



ECi USERS CONFERENCE
Empowerment 2008

Power of Procs



Contents

About Procs	3
Creating a Proc File	3
Running a Proc File	4
Procs at a Glance	5
Using the Text Editor	6
Understanding Procedure Commands	7
Understanding Replaceable Variables	11
Using Procedure Commands	12
Understanding Error Messages	41
Understanding Error Message Descriptions	42
Troubleshooting.....	43
Appendix:A Proc Demonstration	44

About Procs

Your computer is good for more than calculating data and running reports. There are many repetitive tasks that must be completed at the computer every day. For example, you may have a daily routine that involves repetitive keystrokes. No matter how fast you type, this task still takes time. You can address this by setting up and launching procedure files (procs) to perform the keystrokes for you. Procs can quickly perform routine tasks even after hours, freeing you and your system for other tasks.

Creating a Proc File

You can create procs using the text editor or the word processor supplied with the DDMS system. Each line of the document is a proc command. Proc commands tell the proc executor what to do next, whether it is to run a program, to get input from the user or to display a message on the screen. Since the proc is a normal word processor file, it can be changed easily and quickly. The proc executor program is also called an interpreter. Each line of the proc is read and acted upon by the proc executor. The lines of the proc are read in order, unless the executor comes to a line that sends it to another place in the proc. Procs and their parameters may be placed anywhere on the line with at least one space between the procedure command and the parameter.

For example, each of the lines below is correct.

```
col. 123456789 etc.
|-+++++
  SHOWLN Hello World

          SHOWLN Hello World
^^^^^^- Notice that spaces are in front of SHOWLN

        RENAME          MYFILE 4
```

Continuation Character

A + (plus) character in column 80 of the line lets the next line in the proc be a continuation of the current line. This is useful for long RUN commands with extensive lists of input characters.

For example:

```
Column 80 -----+
          |
          v

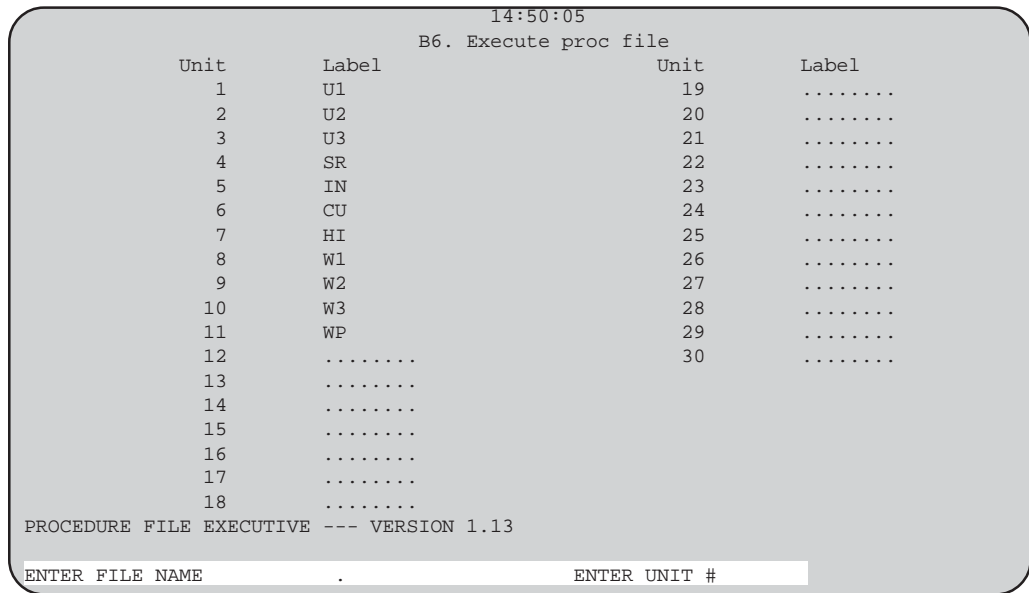
RUN ;NEW:R 4 THIS IS A LONG SEQUENCE OF INPUT CHARACTERS .....+
THIS IS A CONTINUATION OF THE PREVIOUS LINE
```

Running a Proc File

Procs are executed through the (Z) System Utilities screen.

- 1 In the (Z) screen Enter Utility Type field, type **B**.
- 2 In the Enter Subset Number field, type **6**. The (ZB6) Execute Proc File screen opens. See Figure 1.
- 3 In the Enter File Name field, enter the proc name which you created through the text editor or word processor.
- 4 In the Enter Unit # field, enter the unit number. The proc executes the commands until there are no more.

Figure 1: The (ZB6) Execute Proc File Screen



Procs at a Glance

- To schedule a proc, use the (Z)[E7] Activities Scheduler.
- TBLServer must be running at midnight.
- Basic characters:
 - For Enter, type ^ (carat)
 - For Escape, type ~ (tilde)
 - For Remark, type * (asterisk)
 - For Tab, type , (comma)
- Popular functions:
 - Kill Utilities
 - `RUN ;PRORESU6 4`
 - Execute Backup
 - `RUN ;PROBACK 4`
 - Restart Utilities
 - `RUN ;PRORESALL 4`
- When naming a procedure file, use the -NAME convention.
- Use the * command to create comment lines.
- Execute the SAVE [&1] command immediately prior to executing a program that uses a location.
- Use the GETUSAGE command to check disk usage and to prevent filling the unit.
- Check for disk errors when running a (Z)[C3] function when using ERASE, RENAME, or RECLEN commands. Exit the proc if necessary.
- Log messages to a file to produce a concise summary of what the proc has done.

Using the Text Editor

- 1 Access the text editor from either the Master Menu or the (Z) screen.
 - To enter the text editor from the Master Menu, type *
 - To enter the text editor from the (Z) screen, type **B1**

The (ZB1) Text Editor screen opens.

- 2 In the Enter File Name field, enter a file name, as shown in Figure 2.
- 3 In the Enter Unit field, enter the unit of the existing text file or the text file to create. If you do not fill the field, press Enter.
- 4 At the Is This A New File prompt:
 - Type **Y** to create a new file.
 - Type **N** to modify the existing file.
- 5 At the Enter Secondary Source File prompt, you can specify a second text file to edit at the same time. In the Enter File Name field:
 - Enter the file name. If you do not fill the field, press Tab. Go to **Step 6**.
 - If you are not editing a second file, press Enter. Go to **Step 7**.

Figure 2: The (ZB1) Text Editor Screen

```

15:01:08
B1. Text editor
Unit      Label      Unit      Label
  1      U1          19      .....
  2      U2          20      .....
  3      U3          21      .....
  4      SR          22      .....
  5      IN          23      .....
  6      CU          24      .....
  7      HI          25      .....
  8      W1          26      .....
  9      W2          27      .....
 10      W3          28      .....
 11      WP          29      .....
 12      .....      30      .....
 13      .....
 14      .....
 15      .....
 16      .....
 17      .....
 18      .....

EDITOR (I004)
ENTER PRIMARY SOURCE FILE
ENTER FILE NAME : [ANYNAME ] ENTER UNIT : [ ]
    
```

- 6 In the Enter Unit field, enter the unit of the second text file.
- 7 At the Enter Requested Function prompt:
 - Type **D** to display a text file on the screen and modify the file.
 - Type **P** to print a text file.
 - Type **R** to reindex a text file. (This should be done after lines have been added or deleted.)
 - Type **W** to look for a specific word, and replace it with another.
 - Type **F** to change to the second text file.
- 8 Within the file, you can use control commands to execute tasks within the text file. To use a control command, press the Ctrl key and another key.
 - Press Ctrl + **A** to add a character
 - Press Ctrl + **D** to delete a character
 - Press Ctrl + **E** to add a line (or section)
 - Press Ctrl + **X** to delete a line
 - Press Ctrl + **V** to mark the beginning of a section
 - Press Ctrl + **V** to mark the end of a section
 - Press Ctrl + **C** to move down one page
 - Press Ctrl + **O** to move up one page
 - Press Ctrl + **Z** to move up one line.
- 9 When you finish, you can exit the text editor. Press Esc until the Enter T = Terminate Editor prompt appears, then type **T**.

For details on using the text editor and word processor, go to http://www.ddms.com/resources/doc/standard_doc/Standard/BK7SYS/BK7VOL6.PDF.

Understanding Procedure Commands

The following table shows commands to use with the procedure executor, and their required parameters.

Command	Description	Parameters Required
*	denotes a proc remark - no action is performed	See Using Procedure Commands
ASSIGN	lets a proc program set a replaceable parameter	&<variable#> <contents>
BEEP	beeps the display terminal	See Using Procedure Commands
BEGIN	executes a program in the background (i.e. BATCH)	<program> <unit/volser> <device>
CHAIN	exits the proc and runs another program	<program> <unit/volser>
CLEAR	clears the display terminal screen	See Using Procedure Commands
DATECALC	lets a proc add/subtract days to/from a date	<date> <amount to add/subtract> &<variable#>
DATEDOW	returns the day of week for any date	<date> &<variable#>
DEC	decreases one of the user variables by one	&<variable#>
DSKCHKON	turns on file error checking for RENAME and ERASE	See Using Procedure Commands
DSKCHKOFF	turns off file error checking for RENAME and ERASE	
ENDIF	works with the IF command	See Using Procedure Commands
ENDTEXT	works with TEXT command to display text on screen	See Using Procedure Commands
ERASE	lets a file be deleted	<filename> <unit/volume>
EXIT	unconditionally stops execution of a proc	See Using Procedure Commands
FINDFILE	searches all hard drives for a file and returns the unit	<filename>
FLUSH	flushes the internal screen/program cache memory	See Using Procedure Commands
GET\$	lets you enter text into one of the variables	&<variable#>
GET%	lets you enter text with a given length	<length> &<variable#>
GET#	gets numbers only into one of the variable parameters	<length> &<variable#>
GETCHAR	gets a single character into one of the variable parameters	&<variable#>
GETMAIL	gets a number from one of the 16 internal mailboxes	<whichone (1-16)>.&<variable#>
GETUSAGE	returns the percentage in use on any disk unit	<unit/volume>
GOSUB	executes a routine and returns to where it left off	<label>

Command	Description	Parameters Required
GOTO	starts execution at different place	<label>
IF	performs command based on a condition	<NOT> <str1> <condition> <str2> <COMMAND>
IF#	performs command based on a numeric condition	<NOT> <str1> <condition> <str2> <COMMAND>
INC	increases one of the user variables by one	& <variable#>
INITIALIZE	lets a proc initialize a unit	<unit/volume>
KEYON	lets you abort a proc with a single keystroke	See Using Procedure Commands
KEYOFF	disables key checking while proc is running	See Using Procedure Commands
KEYPRESSED	lets a proc check if a key is pressed on keyboard	& <variable#>
LOCK	lets a proc lock the system program loader	See Using Procedure Commands
MENU	displays a menu and lets the user select an item	<col> <row> <promptchar>
MENUEND	denotes the end of menu selections	See Using Procedure Commands
PAUSE	temporarily stops execution of proc for a period of time	<ticks/seconds> SECONDS
POSITIONXY	positions cursor in a particular spot on terminal	<col> <row>
POSITIONXYC	positions cursor at the end of the line	<col> <row>
POSITIONXYS	positions cursor at the end of the screen	<col> <row>
PRETTY	prints a program source	See Using Procedure Commands
PRINT	outputs a hard copy of a document file to a printer	<filename> <unit/volume> <printer> <copies>
PRINTWP	uses new version of print program to print a file	<filename> <unit/volume> <printer> <copies> <a/o/e>
RANDOM	returns a random number for use with diagnostic routines	<bounds> & <variable#>
REBOOT	allows the proc to reboot the system	See Using Procedure Commands
RECLEN	returns the record length given the file and unit	See Using Procedure Commands
RENAME	gives an existing file a new name	See Using Procedure Commands
RESET	resets the screen table area	See Using Procedure Commands
RESTORE	restores a parameter from save area	& <variable#>

Command	Description	Parameters Required
RETURN	returns to the place from which a GOSUB was called	
RUN	executes a program	<program> <unit/volume> <input characters>
SAVE	stores a replaceable parameter in save area	&<variable#> <contents>
SENDMSG	sends a message to a device	<device> <message>
SETDEVICE	prepares a device for RUN commands	<device>
SETHI	highlights subsequent text	See Using Procedure Commands
SETLO	displays subsequent text at normal intensity	See Using Procedure Commands
SETMAIL	sets one of the 16 internal mailboxes	<whichone (1 - 16)> <to what (0 - 65535)>
SHOW	displays a message on the terminal	<text>
SHOWAT	displays text at specific location on the terminal	<col> <row> <text>
SHOWLN	displays a message on the terminal followed by cr/lf	<text>
STRMOVE	allows extraction or insertion of a string into a string	<position1 > <string1 > <position2> <string2> <length>
SYSRES	allows the proc to change the current system resident unit	See Using Procedure Commands
TEXT	displays following lines of text on screen until ENDTTEXT	See Using Procedure Commands
TRACEON	turns proc file debug on	See Using Procedure Commands
TRACEOFF	turns proc file debug off	See Using Procedure Commands
UNLOCK	unlocks the system after a lock command	See Using Procedure Commands
WAIT	holds until a user presses a key on the terminal	See Using Procedure Commands
WAITUNTIL	stops execution until a certain time has elapsed	See Using Procedure Commands

Understanding Replaceable Variables

Replaceable variables are used to insert certain text into the commands being processed. They are expanded before the command is executed.

Variable	Description
&1 - &99	Hold input from the GET\$ or ASSIGN command. These system variables give the proc writer convenient access to commonly used data such as the date, time, etc. These CANNOT be changed within the proc executive. Like the replaceable variables, these are expanded in the proc before execution.
&D	Inserts the current system date into the proc command.
&T	Inserts the current system time into the proc command.
&M	Inserts the current system month into the proc command.
&W	Inserts the current system day into the proc command.
&Y	Inserts the current system year into the proc command.
&H	Inserts the current system hour into the proc command.
&N	Inserts the current system minutes into the proc command.
&S	Inserts the current system seconds into the proc command.
&C	Inserts the type of OPD system currently operating (i.e. ASST for the ASSISTANT or OPD for the standard OPD-DEALER).
&V	Inserts the current version of the operating system (i.e. B115).
&E	Is used with the ERASE, RENAME, or RECLen command. &E contains a YES if a disk error has occurred, otherwise, it is blank. Some programs when used with the RUN command also set this variable, i.e., (Z)[C1] and (Z)[C3].
&F	Contains the percentage of disk space used from the last invocation of the GETUSAGE command.
&R	Contains the results from the last invocation of the RECLen command.
&B	Spaces that let the proc determine if one of the other variables is blank, i.e., IF &1 = &B SHOWLN THAT FIELD IS BLANK.
&K	Contains the text from the last KEYPRESSED operation: YES if a key is available; NO if no key is waiting.
&U	Contains the unit number after a call to the FINDFILE command. If the file was not found, it contains blanks. If the file was located on multiple units, it contains a 99; otherwise it contains the unit number of found file.
&Z	Contains the number of the menu item that the user entered.
&P	Contains the O/S type: R for real mode, P for protected mode.

Using Procedure Commands

Proc commands are listed below, along with their appropriate usage and proper syntax.

* Command (Remark Character)

Purpose: To let you imbed remarks in a proc file.

Usage: *

Example: *

```
* THIS IS A PROC TO PERFORM MY MONTH END FUNCTIONS  
*
```

Remarks in a procedure file are for internal information. They make the proc more readable to proc writers. Remarks in a proc show how the proc writer intended various parts of the proc to work. When the proc executive encounters a remark, no function is performed. The remark lines are simply skipped over.

Note: The asterisk character (*) must be located before any other text on the line for the remark to function properly.

ASSIGN Command

Purpose: To allow the proc to set one of the nine replaceable parameters to a given value.

Usage: ASSIGN &<variable#> <contents>

Example: *

```
* This is an example of the ASSIGN command  
*  
ASSIGN &1 Hello World  
SHOWLN &1
```

This example assigns the text, Hello World, to the user variable &1. The SHOWLN command following the ASSIGN command adds the &1 variable to the display. The ASSIGN command lets the proc writer store information in the replaceable variables without using the GET\$ command to do it.

Example: *

```
ASSIGN &1 FLAG1  
----- more processing here  
----- may set &1 to something else  
IF &1 = FLAG1 GOTO :LABEL1  
IF &1 = FLAG2 GOTO :LABEL2  
IF &1 = FLAG3 GOTO :LABEL3  
IF &1 = FLAG4 GOTO :LABEL4
```

Another function of the ASSIGN command is to set flags to alter flow of control in a proc. This is common in many programming languages.

BEEP Command

Purpose: To create an audible sound on the terminal. It usually signals the completion of a previous command or event.

Usage: BEEP

Example: * this is a proc to test the BEEP function.

```
SHOW PRESS ANY KEY WHEN READY
BEEP
WAIT
```

This proc displays the message, Press Any Key When Ready, then beeps and waits for the user to press a key. (See also **SHOW Command** and **WAIT Command**.)

BEGIN Command

Purpose: To start a program in the background on a batch, utility, commline or a printer device. It is generally used for programs that do not require any user intervention, such as sorting a file or printing a report.

Usage: BEGIN <program> <unit/volser> <device>

Where <program> is the name of the program being executed and <unit> is the disk unit or volume serial in which the program is located.

Note: For the executive to recognize that a volume serial is being used, it must be preceded by the @ character.

Example:

```
SHOWLN NOW RUNNING DAY END REPORT
BEGIN ;RPT:DAYEND 4 B:
OR
BEGIN ;RPT:DAYEND @SR?????? B:
```

This starts the program ;RPT:DAYEND on unit 4 in the first case and on SR?????? in the second case on the first available batch device (B:).

CHAIN Command

Purpose: To transfer control to another program, replacing the proc executive. This is like an EXIT command, except that it exits the proc to the program of your choice instead of MASTER.

Usage: CHAIN <program> <unit/volser>

Where <program> is the name of the program to execute and <unit> is the disk unit or volume serial in which the program is located.

Note: To signal a volume serial is being used, it must be preceded by a @ character.

Example:

```
SHOWLN NOW EXITING THE PROC TO RUN A CONVERSION PROGRAM
CHAIN ;SPC:CNVT 4
or
CHAIN ;SPC:CNVT @SR??????
```

This command transfers control to program ;SPC:CNVT on unit 4 in the first case and on SR?????? in the second case.

Note: Any lines following the CHAIN command are never executed. The proc is essentially terminated.

CLEAR Command

Purpose: To clear the terminal screen. It unclutters the display and makes it easier to read.

Usage: CLEAR

Example:

```
* proc to test CLEAR screen function
CLEAR
SHOWLN END OF PROC -- PRESS ANY KEY
WAIT
```

This command clears the screen and displays the message End Of Proc -- Press Any Key. The program pauses until the user presses a key. (See also **SHOWLN Command** and **WAIT Command**.)

DATECALC Command

Purpose: To calculate a new date from a given date.

Usage: DATECALC <date> <amount to add> <new date variable>

Where <date> is the date from which to calculate. It must be in the format of MM/DD/YY or MM/DD/YYYY. If the date is specified in any other way, a syntax error is generated.

<amount to add> is the number of days added to or subtracted from the first parameter, the date. If this number is positive (>0), the days are added to the date, yielding a date that is after the date specified. If the number is negative (<0), the days are subtracted from the date, yielding a date that is before the date specified.

<new date variable> is one of the 99 user variables in which to place the newly calculated date. If anything other than a user variable is specified, a syntax error is generated. The date returned is either in the format of MM/DD/YY or MM/DD/YYYY, depending on which date format was used in the first parameter.

Example:

```
* This is a simple proc describing the use of the
* DATECALC command.
CLEAR
* Calculate a date 10 days from today.
DATECALC &D,10,&1
SHOWLN The date that is 10 days from today is &1
WAIT
* Now calculate a date that is 10 days in the past
DATECALC &D,-10,&1
SHOWLN The date that was 10 days ago is &1
WAIT
*End of Proc
```

DATEDOW Command

Purpose: To calculate the day of the week from any date.

Usage: DATEDOW <date> <user variable>

Where <date>, in the format of MM/DD/YY or MM/DD/YYYY, is the date for which to return the day of the week. <user variable> is one of the 99 user variables in which to return the day of the week.

Example:

```
* This is an example of the DATEDOW command
CLEAR
DATEDOW &D,&1
SHOWLN The day of the week for today is ==>&1
WAIT
*End of proc
```

This proc gives you the day of the week (i.e. Monday, Tuesday, Wednesday, etc.) for the current system date. The day of the week is stored in &1.

DSKCHKON and DSKCHKOFF Commands

Purpose: To allow error checking to be disabled or enabled with the ERASE and RENAME commands. (See **ERASE Command** and **RENAME Command**.)

Usage: DSKCHKON
or
DSKCHKOFF

Example:

```
*
* This is an example of the DSKCHKON command
*
DSKCHKON
* From this point on file errors on ERASE, RENAME and RECLEN will be
* reported as error messages.
DSKCHKOFF
* Now error checking and reporting is OFF. Errors are still,
* however, contained in the system variable &E.
```

These two commands give you flexibility in handling the ERASE and RENAME functions. Sometimes it is necessary for a proc to disregard file errors in processing a file command (i.e. ERASE and RENAME). For example, you can ERASE a file if it exists and continue processing if it doesn't.

ENDTEXT Command

Purpose: To stop displaying text when used with the TEXT command. (See **TEXT Command**.)

Usage: ENDTEXT

Example:

```
*
* Proc to show the use of the ENDTEXT command
*
TEXT
These lines will display on the screen
until we come to an ENDTEXT command
ENDTEXT
```

The TEXT/ENDTEXT combination displays the text between these commands, up to but not including the ENDTEXT command.

ERASE Command

Purpose: To erase a file from the disk unit.

Usage: ERASE <filename> <unit/volume>

Where <filename> is the name of the file to erase. <unit/volume> is the disk unit or volume serial the file is on.

Note: If a volume serial is specified, it must be preceded by the @ character.

Example:

```
*
* This is a proc to test the ERASE function
*
ERASE MYFILE 4
*   or
ERASE MYFILE @SR??????
```

This command erases the MYFILE file located on disk unit 4, or as in the next line, deletes it from volume SR??????. If the operation is successful, the file is deleted; otherwise, an error message appears.

Note: Be careful not to delete files which are important to the operation of the system, i.e., libraries, user data files, etc. This operation completely removes the specified file.

EXIT Command

Purpose: To allow the program to stop execution at some point in the procedure file. This command may be used if the proc is interacting with you and you type QUIT to terminate the proc.

Note: If the proc does not have an EXIT command in it, the proc terminates after the last executable command in the file. Usage of the EXIT command is optional in some cases.

Usage: EXIT

Example: SHOW ENTER PROGRAM TO EXECUTE====>

```
GET$ &1
IF &1 = QUIT EXIT
```

This proc snippet compares the user variable &1 to the QUIT text. It exits the proc if this is true. (See **GET\$ Command** and **IF Command**.)

FINDFILE Command

Purpose: To let a proc determine the location of a file, eliminating the need for multiple lines of commands.

Usage: FINDFILE <filename>

Where <filename> is the name of the file to locate. The command searches all hard drives for the specified file and returns the result into one of the system variables (&U). If the file is located on one of the disk units, the number of that disk unit is placed into &U. If the file is not found at all, &U is blank and can be tested with the &B (blank) system variable. If the file is found on multiple disk units, &U contains a 99.

Example:

```
*
* This is a proc to demonstrate the FINDFILE command
*
SHOW Enter a file to search for ===>
GET% 10 &1
SHOWLN
SHOW Searching....
FINDFILE &1
SHOWLN
IF &U = &B THEN
    SHOWLN That file was not found
    WAIT
    EXIT
ENDIF
IF &U = 99 THEN
    SHOWLN That file was located on multiple disk units
    WAIT
    EXIT
ENDIF
SHOWLN
That file was found on unit &U
WAIT
```

This command prompts you for a file name, displays the message Searching as it looks for the file. The results are displayed for all the conditions that arise using this command. The file is either not found at all, it is found, or it is found in more than one place.

GET\$, GET% and GET# Commands

Purpose: To let you enter text into one of nine 30-character variables. Some variations restrict text length, or allow numbers only.

Usage: GET\$ &<variable#>

or

GET% <length> &<variable#>

or

GET# <length> &<variable#>

Where &<variable#> is one of the nine input variables. The procedure executive waits until you enter text (up to 30 characters). The first variable is &1, &2 is the second, and so forth, up to &99. The contents of the variables remain intact for the life of the proc or until another GET\$ command or ASSIGN command replaces them. (See also **ASSIGN Command**.)

Example:

```
* proc to demonstrate the GET$ function
CLEAR
SHOW ENTER YOUR NAME =====>
GET$ &1
SHOWLN
SHOWLN YOUR NAME IS &1
SHOW PRESS ANY KEY
WAIT
```

This command clears the screen and displays the message, Enter Your Name. The user's input is stored in variable &1. The proc now displays the message Your Name Is <name> where <name> is the text the user entered. The proc displays the Press Any Key message and waits for the user to press any key.

To specify the size of the input field, use the GET% command or the GET# command. The GET% command lets you specify how many characters the can be entered. For example:

```
GET% 10 &1
```

This lets the user enter up to 10 characters, which are stored in variable &1. The GET# command is similar to the GET% command; however, it only accepts numbers. This is useful for retrieving a unit number for a file operation.

GETCHAR Command

Purpose: To let the user enter text into one of nine 30-character variables.

Usage: GETCHAR &<variable#>

Where &<variable#> is one of the nine input variables. The procedure executive waits for the user to enter a character. The first variable is &1, &2 is the second, and so forth, up to &99. The contents of the variables remain intact for the life of the proc or until another GETCHAR command replaces them.

Example:

```
* proc to demonstrate the GETCHAR function
CLEAR
SHOW -- ENTER Y or N =====>
GETCHAR &1
SHOWLN
SHOWLN THE KEY YOU PRESSED IS =====> &1
SHOW PRESS ANY KEY
WAIT
```

This command clears the screen and displays the message Enter Y Or N. The user presses a key, which is stored in variable &1. The proc displays the message, The Key You Pressed Is <Key>, where <key> is the key the user pressed. The Press Any Key message appears until the user presses a key.

GETMAIL Command

Purpose: To store a value in one of 16 internal mailboxes.

Usage: GETMAIL <whichone> &<variable#>

Where <whichone> is a number from one to 16 which is the internal mailbox from which to retrieve the value. The user variable places the value in the specified mailbox. The values contained in these mailboxes range from -32768 to 32767 and are strictly numeric. Some programs use these mailboxes to store error return numbers when the program abnormally terminates. The GETMAIL command allows a proc to use a similar mechanism. (See **SETMAIL Command**.)

Example:

```
* This is an example of the GETMAIL command.
CLEAR
* First lets set a value in one of the mailboxes
SETMAIL 10,100
* Now lets retrieve the value we just set and display it
GETMAIL 10,&1
SHOWLN The value we saved in mailbox #10 was =>&1
WAIT
*End of proc
```

This proc demonstrates the use of the GETMAIL command with the SETMAIL command. Mailbox #10 is assigned a value of 100.

The GETMAIL command retrieves this value and displays it.

Note: The 16 available mailboxes are shared by all applications. If there are multiple procs running on multiple devices, it is possible for other devices to change the value of the mailboxes without notice. This can lead to inconsistent results. It could also allow multiple procs to cooperate by sharing a mailbox.

GETUSAGE Command

Purpose: To determine the capacity of a disk unit.

Usage: GETUSAGE <unit/volume>

Where <unit/volume> is the unit number or volume serial on which to find disk capacity. The capacity of the disk unit specified is expressed as a percentage used of the total capacity.

Example: * This is an example of the GETUSAGE command

```
CLEAR
GETUSAGE @SR
IF &F > 90 THEN
SHOW THAT DISK IS TOO FULL!
WAIT
ENDIF
```

This proc determines the disk usage of drive volume SR. It clears the screen, then retrieves disk usage. The IF command determines if disk usage is more than 90%. If the usage of that disk is over 90%, a Too Full message displays.

GOSUB Command

Purpose: To call a routine and return to the location from which the routine was called. The GOSUB command may be nested four levels deep. You can use a GOSUB command when a chunk proc code needs to be executed more than once. Instead of having duplicate sections of proc code, you can use a GOSUB routine.

Usage: GOSUB <label>

Where <label> is the name of the routine to call. Labels must start with the : (colon) character and may be followed by any sequence of characters. The routine MUST terminate with a RETURN command. (See **RETURN Command**.)

Example:

```

*
* This is an example proc to demonstrate the GOSUB command
*
CLEAR
SHOWLN This is an example to show how GOSUB is used.
GOSUB :DISPLAY_PROMPT
CLEAR
SHOWLN This is some more text
GOSUB :DISPLAY_PROMPT
EXIT
* We now start the subroutine
:DISPLAY_PROMPT
POSITIONXY 1 23
SHOW Press any key to continue
WAIT
RETURN

```

This proc displays text messages and calls the routine :DISPLAY_PROMPT. It prompts the user with the Press Any Key To Continue message. The subroutine lets you use fewer commands for common functions.

GOTO Command

Purpose: To transfer control of the proc to another command line.

Usage: GOTO <label>

Where <label> is the position in the proc to which to transfer control. Labels must start with the : (colon) character. They may be followed by any sequence of characters.

Example:

```

*
* this is a proc to show how to use the GOTO feature
*
* Clear the terminal screen
CLEAR
* the following command ":START" is a label
:START
* now display a message on the screen
SHOWLN HERE I AM - PRESS A KEY
* let the user terminate the proc by hitting escape if they want
WAIT
* heres a GOTO command now
GOTO :START

```

This proc clears the screen and displays the words, Here I Am - Press Any Key. When the user presses a key, the cycle starts over and the same message is displayed again. The proc terminates when the user presses Esc.

INC and DEC Commands

Purpose: To count up or down.

Usage: INC &<variable#>

Where &<variable#> is one of the user variables to increase or decrease.

Note: Make sure the user variable (&1 - &99) you specify contains a number, and not text. If the INC or DEC command is used on a text variable the results are undefined.

Example:

```

*
* This is an example to show the use of the INC and DEC commands
*
CLEAR
ASSIGN    &1  1
:COUNT_UP
IF NOT &1 = 99 THEN
SHOWAT   10 10 &1
INC    &1
GOTO  :COUNT_UP
ENDIF
:COUNT_DOWN
IF NOT &1 = 1 THEN
SHOWAT   10 10 &1
DEC    &1
GOTO  :COUNT_DOWN
ENDIF
:DONE

```

This proc clears the screen. Then the number 1 is assigned to the user variable &1 to initialize it to a known value. (It is a good practice to initialize variables before using them.) The user variable &1 displays at column 10, row 10 and increases incrementally by one. It continues to increase by one until the user variable is equal to 99. Next, the user variable reverses the process, decreasing incrementally by one.

IF, IF# - THEN and ENDIF Commands

Purpose: To execute a command if a condition is true (or false).

Usage: IF <NOT> <str1> <condition> <str2> <COMMAND>

or

IF <NOT> <str1> <condition> <str2> THEN

<COMMAND>

<COMMAND>

ENDIF

or

IF# may be substituted in the above expressions.

Where <str1> and <str2> are the text strings to compare and <COMMAND> is a valid procedure file command to execute if the comparison is true. <condition> is the type of comparison to make. Valid conditions are = (equal to), < (less than) or > (greater than).

The IF command compares text strings; the IF# command compares numbers. The variables compared must be whole numeric values and less than or equal to ten digits in length.

To see if two words are equal, type:

```
IF string1 = string2 GOTO :EQUAL_TO
```

To see if one string occurs alphabetically before another string, type:

```
IF string1 < string2 GOTO :LESS_THAN
```

To see if a string occurs alphabetically after another string, type:

```
IF string1 > string2 GOTO :GREATER_THAN
```

If the keyword THEN is found instead of a valid command then the commands following the IF command are executed. This allows faster processing of the IF command if multiple commands are required.

Note: If you use the IF - THEN command, you must use the ENDIF command after all of the commands.

If the <NOT> keyword is used, as shown, then <COMMAND>(s) is only executed if the condition is false.

Example:

```
*
* this is a simple proc to show how to use the "IF" command
*
:START
* clear the screen
CLEAR
* ask user to type in their name
SHOW ENTER YOUR NAME (OR QUIT TO EXIT) ==>
* input their name into variable 1 (&1)
GET$ &1
* see if they type "QUIT" and if not, start over
IF NOT &1 = QUIT GOTO :START
SHOWLN
SHOWLN THIS IS THE END - BYE!
```

This proc clears the screen and prompts the user to enter a name or type QUIT to stop. It saves the keystrokes in the variable &1. To see if the user typed QUIT, use the line:

```
IF NOT &1 = QUIT GOTO :START
```

If the variable &1 does not include the text QUIT, it starts the proc over.

Example:

```
*
* Proc to show the IF - THEN - ENDIF usage
*
:START
* clear the screen
CLEAR
* ask user to type in their name
SHOW ENTER YOUR NAME (OR QUIT TO EXIT) ==>
* input their name into variable 1 (&1)
GET$ &1
* see if they type "QUIT" and if not, start over
IF &1 = QUIT THEN
    SHOWLN
    SHOWLN THIS IS THE END - BYE!
    WAIT
    EXIT
ENDIF
GOTO :START
```

This example demonstrates the IF - THEN command. You can indent the commands following the IF <exp> THEN to show the nesting of commands. Nesting capability is virtually unlimited for IF - THEN commands.

Example:

```
IF <exp> THEN
    IF <exp> THEN
        IF <exp> THEN
            <command>
            <command>
        ENDIF
    <command>
ENDIF
ENDIF
ENDIF
```

KEYON and KEYOFF Commands

Purpose: To allow a proc to be aborted. By default, the KEYON command is active. During execution of the proc file, if a user presses a key, an Abort Or Continue message appears. However, with complex menu displays, pressing a key could disrupt the the proc display. The KEYOFF command makes the proc executive ignore any keystrokes until activated by a KEYON command.

Usage: KEYON

or

KEYOFF

Example:

```
*
* This is proc to show the KEYON and KEYOFF command
*
KEYOFF
- - - - - display a menu or something here with keystroke
- - - - - trapping disabled
KEYON
* reenable keystroke trapping
```

The KEYON command lets you gain control of runaway procs. For example, if there were an endless loop in your proc, you could press a key and abort the proc. If you use the KEYOFF command, however, the proc runs its endless loop and the user cannot intervene.

Note: The KEYOFF feature significantly speeds file processing since keystrokes are not checked between lines.

KEYPRESSED Command

Purpose: To determine if a key has been pressed.

Usage: KEYPRESSED &<variable#>

This command is useful for performing background tasks while waiting for the user to press a key. If the user presses a key, the keystroke is stored in the variable parameter you specify. The proc checks the &K system variable for the keystroke. This variable always contains the result of the last KEYPRESSED operation. If there is a keystroke, the &K variable contains a YES; otherwise, it contains a NO.

Example:

```
*
* This is a proc to demonstrate the KEYPRESSED command
*
SHOWLN
SHOWLN  Press a key to pause the time display
SHOWLN
:WAIT_FOR_KEY
    KEYPRESSED  &l
    IF  &K = NO  GOTO :DISPLAY_TIME
    SHOWLN
    SHOWLN  You pressed a key! That key you pressed was &l
    WAIT
GOTO :WAIT_FOR_KEY
* Display the time while not doing anything
:DISPLAY_TIME
    POSITIONXY   10 10
    SHOW &T
GOTO :WAIT_FOR_KEY
```

This proc checks for keystrokes. If no key has been pressed, the current system time displays on column 10, row 10 of the terminal screen. If a key was pressed it displays a message.

LOCK and UNLOCK Command

Purpose: To prevent other programs from loading.

Usage: LOCK

or

UNLOCK

This is a system-level command. It prevents the operating system from letting any other terminals or devices load programs. This is a dedicated operation and should be used with care. If an UNLOCK command is not generated sometime before the proc exits, the system could stay locked until you reboot.

MENU and MENUEND Commands

Purpose: To generate simple selection menus.

Usage: MENU <col> <row> <promptchar>
<menu selection 1>
<menu selection 2>
<menu selection 3>
<menu selection n>
MENUEND

Where <col> is the column position to start display of the menu selections and <row> is the line position to start displaying. The <promptchar> parameter is optional. This parameter displays the character of your choice beside a menu item as the user scrolls through the menu selections. This is used mainly for users with terminals that lack low or hi-intensity capability. The lines in the proc following the MENU keyword are the actual menu selections displayed. The maximum number of menu selections is 23. Using more than that results in a syntax error. The keyword MENUEND tells the proc executive when to stop displaying menu selections. You must end your menu selections with this command.

The MENU command displays the menu selections in a column. The first menu selection is highlighted. The proc controls the screen intensity, so any SETHI or SETLO commands issued before this are null. The user makes menu selections by pressing the Tab key, by pressing the Backspace key, by typing U, or by typing D. These keys highlight the next or previous menu selection. The Esc key lets the user abort the menu. Pressing Enter selects the current highlighted selection.

The results of the menu selection process are stored in the variable &Z. The menu selections are numbered starting at one for the first menu selection, and ending at 23 for the last menu selection. The last menu selection varies with the number of selections in your menu.

If the user presses the Esc key during the menu sequence, the &Z variable is blank, giving you complete control over the menu process.

Example:

```
*
* This is a proc to demonstrate the MENU and MENUEND commands
*
CLEAR
MENU 30 10
    THIS IS MENU SELECTION 1
    THIS IS MENU SELECTION 2
    THIS IS MENU SELECTION 3
    THIS IS MENU SELECTION 4
MENUEND
IF &Z = &B SHOWAT 1 23 THEY PRESSED ESCAPE!
IF &Z = 1 SHOWAT 1 23 THEY PICKED MENU SELECTION 1
IF &Z = 2 SHOWAT 1 23 THEY PICKED MENU SELECTION 2
IF &Z = 3 SHOWAT 1 23 THEY PICKED MENU SELECTION 3
IF &Z = 4 SHOWAT 1 23 THEY PICKED MENU SELECTION 4
WAIT
```

This proc clears the screen, then issues a MENU command to tell the proc executive that the next lines up to the MENUEND command are menu selections. The menu selections start displaying at column 30, row 10. The next four lines after the MENU command display with the first selection highlighted. The user can scroll through these selections and select one by pressing Enter. The user's selection is displayed at the bottom of the screen. Optionally, you can use the prompt character capability to make scrolling more visible.

Example:

```
MENU 30 10 >
....
....
MENUEND
```

This format is like the previous example, except that each currently highlighted menu selection is preceded by a > character.

PAUSE Command

Purpose: To hold the execution of the proc for a specific time.

Usage: PAUSE <ticks/seconds> SECONDS

Where <ticks/seconds> is the number of clock ticks or optionally the number of seconds to delay the proc.

Note: To use seconds instead of ticks, the keyword SECOND(S) must follow the number of seconds requested.

Example:

```
*
* proc to demonstrate the PAUSE function
*
PAUSE 36
* or
PAUSE 5 SECONDS
```

The PAUSE 36 line delays the proc for 36 clock ticks. The PAUSE 5 SECONDS line delays the proc for five seconds. There are about 18 ticks in one second, so 36 ticks delays the proc about two seconds. Use ticks for greater control.

POSITIONXY, POSITIONXYC, POSITIONXYS Commands

Purpose: To place the cursor at a specific location on the CRT screen. This command is useful for displaying messages at a specific location. Variations of this command move the cursor and clear the line to the end or clear the rest of the screen.

Usage: POSITIONXY <col> <row>

or

POSITIONXYC <col> <row>

or

POSITIONXYS <col> <row>

Where <col> is the screen column and <row> is the screen row at which to place the cursor. Most CRT screens are 80 columns wide by 24 rows in length.

Example:

```
*
* proc to demonstrate the POSITIONXY function
*
POSITIONXY 10 1
SHOW HELLO WORLD
WAIT
```

This proc moves the cursor to column 10, row 1, and displays the message, Hello World.

Variations: The POSITIONXYC command is similar to the POSITIONXY command, except that after it positions the cursor at the coordinates specified, it clears all text from the coordinates to the end of the line.

The POSITIONXYS command, like POSITIONXYC command, positions the cursor and clears text from that point to the end of the display screen.

PRETTY Command

Purpose: To print a program source file using the pretty print program.

Usage: PRETTY <filename> <unit/volume> <printer>

Where <filename> is the name of the program file located on <unit/volume>. The printer device is specified by <printer>. Replaceable variables may be used in all of the parameters. This program requires the ;CMP library to exist and the library name PRETTY to exist in the library.

All options available in the pretty print program use the defaults.

PRINT Command

Purpose: To print a document file to the printer of choice.

Usage: PRINT <filename> <unit/volume> <printer> <copies>

Where <filename> is the name of the document file located on <unit/volume>. The printer device is specified by <printer>. For multiple copies of the document, specify in <copies> the number of copies to print. Replaceable variables may be used in all of the parameters for the PRINT command.

Example:

```
*
* This is a proc to test the PRINT command.
*
PRINT MYFILE @SR????? P1 3
WAIT
```

This proc prints three copies of the document file MYFILE located on volume serial SR?????? on printer P1.

PRINTWP Command

Purpose: To print a document file to the printer of choice using the new document formatter and printer.

Usage: PRINTWP <file> <unit/volume> <printer> <copies> <pages>

Where <filename> is the name of the document file located on <unit/volume>. The printer device is specified by <printer>. For multiple copies of the document, specify in <copies> the number of copies to print. The <pages> parameter is used to print ALL pages, ODD pages or EVEN pages. If you do not specify a <pages> parameter, all pages print. Replaceable variables may be used in all of the parameters for the PRINTWP command.

Example:

```
*
* This is a proc to test the PRINTWP command.
*
PRINTWP MYFILE @SR????? P1 3 O
WAIT
```

This prints three copies of the document file, MYFILE, located on volume serial SR?????? on printer P1. Only odd pages print.

RANDOM Command

Purpose: To allow the user to generate a pseudo random number.

Usage: RANDOM <bounds> <user variable>

Where <bounds> is the maximum value for which to generate the random number. <user variable> is the user variable into which to store the random number result. This command can be useful for diagnostic work where a random value is needed to add a humanizing factor to a test. The random number generated ranges from 0 to <bounds>.

Example:

```
*This is an example of how to use the RANDOM command
CLEAR
* Lets generate a random number between 0 and 10
RANDOM 10,&1
SHOWLN The number is =>&1
WAIT
*end of proc
```

This example uses the RANDOM command to generate a number between 0 and 10, then displays the result on the screen.

REBOOT Command

Purpose: To allow the user to reboot the system.

Usage: REBOOT ALL

or

REBOOT LOCAL

Where ALL tells the proc executive to reboot all of the machines in a network configuration, and where LOCAL reboots only the machine running the proc. If the system is not networked, both commands have the same effect.

RECLEN Command

Purpose: To determine the record length of a given file.

Usage: RECLEN <filename> <unit/volume>

Where <filename> is the name of the file for which to find the record length and <unit/volume> is the disk unit or volume serial where the file is located. The record length for the file is stored in the system variable &R. If the file was not found and disk error checking is on, an error message is produced. If disk error checking is off, &E system variable is set to YES if the file is not found. (See also **DSKCHKON** and **DSKCHKOFF** Commands.)

Note: If a volume serial is specified, it must be preceded by the @ character.

Example:

```

*
* This is a proc to show the RECLLEN command
*
CLEAR
SHOW Enter the name of the file to show the record length for ::
GET$ &1
SHOWLN
SHOW Enter the unit # for that file ::
GET$ &2
RECLLEN &1 &2
SHOWLN
SHOW The record length for &1 on unit &2 is &R

```

This example prompts the user for a file name and unit and then uses the RECLLEN command to determine the record length of the file.

RENAME Command

Purpose: To allow a file to be given a new name.

Usage: RENAME <filename> <unit/volume> <newfilename>

Where <filename> is the name of the file to rename and <unit/volume> is the disk unit or volume serial on which the program is stored. <newfilename> is the new name.

Note: If a volume serial is specified, it must be preceded by the @ character.

Example:

```

*
* This is a proc to show the usage of the RENAME function.
*
RENAME MYFILE 4 YOURFILE
or
RENAME MYFILE @SR?????? YOURFILE

```

This command changes the file name MYFILE located on disk unit 4, to the name YOURFILE. The command on the second line renames MYFILE to YOURFILE located on volume SR??????. If the RENAME command is not successful, an error message is generated.

RESET Command

Purpose: To allow the user to reset the special screen area.

Usage: RESET

This command is most commonly used with the SAVE and RESTORE commands. It resets the special screen area used for inter-program communication. The RESET command clears the area, leaving it in a default condition to be used by other SAVE or RESTORE commands. (See also **SAVE Command** and **RESTORE Command**.)

Example:

```
* This is an example of the RESET command
CLEAR
RESET
RUN SOMEPROG 4
WAIT
* End of proc
```

RESTORE Command

Purpose: To replace variables with those saved in the special program save area. (See also **SAVE Command**.)

Usage: RESTORE &<variable#>

Example:

```
*
* This is an example to show the RESTORE command
*
SHOWLN INPUT TEXT BE SAVED
GET$ &1
* now lets save it away
SAVE &1
* now were going to blow that variable away
SHOWLN INPUT MORE TEXT
GET$ &1
SHOWLN you have just input ==> &1
* now lets bring it back to life
RESTORE &1
* we will now display what the user input originally
SHOWLN &1
```

First, this proc displays a message and waits for the user to enter text. The text is saved in program memory. The proc requests input from the user again using the same variable. The RESTORE command retrieves the text from program memory.

Note: The SAVE and RESTORE commands are normally used for inter-program communication if two programs agree on the SAVE and RESTORE sequence. In other words, SAVE and RESTORE provide a means for one program located on one device to pass information to another program located on a different device. Future applications may use this feature to integrate the proc executive into applications.

The SAVE command stores information until the next SAVE command, which silently overwrites any information previously saved there. The RESTORE command may be used to restore information stored by a SAVE command.

RETURN Command

Purpose: When used with the GOSUB command, to return to the point in the proc from which the GOSUB was called.

Usage: RETURN

RUN Command

Purpose: To execute a program from a proc either with supplied keystrokes for automatic execution, or without supplied keystrokes for interactive programs.

Usage: RUN <program> <unit/volume> <input characters>

Where <program> is the name of the program to execute. <unit/volume> is the disk unit or volume serial where the program is stored and <input characters> are the supplied keystrokes for the program.

Note: If a volume serial is specified, it must be preceded by the @ character.

Example: RUN ;NEW:MA 4 YACY ~

This command executes the program ;NEW:MA located on disk unit 4 with the input keystrokes, YACY <ESC>.

If the input characters are not specified, the program runs as usual: users are prompted for keystrokes where needed.

SAVE Command

Purpose: To let the proc writer save variables from the special program save area. (See also **RESTORE Command**.)

Usage: SAVE &<variable#> <contents>

Example:

```
*
* This is an example to show the SAVE command
*
SHOWLN INPUT TEXT BE SAVED
GET$ &1
* now lets save it away
SAVE &1
* now were going to blow that variable away
SHOWLN INPUT MORE TEXT
GET$ &1
SHOWLN you have just input ==> &1
* now lets bring it back to life
RESTORE &1
SHOWLN &1
```

First, this proc displays a message and waits for the user to enter text. The text is saved in program memory. The proc requests input from the user again using the same variable. The RESTORE command retrieves the text from program memory.

Note: The SAVE and RESTORE commands are normally used for inter-program communication if two programs agree on the SAVE and RESTORE sequence. In other words, SAVE and RESTORE provide a means for one program located on one device to pass information to another program located on a different device. Future applications may use this feature to integrate the proc executive into applications.

The SAVE command stores information until the next SAVE command, which silently overwrites any information previously saved there. The RESTORE command may be used to restore information stored by a SAVE command.

Tip: Normally, the SAVE command is used to pass application-specific information, such as the master location or the order-writer number, to the program being invoked. You can assign values to these variables using the ASSIGN command that can be used by the SAVE command in the proc.

Record Layout:

Variable Name	Length	Type	Position	Description
GLOBAL START				
TERMINAL	2,0	Binary	1	Terminal ID
UNIT_NBR	2,0	Binary	3	System library unit
MASTER_LOC	2,0	Numeric	5	Master location
INQUIRY	1	Alpha	7	Inquiry action
REQUEST	3	Alpha	8	Program request
WHO	4	Alpha	11	Order-writer
END GLOBAL				
Total Record Size = 14 Bytes				

Example:

```

ASSIGN &1 ' 99 '
*          ^^^^^^^^^^^^^^^
*          12345678901234
SAVE &1
RUN ';NEW:SR' 4 <put keystrokes here>
    
```

When &1 is used with the SAVE command, location 99 is stored at the appropriate location (position 5) in the record. Thus, when the program ;NEW:SR is run, location 99 is passed to it. This is the same as if the program were running on a terminal set up as location 99.

SENDMSG Command

Purpose: To allow text to be sent to a specific device or group of devices.

Usage: SENDMSG <device> <text>

Where <device> is the logical name of the device to which to send the text, i.e., T1, T5, T: etc., and <text> is the text message to send.

Example:

```

*
* This is an example of the SENDMSG command
*
* Send a message to all terminals
SENDMSG 'T:', 'Starting Backup on &D at &T'
.....<Command to backup system here>.....
IF &E = NO SENDMSG 'T:', 'Backup finished on &D at &T'
IF &E = YES SENDMSG 'T:', 'ERROR occured during backup on &D at
&T'

```

This example sends the text, Starting Backup on <current date> at <current time>, to all T: devices. The proc executive then backs up the system (commands omitted for clarity). Next, based on the result of the error return variable (&E), a message is sent to all T: devices indicating the results.

Note: This command works only with programs that intercept mail messages, i.e., the MASTER programs.

SETDEVICE Command

Purpose: To allow a proc to set the device to use for all RUN commands.

Usage: SETDEVICE <device>

Where <device> is the logical name of the device to use for all subsequent RUN commands, i.e., B1, B:, P: etc.

Example:

```

*
* This is an example of the SETDEVICE command
*
SETDEVICE 'B:'
RUN .....<Command to run here>

```

This example sets the device B: to run the program, instead of the default DY device.

SETHI Command

Purpose: To display text with a high intensity attribute. (See also **SETLO Command**.)

Usage: SETHI

Example:

```
*
* this will demonstrate the SETHI function
*
SETLO
SHOWLN THIS IS NORMAL INTENSITY
SETHI
SHOWLN THIS IS HI-INTENSITY
```

This example displays the message, This Is Normal Intensity. After the proc sets the intensity to high, it displays the message, This is Hi-Intensity.

SETLO Command

Purpose: To display text at normal (low) intensity.

Usage: SETLO

SETMAIL Command

Purpose: To allow the proc to set one of 16 internal mailboxes to a value.

Usage: SETMAIL <mailbox# (1 - 16)> <new value>

Where <mailbox#> is the mailbox to set. This may be in the range of 1 to 16. <new value> is a number in the range of -32768 to 32767 in which to set the specified mailbox.

Example:

```
* This is an example of the SETMAIL command.
CLEAR
* First lets set a value in one of the mailboxes
SETMAIL 10,100
* Now lets retrieve the value we just set and display it
GETMAIL 10,&1
SHOWLN The value we saved in mailbox #10 was =>&1
WAIT
*End of proc
```

This proc shows how to use the SETMAIL command with the GETMAIL command. After clearing the screen, mailbox #10 is set to the value of 100. Next, the GETMAIL command retrieves this value and displays it.

SHOW Command

Purpose: To display text on the CRT screen.

Usage: SHOW <text>

Where <text> is any text to be displayed on the CRT.

Example:

```
*
* This is an example of the SHOW command
*
SHOW hello world
```

This displays the message, Hello World, at the current cursor position. This command is often used after a POSITIONXY command, which places the cursor at a specific row and column.

SHOWAT Command

Purpose: To display text on the CRT screen at a specific location.

Usage: SHOWAT <col> <row> <text>

Where <col> is the column number and <row> is the row number of the display terminal at which to position the cursor before displaying <text>. Where <text> is any text to be displayed.

Example:

```
*
* This is an example of the SHOWAT command
*
SHOWAT 10 23 hello world
```

This displays the message, Hello World, at column 10, row 23. This command is a degree faster than using POSITIONXY then SHOW commands, because the lines are not read from the proc before execution.

SHOWLN Command

Purpose: To display text, followed by a carriage return.

Usage: SHOWLN <text>

Where <text> is any text to be displayed.

Example:

```
*
* This is an example of the SHOWLN command
*
SHOWLN HELLO WORLD
SHOWLN THIS LINE COMES AFTER HELLO WORLD
```

This displays the text, Hello World, with the line, This Line Comes After Hello World, displayed on the next line.

STRMOVE Command

Purpose: To let the user extract and insert characters in a string.

Usage: STRMOVE <pos1> <string1> <pos2> <destvar> <length>

You can extract characters from anywhere in a string and insert them anywhere in another string.

Where <pos1> is the position from which to start copying characters in string1. This number starts at 1 for the beginning of the string and cannot be greater than 160.

Where <string1> is the string from which to copy the characters. It can contain any variables needed.

Where <pos2> is the position in the destination string to which to copy the characters. This number can range from 1 to 160.

Where <destvar> is the user variable to which to copy the characters. It must be one of the 99 user variables.

Where <length> is the number of characters to move from the source string to the destination variable.

Example:

```
* This is an example of the STRMOVE command.
* What we're going to do here is extract the year
* from the system date and display it.
* The system date is in the form of :
* MM/DD/YY
* 12345678 <- This is the position of each character in the date.
* So, what we want to do is to copy 2 characters starting
* from character number 7.
CLEAR
STRMOVE 7,&D,1,&1,2
SHOWLN The current system date year is =>&1
* End of proc
```

This proc copies the two characters starting from character number 7. If the system date is 11/27/06, this proc displays 06.

The command, STRMOVE 7,&D,1,&1,2 breaks down as:

SRTMOVE: Command

7: Position from which to start copying

&D: System date variable

1: Point to which to copy characters

&: User variable to store result

2: Number of characters to copy

To extract the day from the date string, modify the STRMOVE command to start copying at the fourth position of the date string:

STRMOVE 4,&D,&1,2 copies the two day characters from the system date and place them in first user variable.

In these examples, the characters copied from the system date are placed in the first position of the user variable. You could, however, place them anywhere in the user variable. For example, if you have a complex keystroke response line set up for a RUN command, you could place the portion of the date extracted above anywhere in that keystroke response line by changing the position to which to copy the characters from 1 to any other position.

TEXT Command

Purpose: To display a large block of text on the screen. To stop display, the ENDTEXT command must be used. (See also **ENDTEXT Command**.)

Usage: TEXT

Example:

```
*
* Proc to demonstrate the TEXT command
*
CLEAR
TEXT
THE TEXT COMMAND WILL DISPLAY THIS TEXT
UPTO BUT NOT INCLUDING THE ENDTEXT COMMAND
ENDTEXT
```

TRACEON Command

Purpose: To give the proc writer a means to debug a proc. The TRACEON command displays the name of the proc command currently being executed. It continues to display commands until the TRACEOFF command is reached. (See also **TRACEOFF Command**.)

Usage: TRACEON

Example:

```
*
* This is a proc to show the usage of the TRACEON command.
*
SHOWLN PRESS ANY KEY WHEN READY ----
WAIT
TRACEON
SHOWLN TRACING THE PROC NOW....
TRACEOFF
```

This excerpt prompts the user with the message, Press Any Key When Ready. When the user presses a key, the trace mode begins. If a proc command has multiple parameters, these display as well, so you can verify what the proc executive sees when executing a proc.

When this proc is executed, the following is displayed:

```
PRESS ANY KEY WHEN READY ----
(User presses a key here)
TRACE IS NOW ON
SHOWLN
TRACING THE PROC NOW....
TRACE IS NOW OFF
```

TRACEOFF Command

Purpose: To give the proc writer a way to stop debugging a proc. It is used with the TRACEON command. If no TRACEON command is issued, the TRACEOFF command has no effect.

Usage: TRACEOFF

WAIT Command

Purpose: To cease execution of the proc until the user presses a key.

Usage: WAIT

Example:

```
*
* This is an example to show the WAIT command
*
SHOWLN PRESS ANY KEY WHEN READY ----
WAIT
```

WAITUNTIL Command

Purpose: To delay execution of the subsequent proc statements until a given time and/or date has elapsed.

Usage: WAITUNTIL <hh:mm:ss>
or
WAITUNTIL <hh:mm:ss> <day of week>
or
WAITUNTIL <hh:mm:ss> <@date>

Where <hh:mm:ss> is a valid military-style time. The second parameter is optional and can be either the day of the week or an actual date. The parameter for the day of the week is the name of the day, i.e., MONDAY, Tuesday, wednesday, THURSDAY, Friday. Note that the case of the letters is not important. Using a day of the week suspends execution until the day specified is reached. To use an actual date as the parameter, you must precede it with the @ character. The date specified can be either a standard date format, for example, @05/31/06; mm/dd/yy, or an extended form of the date with the full year specified, @05/31/2006.

When using the date format, you can also use a wildcard character on certain digits of the date. This is useful if, for example, you are executing the proc on the fifteenth of every month. To use wildcard characters, replace the digits with a question mark character (?). An example of this date in a proc would be @??/15/?????. This suspends the proc until the 15th of any month or any year. An entry of @05/??/? executes the proc on any day of May.

Example:

```

*
* This is an example of the WAITUNTIL command.
*
WAITUNTIL 01:00:00
* resumes processing here

```

This suspends the proc until 1:00 a.m., then resumes execution.

Note: It is very important to format the time properly. For example, use leading zeros and colons. An improperly formatted date can generate an error or prevent the proc's execution entirely.

Example:

```

*
* This is another example of the WAITUNTIL command.
*
WAITUNTIL 19:00:00 MONDAY
* resumes here on monday

```

This example suspends the proc until 7:00 p.m. Monday.

Example:

```

*
* This is an example of using a date with the WAITUNTIL command.
*
WAITUNTIL 19:00:00 @??/10/06
* resumes here

```

This example executes the proc at 7:00 p.m. on the tenth day of every month in the year of 2006.

Understanding Error Messages

When the proc executive encounters an error, it displays the general nature of the error, then the line of the proc where the error was found. Then the user is prompted to take some action.

For example, suppose the following line is in your proc: `RUN ;NEW:MA 56`.

Note that an invalid unit (56) is specified for this program. When the procedure executive tries to execute this line, it encounters an error and displays the following:

```

INVALID UNIT SPECIFICATION
ERROR ON LINE : 12
RUN ;NEW:MA 56
ABORT THIS PROC? Y/N

```

You can type **Y** to abort the proc, or type **N** to continue. If you type **N**, the **Re-type This Line** message displays.

You can either let the proc continue to the next instruction, or you can correct the error and the proc executive reexecutes the line.

The **Replace Line In Proc File** message appears. Type **Y** to update the procedure file with the corrected line. The next time the proc encounters this line, it is correct. This lets you fix the proc without using the text editor to make the change and run the proc again.

Understanding Error Message Descriptions

The following is a list of messages displayed, their probable cause and their solution:

- Error:** Invalid File Specification Or File Not Found
- Cause:** The procedure executive tried to run a program that could not be located on the specified disk.
- Remedy:** Check the disk for the file and modify the proc appropriately.
- Error:** Invalid Unit Or Volume Specification
- Cause:** The unit or volume serial specified for a RUN or BEGIN command was invalid.
- Remedy:** Change the proc to reflect the correct unit or volume serial and run the proc again.
- Error:** No ENDTEXT Command Specified
- Cause:** The proc executive reached the end of the file when displaying text used with the TEXT command.
- Remedy:** You must use the ENDTEXT command to terminate the display of text.
- Error:** GOSUBS Nested Too Deep -- Mamimum Is 4 Levels
- Cause:** The GOSUB command is restricted to four levels of nesting.
- Remedy:** Rewrite GOSUB command with no more than four nesting levels.
- Error:** Return Command Without GOSUB
- Cause:** The proc executive encounters a message, GOTO/GOSUB Label Not Found. The proc specified a GOTO or GOSUB labels that did not exist anywhere in the proc.
- Remedy:** Make sure that all GOTO/GOSUB label are preceded by a colon (:) and are on a line by themselves.
- Error:** End If For If <Exp> Then Command Not Found
- Cause:** The proc could not find a matching ENDIF command for an IF - THEN command.
- Remedy:** Make sure there are enough ENDIF commands for all IF - THEN commands.
- Error:** No MENUEND Command Specified
- Cause:** The proc could not find a matching MENUEND command for a MENU command.

- Remedy:** Make sure that there is a MENUEND command for all MENU commands.
- Error:** Procedure Syntax Error ---
- Cause:** There is a discrepancy with some parameter of a proc command, for example, an invalid time is entered, or an improper command format, etc.
- Remedy:** Fix the line in the proc and run it again.

Troubleshooting

When writing procs:

- Get it right the first time.
- Check keystrokes in appropriate screens.
- Use at least five ~~~~~ (Esc) at the end of a RUN command.
- Try out new procedures by running them manually using the (Z) [B6] function after backing up your system.

Maintenance:

- If you change a function password, remember to change it in your proc.
- ECi tries to keep keystrokes the same when a new software update is released, but this is not guaranteed. When you load a software update, check the RUN commands in your procs to make sure keystrokes have not changed.
- If you copy the library from a software update without using the INSTALL proc, the launcher program is disabled. You must run the launcher enable diskette again.

Appendix: A Proc Demonstration

The following proc is a demonstration of many features of Interactive Procedure Language. It can be entered as shown and run using the instructions described earlier in this handout.

```

*
* PROC TO DEMO THE NEW PROC INTERPRETER
*
* The proc starts here ----
*
:START_PROC
    CLEAR
    ASSIGN &9
    SETLO
    SHOWLN -----
    SHOWLN PROCEDURE EXECUTIVE DEMONSTRATION PROC - VERSION 1.0
    SHOWLN -----
    SETHI
    * Display the initial system time and date
    POSITIONXY 1 4
    SHOW &D
    POSITIONXY 65 4
    SHOW &T
    * Lets see what kind of computer we have here
    IF &C = ASST THEN
    POSITIONXY 21 4
    SHOW THIS IS AN ASSISTANT COMPUTER
    ENDIF
    IF &C = OPD THEN
    POSITIONXY 21 4
    SHOW THIS IS AN OP-DEALER COMPUTER
    ENDIF
    *
    * The next confusing looking lines show the menu and
    * highlights the letters in front of each selection
    * (a cleaner way to do this might be to use the MENU
    * command)
    SETLO
    POSITIONXY 20 10
    SETHI
    SHOW A.
    SETLO
    SHOW RUN A PROGRAM
    POSITIONXY 20 11
    SETHI
    SHOW B.
    SETLO
    SHOW FIND A FILE
    POSITIONXY 20 12
    SETHI
    SHOW C.
    SETLO
    SHOW PRINT A DOCUMENT
    POSITIONXY 20 13
    SETHI
    SHOW X.
    SETLO
    SHOW EXIT PROC TEST
    POSITIONXY 20 16
    SETHI
    SHOW ENTER LETTER OF COMMAND TO TEST === [ ]

```

```

                                SETLO
:GET_INPUT
                                POSITIONXY 57 16
                                PAUSE 20
                                KEYPRESSED &9
                                IF &K = NO GOTO :SHOW_DATE
                                * IF WE GET PAST HERE, SOMEONE PRESSED A KEY
                                IF &9 = A GOSUB :RUN_PROGRAM
                                IF &9 = B GOSUB :FIND_FILE
                                IF &9 = C GOSUB :PRINT_FILE
                                IF &9 = X THEN
                                    POSITIONXY 20 23
                                    SHOW END OF PROC --- PRESS ANY KEY TO EXIT
                                    WAIT
                                    EXIT
                                ENDIF
GOTO :START_PROC
* Show the system date and time on the top part of the screen
:SHOW_DATE
                                POSITIONXY 1 4
                                SHOW &D
                                POSITIONXY 65 4
                                SHOW &T
GOTO :GET_INPUT
*
* RUN A PROGRAM
*
:RUN_PROGRAM
                                CLEAR
                                POSITIONXYS 10 10
                                SHOW ENTER THE PROGRAM TO RUN - ENTER "QUIT" EXIT
                                SETHI
                                POSITIONXYC 10 11
                                SHOW PROGRAM NAME <<_____>> ON UNIT <<_>>
                                SETLO
                                POSITIONXY 25 11
                                GET% 10 &1
                                IF &1 = QUIT RETURN
                                POSITIONXY 48 11
                                GET% 2 &2
                                RUN &1 &2
                                * CHECK FOR NON EXISTENT PROGRAM
                                IF &E = YES THEN
                                    POSITIONXYC 10 12
                                    SHOW SORRY - THAT PROGRAM WAS NOT FOUND
                                    WAIT
                                    GOTO :RUN_PROGRAM
                                ENDIF
GOTO :RUN_PROGRAM
*
* FIND A FILE FOR THE USER
*
:FIND_FILE
                                CLEAR
                                POSITIONXYS 10 10
                                SHOW ENTER FILE TO SEARCH FOR - ENTER "QUIT" STOP FINDING F
                                POSITIONXYC 10 11
                                SHOW <<_____>>
                                POSITIONXY 12 11
                                GET% 10 &1
                                IF &1 = QUIT RETURN
                                FINDFILE &1
                                * CHECK FOR BLANK RETURNED UNIT #
                                IF &U = &B THEN

```

Power of Procs

```
                POSITIONXCYC 10 12
                SHOW THAT FILE WAS NOT FOUND
                WAIT
                GOTO :FIND_FILE
ENDIF
IF &U = 99 THEN
    POSITIONXCYC 10 12
    SHOW THAT FILE EXISTS IN MORE THAN ONE PLACE
    WAIT
    GOTO :FIND_FILE
ENDIF
POSITIONXCYC 10 12
SHOW THAT FILE WAS FOUND ON UNIT &U -- PRESS ANY KEY ---
WAIT
GOTO :FIND_FILE
*
* PRINT A DOCUMENT
*
:PRINT_FILE
    CLEAR
    POSITIONXYS 10 10
    SHOW ENTER THE DOCUMENT TO PRINT - ENTER "QUIT" EXIT
    SETHI
    POSITIONXCYC 10 11
SHOW DOCUMENT NAME <<_____>> ON UNIT <<__>> ON PRINTER <<__>>
    SETLO
    POSITIONXCY 26 11
    GET% 10 &1
    IF &1 = QUIT RETURN
    POSITIONXCY 49 11
    GET% 2 &2
    POSITIONXCY 67 11
    GET% 2 &3
    PRINT &1 &2 &3 1
    * CHECK FOR NON EXISTENT DOCUMENT
    IF &E = YES THEN
        POSITIONXCYC 10 12
        SHOW SORRY - THAT DOCUMENT WAS NOT FOUND
        WAIT
        GOTO :PRINT_FILE
    ENDIF
GOTO :PRINT_FILE
*
* E N D   O F   P R O C
*
```