2 : mca scan off

Command : MCA SELECT (?)

Purpose : It selects the memory group for the scan and display.

Format : mca select <group>

Example : mca select H0 (It selects group zero (4096 channels) among the two groups.)
: mca select Q1 (It selects group one of the four groups.)
: mca select E2 (It selects group two of the Eight groups.)
: mca select S5 (It selects group five of the Sixteen groups.)
: mca select T7 (It selects group seven of the thirty-two groups.)

Command : MCA TIME ACQUIRE

Purpose : It starts MCA acquiring data with a real time data graph unless' -n' is specified.

Format 1 : mca time acquire <seconds> [-n]

Format 2 : mca time <mode>

Example 1 : mca time acquire 0 (MCA starts acquiring until an OFF command to stop)
: mca time acquire 10 -n (MCA acquires data for 10 seconds without plot)

Example 2 : mca time live
: mca time real

The following is an example of using an MCA's region of interest as a counter in a scan. The descriptions are included in (). Make sure the MCA roi is included in the counter database file as a counter and enabled for counting (see section 1.3.2 for details).

newfile test (new data file scan.xxx for all motors' pos. and counts)
mca scan off
mca erase all (erase all of MCA-channels' data)
align th (start scan)

mca save kate all (save all MCA-groups' spectrum data in new file kate.xxx)

1.4 X window Interclient Communication (?)

Besides sending commands through keyboards, users can send commands to xexp program through their own programs from other windows by calling function xcom_wrt_str(). The detail procedures are as followings:

1: Users can send any command to the ace program by calling function xcom_wrt_str() passing the command string as the parameter.
Example: xcom_wrt_str("mr tt 0.1") or xcom_wrt_str("scan.mac").
The above "scan.mac" is a macro file created inside or outside a user's own program.

2: In the user's makefile, they have to call libraries in the following order: -lxcom -lX11 -lnsls_s.

3: Before executing, make sure the ACE program is running in the root window. At the end of communication, the screen will echo "done!".

CHAPTER 2

How to get around in UNIX ?

There are too many UNIX commands for our busy users to remember. Besides, many of the commands are user-unfriendly. After tailoring the UNIX commands, we can get around easier. The commands mentioned in this chapter are consist of the standard UNIX commands and the user-defined commands.

2.1 Floppy disk drive - Bread and Butter

If you have difficulty in using floppy diskettes drive, just remember how easy life is with 'bread and butter'. Where there is 'bread and butter', there is 'aread and autter'. 'bread, butter or bwrite' read, append, or write data to the 3.5" 1.44M bytes double side, high density floppy disk. The commands 'aread, autter or awrite' read, append, or write data to the 5.25" 1.2M bytes double side, high density floppy disk. The wild card usage for pathname applies as usual. As regards to other kinds of floppy disks, please see your system administrator.

In UNIX environment:

aformat = format /dev/rdsk/f0q15dt : format the disk on a 5.25" disk drive.

bformat = format /dev/rdsk/f1q18dt : format the disk on a 3.5" disk drive.

adir = tar tvf /dev/fd096 : dir files in the floppy disk of the driver

aread= tar xvf /dev/fd096 : read out files from the floppy disk.
example:

aread (read out all files from the disk)

aread *.c

aread pathname (read out one file from the disk)

(p.s. It will overwrite the files with the same name in the current directory.)

awrite= tar cvf /dev/fd096: write files into the floppy disk.
example:

awrite *.c (wildcard applies, it reads all C files th the disk)

awrite pathname (read out one file from the disk)

(p.s. It will overwrite any file which was originally in the disk)

On some computers with two floppy drives, the command will be aformat, aread, awrite for drive A, and bformat, bread, bwrite for drive B instead of aformat, aread, awrite.

In DOS under UNIX:

Please type "vpix" to enter the bridge between DOS and UNIX. In vpix, you can access floppy diskettes drive the same way as DOS (A:).

You can copy binary files directly between DOS and UNIX, but you have to specify a parameter to copy text files between DOS and UNIX.

To copy a DOS-format text file from a: driver to the hard disk, you can do the followings:
copy /u a:pathname pathname
(where /u means copying a DOS-format file into a UNIX format file)

To copy a UNIX-format text file from the hard disk to a: driver , you can do the followings:
copy /d pathname a:pathname

2.2 Other commonly used commands

pwd	print the current working director
cd	change the current working directory

2.3 Norton Utilities

Note: Norton utilities CAN NOT recover all the files deleted accidentally!

It will recover the file which was unique and was deleted with the command "rm filename". If the file was overwritten,it is impossible to recover it.

To recover the file you deleted, type "nue filename". To get a menu-driven interface with help information, speed search, and command history for the Norton Utilities, type "ni".

2.4 X window commands

To enter X window, type 'xinit'.

To create another window, you can do one of the followings: To close a window, you can just logout the window by moving cursor to the window, and then type ^D (CTRL+D).

2.5 Networking

To find out your computer system name, please type 'uname -n'.

2.5.1 Transfer files to and from a remote UNIX system host

To transfer file through Ethernet, please type 'ftp -n nodename' (e.x.: ftp -n bnlls1) . The remote nodename must be a UNIX system. Then type 'user' to get the prompt (username), and type in your login name, followed with an ENTER and your password. Now you can use 'cd' to change the remote computer's directory, use 'put pathname' to put a file to the remote computer, use simply type 'bye'.

2.5.2 login to a remote UNIX host

To login to a remote UNIX host, please use 'rlogin host -l username'. One example is 'rlogin bnlls1 -l feng'. From host bnlls1, you can login to a VAX/VMS host using 'dlogin hostname'.

DATE
FILMED
8/31/93
END