

Changes And Additions For Version 9.0

Gee, this is an ugly document! I agree. However, my rationale for it is quite simple, I committed myself to releasing this software on the 25th of April (at the Chicago Mac Expo show), and it is now 6:45 in the morning on April 25th as I put the finishing touches on this document. Hopefully, it will be enough to explain those major differences between versions 8.0 and 9.0 until I can get a complete 9.0 manual in final form. The final manual will be available for downloading from the FreeSoft RoundTable on GENie just as soon as I finish it.

I will also be doing a separate "Masters Guide To Red Ryder Procedures" to explain the many uses of the additional Procedure commands added to this version, with a large sprinkling of examples, hopefully, this will make up for the terseness of the following descriptions of those commands. If you're not a very experienced programmer type, my recommendation is to read the Version 8.0 documents, and then read everything below up to the "New Procedure Commands" section to familiarize yourself with what's most likely to affect you.

The big surprise in this version is undocumented in this manual, and will remain so until I finish the tools necessary to allow non-Scott Watson's to take advantage of them. This is the Nautilus driver, which is an extremely powerful graphics driver that I'm very excited about. There are another half dozen or so Procedure commands that relate only to Nautilus drivers, but all will be made clear in a week or so. If you are not already, get online at the FreeSoft RoundTable (you must be a registered user to do so) so you won't miss any of the exciting announcements to come.

BUG FIXES

All reported bugs in version 8.0 have been corrected. Gee, that was easy to document... There have been the normal billion or so minor cosmetic improvements.

NEW FEATURES

1) Better than 100% throughput increase - 9.0 should easily run 2400 baud now full throttle with no buffering. There's still plenty of room for optimization and improvement if those of you with 2400 baud modems find I'm cutting it a bit close. I think we can achieve 4800 baud and hope to do that in the next version or so.

2) During a file transfer, the cursor now turns into an animated hourglass. There's some other "pretty-fying" graphics at other parts of the program, as you'll see. Who sez powerful has to be ugly, right? Before anyone asks, no it doesn't affect the speed of a file transfer.

3) There is a new Printer Preferences dialog box. This enables the user to specify the number of lines per page before doing a formfeed (should he decide that's desirable) and whether or not to put a time/date heading and page number on the first page, every page, or not at all. If you don't do any form feeds, the time/date item will only be done on the first page. I

think the form feed is also necessary to make the LaserWriter spit out a page. This dialog box applies to the "Echo To Printer" and "Print TEXT File" menu choices and their Procedure command equivalents. The box also allows you to force send a linefeed or formfeed to the printer by clicking in the appropriate button.

4) XMODEM receive throughput has been greatly increased. Runs like a bat out of Hell now. A bat with his butt on fire out of Hell... A bat... Well, coupled with a RAM disk I really don't think XMODEM or XMODEM/CRC can be made to run any faster than this on a Mac.

5) Strings sent by a Procedure TYPE command are now displayed when using HALF duplex. If you don't want something to be typed on the screen, like a password, simply switch to FULL duplex, TYPE the password, and then switch back to HALF duplex in your PROCEDURE.

6) Procedures no longer "lock out" the use of the mouse (but disable a few menu choices that would not be appropriate while a Procedure is executing). This means that you could have a Procedure running in the background to load in certain Macro Key files (and have the most often used functions displayed as mousable buttons in the Macros Status Bar) as the user branches to various menus within a service. The new menu command "Monitor Procedure" is equivalent to the Procedure command LOUD in its checkmarked state, and the Procedure command QUIET in its unchecked state. A Procedure now starts out in the mode specified by that menu choice. To enable the use of the various status bars while a Procedure is running, just uncheck the "Monitor Procedure" menu choice, or use a QUIET command in your Procedure.

7) Desk accessories can now be zapped "underneath" the Red Ryder window, and brought back on top using the "Bring Desk Accessory Windows To Top" menu choice. Note that having desk accessories active in this manner can cut down on throughput at the faster baud rates, but it makes the chore of doing a lot of little copies and pastes to RedWriter a lot easier.

8) CIS 'B' protocol downloads always use the small progress indicator. Why? Cause it was a quick fix for a problem in conjunction with the large progress indicator.

9) The "Edit" menu now contains an "Append To Archived Screens File" choice which does exactly that (or creates the "Archived Screens" file if it doesn't exist) with selected text.

10) The "VT100 Modes..." dialog box has two new additions, the "Mac Plus Keyboard" checkbox should be checked if you are using a Macintosh Plus keyboard. The layout of the keypad is a physical equivalent of the VT52/100 terminals' keypad layout, and so different values than what is printed on the keycaps may be sent by certain keys. There is now a "Show Keypad Mapping" button that brings up a picture of the mapping layout for those who need a quick refresher..

11) A "Buffered Keyboard" choice is available whenever you are using the 80 X 24 display screen mode. This holds everything you type in a "buffer" that is displayed at the bottom of the screen until you press the RETURN key. It then sends the entire buffer to the remote machine all at once. It does not buffer macro keys or control characters. I find this feature most useful for "Conferences" on hosts like CompuServe, Delphi, and GENie, and it can be especially useful with HALF-duplex systems to enforce the backspacing limit.

By far, the most drastic changes were made to the Procedure language. There are several commands no longer supported or replaced by new ones, and a whole bunch of new commands.

CHANGES MADE TO VERSION 8.00 PROCEDURE COMMANDS

SCREENCOPY - no longer exists or is supported. Replaced by SCREENPRINT and SCREENDISK as described below.

WAIT - no longer "freezes" up Red Ryder until the specified time. It simply does not execute the next Procedure instruction until the specified time - you can still use the rest of Red Ryder as normal.

PAUSE number - Upwardly compatible with 8.0, if no "number" is specified, it simply pauses Red Ryder for 1 second. Otherwise, the "number" is a number of 1/60th of a second increments to pause.

PAUSE 180 - would pause 3 seconds.

PANIC0

PANIC1

PANIC2

.
.
.

PANIC 9 - no longer exist or are supported. Replaced by PANICAFTER and ONPANIC commands described below.

File transfers and Redials executed by a Procedure that are aborted (by either a mousepress or error count) no longer abort the Procedure in progress, they instead set the YES/NO flag (described below) to reflect whether or not the intended function did or didn't work.

NEW PROCEDURE COMMANDS

SCREENDISK

Function: Equivalent to pressing the "Dump Screen To Disk File" button on the General Status Bar. Dumps a copy of the current display screen to the disk file "Current Screen", destroying any old data in that file.

SCREENPRINT

Function: Equivalent to pressing the "Dump Screen To Printer" button on the General Status Bar.

CURSOR H

Function: Turns the mouse cursor into an animated hourglass.

CURSOR A

Function: Turns the mouse cursor into the normal arrow symbol. The cursor is always initialized back to an arrow at the conclusion or cancellation of a Procedure file.

ONPANIC command

Function: When a "panic" condition exists, the Procedure command "command" is immediately executed. It's used in conjunction with the PANICAFTER command.

PANICAFTER seconds

Function: This command specifies the number of seconds to wait at the next PROMPT procedure command before falling into a "panic" condition. When a "panic" condition exists, Red Ryder will immediately execute the Procedure command (usually a JUMPTO or DO command) specified in the last ONPANIC command. Any time a PROMPT, or ALERT command is successful, the panic timer is turned off and must be reset using another PANICAFTER command before the next PROMPT command (if desired). Note that the Procedure command specified in a ONPANIC command is not lost when the panic timer is turned off, so it stays static until changed, or the Procedure terminates.

However, it's very important to have at least one ONPANIC command executed in your Procedure before any PANICAFTER commands so Red Ryder knows what to do in case of a "panic" condition.

REDIAL LIMIT number

Function: New FCC regulations specify that no automatic redialing device may not be allowed to progress past 15 consecutive tries, and this command tells Red Ryder how many maximum number of redial attempts to do before stopping and alerting you. "number" must range between 1 and 255 (we allow 255 because of many of our customers live in countries not governed by the FCC ruling - please adhere to any regulations that apply to you for the country you live in.

PROCEDURE VARIABLES

Up until now, the idea that Red Ryder (and many other communications programs) had a true command language was arguably farcical. A command language that cannot do real time polling and decision making based on the user's input is at best a robotic tool rather than an intellectual one.

I'd heard so many people talking about how CrossTalk XVI on the IBM-PC was the program to beat as far as a "script" or command language that I went out and bought a copy. Surprise! Red Ryder 8.0 already had it beat by a mile. At least I bought the damn thing on sale. The difference between Red Ryder 9.0 and other microcomputers communications software is a question of flexibility; it's like comparing a Jeep to a Railroad engine when you ask "where can I go with it"? For a person who is ready for this kind of power, I've tried to keep the functionality (but admittedly a different syntax) close to the BASIC language to keep the learning curve short.

There are ten "string" and ten "numeric" variables available for your use in Red Ryder Procedures. These variables are undefined (are considered to contain garbage or NIL values) at the startup of Red Ryder, but they do not lose their contents when a Procedure terminates. String and numeric variables may be used in place of many Procedure command parameters, can be tested, modified, erased, or used to send messages or ask for information from the user. More on that later. The way we refer to a string variable is with the ~ symbol (shifted version of the top leftmost key on the keyboard), followed by a number from 0 to 9.

In other words:

TYPE ~4

would TYPE the characters contained in string variable number 4.

Note that the command:

TYPE ~4^M

would not actually type the carriage return after the value contained in numeric variable 4. Instead, you must break the command into:

TYPE ~4

TYPE ^M

A string variable may be up to 80 characters long - DO NOT TRY TO PUT MORE CHARACTERS THAN THAT INTO ONE OR YOU COULD CORRUPT OTHER STRING VARIABLES OR EVEN PROGRAM EXECUTION!

Numeric variables can hold positive or negative values for values up to 1,000,000 in either direction. They are specified by using the `__` character (the unshifted version of the '~' key). and that character is again followed with a number from 0 to 9. In other words:

TYPE `4

would convert the number held in numeric variable #4 to a string of characters (the number 53 would be converted to the string "53") and that string would be TYPED to the modem.

There is an automatic conversion of a numeric variable to its string equivalent, but not the other direction. In other words, if numeric variable #3 held the number 300 and you did a:

PAUSE `3

Red Ryder would actually convert variable `3 to a string of characters, then convert that back to a number for the PAUSE command to use.

Here are some new Procedure commands for getting information in and out of string variables:

COPYINTO stringvariable,string

Function: "stringvariable" must be a string variable from ~0 to ~9, "string"

can either be a string variable a direct string of characters, or a numeric variable. This commands copies the contents of "string" into "stringvariable".

COPYINTO ~1,~2

would copy the contents of variable 2 into variable 1.

COPYINTO ~8,Hello World

would copy the string "Hello World" into string variable 8.

COPYINTO ~4,`2

would convert the number in numeric variable 2 to a string of characters, and then copy that string of characters into string variable 4.

CONCAT stringvariable,string

Function: adds (makes a concatenation to) the contents of "string" which may be either a direct string of characters or a string variable, to the

end of "stringvariable". If the result would be more than 80 characters long, the result is chopped to the leftmost 80 characters.

If string variable number 1 contained the characters "HELLO" and string variable number 2 contained the characters "THERE":

CONCAT ~1,~2

would not change the contents of string variable 2, but string variable 1 would contain "HELLOTHERE". Similarly, you could have done:

CONCAT ~1,THERE

to achieve the same result.

ERASE stringvariable or ALL

Function: Clears out the contents of a single string variable or all string variables.

ERASE ~3

would clear out the contents of string variable 3 only.

ERASE ALL

would clear out the contents of all 10 string variables.

SAVEVAR filename

Function: This command makes the limit of number of variables virtual rather than arbitrary. Have all 10 variables filled up but need another 1 or 2 for some temporary stuff? Use this command to save the current string and numeric variables into a disk file "filename". By doing this, it's conceivable for a large Procedure to have 40-50 string variables in use at once. The contents of the string variables are not changed by this command.

LOADVAR filename

Function: the opposite of SAVEVAR. Loads back in a set of string and numeric variables saved to the file "filename" using a SAVEVAR command.

Special Numeric Variable Commands

LET EQUAL numvar,string

Function: Note there is a space between LET and EQUAL! This simply assigns a value to the specified numeric variable "numvar". The following examples illustrate:

LET EQUAL `1,53

LET EQUAL `3,`4

LET EQUAL `8,~6

ADD numvar,string

Function: this adds the value specified in "string" to the numeric variable specified in "numvar". For example, if numeric variable 5 contains the value 75:

ADD `5,25

would leave numeric variable 5 holding 100.

SUBTRACT numvar,string

Function: The converse of the ADD command, in that it subtracts the value specified by "string" from the numeric variable specified in "numvar".

TEST numvar=string

TEST numvar>string

TEST numvar<string

Function: These 3 commands are used to test the numeric variable specified in "numvar" against the value specified by "string". They test equality, "numvar" is greater than "string", and "numvar" is less than "string". in order as listed above. All three set the YES/NO appropriately as to the true or false result of the test. Note that there are no spaces between "numvar", the test operator (=, >, or <), and "string". For example, if numeric variable 8 contains the value 25:

TEST `8=25

TEST `8>12

TEST `8<50

would all set the YES/NO flag to "YES".

String variables can also be used to replace direct strings in many Procedure commands. Here's a list of those I know about that can use string variables in ways that aren't documented elsewhere here. There may be others, but these I know for sure.

DO ~1

RUN ~1

DELETE ~1

RECA ~1

RECX ~1

RECK ~1

SENDA ~1

SENDX ~1

SENDK ~1

WRITE ~1

DIAL ~1

REDIAL ~1

PROMPT ~1

MACRO ~1

TYPE ~1

JUMPTO ~1

PAUSE ~1

ANSWERBACK ~1

LOADVAR ~1
SAVEVAR ~1
DEFINE MENU ~1
ADD TO MENU ~1
MENUDOES01 ~1
REDIAL LIMIT ~1

In the same manner, a numeric variable can be substituted in those above commands that expect a string of characters that evaluate to a numeric value.

THE TIME VARIABLE

There is also a variable used for keeping track of a certain starting time. This variable doesn't have a typed name, but is used internally. The following commands will make that a bit clearer.

SAVETIME

Function: This sets a "mark" in the internal time variable that will be used to calculate elapsed time with the ELAPSED command.

ELAPSED stringvariable

Function: This calculates the number of minutes that have elapsed since the last SAVETIME command has been executed (it will yield garbage if no SAVETIME command has been executed previous to this one). It converts that number to a string of characters and puts that string into the specified string variable. Notice how this timing mechanism is independent of the elapsed time clock as displayed in the General and Macros Status bars. You could do several SAVETIME and ELAPSED commands without resetting or changing the displayed timer.

So you might do:

SAVETIME

(and a few minutes later)

ELAPSED ~1

TYPE Elapsed time:

TYPE ~1

TYPE minutes.

TIMEDATE stringvariable

Function: This command takes the current time of day and date, converts it to a string in the format "MM/DD/YY HH:MM:SS" and puts it into the specified string variable.

FLAGS AND DECISION MAKING

There are two flags that can be used to execute a specified procedure command based on their state. There is a YES/NO flag and an ERROR flag. The ERROR flag is turned on or off for the disk I/O commands discussed later. The YES/NO flag is set according to tests made on a string variable.

EMPTY stringvariable

Function: This sets the YES/NO flag to indicate whether or not the specified string variable is empty.

CONTAINS stringvariable,string

Function: This sets the YES/NO flag to indicate whether or not the specified string variable contains "string". Upper and lowercase are significant!. As usual, "string" can either be a directly typed string or a string variable.

If string variable 1 contained the characters "Hello World":

CONTAINS ~1,Goodbye

would set the YES/NO flag to "NO".

CONTAINS ~1,ell

would set the YES/NO flag to "YES".

CONTAINS ~1,ELL

would set the YES/NO flag to "NO" (wrong case!).

CONTAINS ~1,Hello World

would set the YES/NO flag to YES (which is a good test for "equals" rather than just "contains").

CONVUP stringvariable

Function: This command converts any lowercase letters in the specified string variable to upper case, and is useful after getting input from the user through a QUERY1 command before doing any CONTAINS tests on the reply.

Once the YES/NO flag is set by the appropriate test, you should use one of the following commands to execute the desired Procedure command based on the results of that test.

IF YES command

Function: Immediately executes the Procedure command "command" if the YES/NO flag is currently set to "YES". An example might be:

IF YES JUMPTO (YESROUTINE)

IF NO command

Function: Immediately executes the Procedure command "command" if the YES/NO flag is currently set to "NO".

SETTING UP YOUR OWN DIALOG BOXES

Red Ryder 9.0 allows you to design and display your own dialog boxes as part of a Procedure. There are 3 types of dialog boxes that can be displayed:

- 1) A dialog box that allows the user to type in up to 80 characters as a reply.
- 2) A dialog box with Yes and No buttons.
- 3) A dialog box with an OK button.

The size and location of the dialog boxes are "hard-wired" into Red Ryder, but you can (and must) specify what prompting or message text is to be displayed in the dialog box. There are three lines of text that can be put into the dialog box, not coincidentally, up to 80 characters per line.

When you execute one of the dialog box commands, it puts whatever characters are in string variable 7 into the topmost line, the characters contained in string variable 8 in the middle line, and the characters contained in string variable 9 in the bottom line. If any of those variables are blank, so is the matching line of text in the dialog box (so you can do some primitive formatting and pretty-fying).

QUERY1 stringvariable

Function: Displays a dialog box with an editable text item of up to 80 characters, and a single OK button. Whatever the user types into the editable text item is copied into the specified "stringvariable" parameter when the user clicks on the "OK" button. Since there's no "Cancel" button, an easy way to tell if the user wants to abort would be to do an EMPTY test on the resulting "stringvariable". You may also wish to validate that the input was appropriate using various CONTAINS tests.

QUERY2

Function: Displays a dialog box with a single "OK" button. Not used to gather input, but just let the user know something.

QUERY3

Function: Displays a dialog box with a "Yes" button and a "No" button. This command sets the YES/NO flag depending on which button the user clicks on to get out of the dialog box. You could then do an IF YES and/or IF NO command based on that input.

DISK FILE INPUT/OUTPUT COMMANDS

If you thought all that stuff was good, you ain't seen nuthin' yet. Red Ryder 9.0 now has the ability to read and write disk text files. Sound simple? - heh, heh. What it means as far as horsepower is concerned is such Procedures as:

- 1) Batch uploading, downloading, and message sending Procedures, where you specify the batch through a text file or dialog box entry.
- 2) Procedures that look for "settings" files, and which can create or change current settings or options.
- 3) Procedures that can use simple input from the user through dialog boxes to generate and execute huge and complicated Procedures for the novice.
- 4) Procedures that keep a "log file" of things that happened during execution of an unattended Procedure.
- 5) Self-modifying Procedures (kinda reminds me of the CoreWars game) based on inconsistent host entry (sometimes I have mail waiting, sometimes I don't).
- 6) The ability to write your own Procedure language, with translation to equivalent Red Ryder Procedure routines. In other words, a picker/parser/mini-compiler.

There are two user "paths" that may be open at one time. They are known as path 1 and path 2. When you open a file for reading or writing, you assign it one of the unused paths. When you close a file, that path then becomes available for use with another file. In other words, you can have up to two file open for reading and writing at the same time. All paths are closed at the termination of a Procedure file.

USEROPENI path,filename

Function: Opens "filename" using path number "path" for Input (that's an "eye" at the end of USEROPEN). Input means that you'll be using only USERREAD commands with that path. If the file doesn't exist, the ERROR

flag is turned on, otherwise, it's turned off.

USEROPENO path,filename

Function: Opens "filename" using path number "path" for Output (that's an "oh" at the end of USEROPEN). Output means that you'll be using only USERWRITE and USERWRCR commands with that path. If "filename" already exists, it will be destroyed and recreated as an empty file by this command.

USEROPENA path,filename

Function: Opens "filename" using path number "path" for Append (that's an - ah hell...) Append means that if the file doesn't exist, it will be created as a new and empty file. If it does exist, any further USERWRITE or USERWRCR commands will be done at the end of the file, so previous data is not destroyed.

USERREAD path,stringvariable

Function: The procedure disk I/O commands are meant to be used with text files that have lines that are a maximum of 80 characters long each and end with a carriage return. This command reads data from path number "path" up to a carriage return, and puts that data into the specified string variable. If the read is unsuccessful (most likely tried to read past the end of the file) the ERROR flag is turned on, otherwise it's turned off.

USERWRITE path,string

Function: Writes the data in "string", which may be either a string variable or a direct string of characters to path number "path". It does not write a carriage return at the end of that data! If the write was unsuccessful (path hasn't been opened, disk full, etc.), the ERROR flag is turned on, otherwise, it's turned off.

USERWRCR path

Function: Writes a single carriage return to path number "path", effectively terminating a line for later input by the USERREAD command.

USERCLOSE path

Function: Closes path number "path" and makes that path available for a USEROPENI, USEROPENO, or USEROPENA command. If the path is not currently open, this command does nothing - so use it if you're at a place in your Procedure where you're not sure if a path is in use or not.

IF ERROR command

Function: If the ERROR flag was turned on by the last Disk I/O Procedure command, the Procedure command "command" is immediately executed. Otherwise, if the ERROR flag is not turned on, this command does nothing. Very useful for testing to see if a file exists, when reading a file

which has an unknown length, or when writing a file with an unknown amount of free space on the disk available.

CREATING YOUR OWN MENUS

Red Ryder 9.0 gives you the opportunity to create your own customized pull down menu that can be used to send the contents of a macro key, execute a Procedure File, or execute a certain Nautilus function (you don't know about that yet). You have complete control of how your menu is named and it's contents. Your menu can contain up to 20 menu choices.

DEFINE MENU name

Function: This gives your menu a name. I highly recommend you do a MENU OFF command before executing this command to be sure any previously defined menu is disposed of before the new one is defined. Note that this just creates the menu in memory - it does not display the menu in the menu bar. Once you've done this command, you'll do some ADD TO MENU and MENU DOES commands to define the contents, appearance, and functions of the menu choices, and then do an ENABLE MENU to display the menu in the menubar and make it choosable by the user. The "name" parameter is the menu's title that will appear in the menu bar when the menu is enabled.

ADD TO MENU string

Function: This adds a menu choice to the menu you've defined. It's added after any existing commands. There is no way to insert a command before others or delete individual commands, so choose the order you use this command with forethought. The "string" item contains the text that will appear in the menu choice with one notable exception. Certain special characters in "string" are used to affect the appearance of the menu choice, so be careful about using non-alphanumeric characters.

You will probably never use hardly any of the following poop (except maybe the dividing line thingy), and some of it is pretty obtuse, but I'll document it here (but probably not in the final docs) so you can play with it and show people all kinds of "betcha didn't know" magic later.

The special characters are ^ ! < / and (and here's what they do:

^char - This will display an icon (which one is according to char) to the left of the menu choice. Unless you want to go digging through Red Ryder to see what menu ICON resources are defined and available, don't fool with this (right....). An interesting thought occurred to

me however, but I haven't tried it. You could actually make your own ICON resources, using resource ID numbers not used by RR for its menu ICONs and use ResEdit to then paste your menu ICON resources into your Procedure file. It would then be available for use with this sort of command. Hmmmmm...

This is hacker city time. What happens is that menu icons are given the reserved ICON resource numbers from 257 to 511. Putting a '1' as "char" in the "string" item means use ICON resource number 257. Putting a '2' means use ICON resource number 258, and so on up the ASCII chart. You can follow an ASCII chart upwards from there to find out what other "char" values would correspond to, should you have the inclination.

!character - Puts "character" to the left of the menu choice. There ain't no way to remove it once it's there, and good luck typing in a checkmark.

<style - The menu item has a special character style. The "style" item is one of the following (shown with the < character):

- <B - boldfaced
- <I - italicized
- <U - underlined
- <O - outlined
- <S - shadowed

I don't know that there's any reason more than one style couldn't be used, but let's keep this thing under control, eh?

/key - keyboard equivalent. Sorry, unless you want to some ResEdit hacking, I don't recommend you try to take advantage of this.

(- item is disabled. There ain't no way to enabled an item, so use this is only one special case. Many Mac menus have a dividing line of disabled hyphen characters. To put one of those in your menu, you would use the special command:

ADD TO MENU (-

As another example, the command:

ADD TO MENU This is a choice<O<U^

That command would put a disabled row of hyphen characters across the width of your menu.

MENUDOESxx number

Function: This command defines exactly what your menu choices do when they are selected. The "xx" part of the command is a number from "01" to "20" (the leading zero _must_ be there for numbers less than 10) that

corresponds to a menu choice (01 being the topmost menu choice and 20 being the bottommost menu choice). The "number" part of the command is a number from 0 to 29 (leading zero not necessary but doesn't hurt) that corresponds to a macro key number to execute when that menu choice is selected. Since there are 3 sets of 10 macros keys each, the "number" parameter works like this:

| number | Macro Key Set/Key# |
|--------|--------------------|
| ----- | ----- |
| 0 | Set 1/Key 0 |
| 1 | Set 1/Key 1 |
| 2 | Set 1/Key 2 |
| . | |
| . | |
| 9 | Set 1/Key 9 |
| 10 | Set 2/Key 0 |
| 11 | Set 2/Key 1 |
| 12 | Set 2/Key 2 |
| . | |
| . | |
| 19 | Set 2/Key 9 |
| 20 | Set 3/Key 0 |
| 21 | Set 3/Key 1 |
| 22 | Set 3/Key 2 |
| . | |
| . | |
| 29 | Set 3/Key 9 |

I'll leave you to fill in the blanks if it doesn't look clear at first glance (in which case the rest of this document must seem a total blur).

When a macro key is executed through a menu choice, and that key is not in the "active" set (the set that is displayed in the Macros Status Bar and can be sent with COMMAND-0 to COMMAND-9), the set containing the selected macro key is made active, and the Macros Status Bar is updated to reflect that if it's being displayed.

INSERT MENU

Function: This command is done after all of the menu items have been added to the menu, it simply draws the menu in the menu bar in an enabled state.

ENABLE MENU number

Function: After any DISABLE MENU command, this command can be used to enable any menu item (if number is from 1 to 20) or the entire menu if number is zero.

DISABLE MENU number

Function: This command can be used to disable (make gray and unselectable) any menu item (if number is from 1 to 20) or the entire menu if number is zero.

MENU OFF

Function: This disables the defined menu, removes it from the menu bar, and wipes it out of memory. If there is no defined menu, this command does nothing. User-defined menus are not removed or disabled when a Procedure terminates - they stay on the menubar until a MENU OFF command is executed. Therefore, it's a real good idea to put a MENU OFF command before any DEFINE MENU commands to wipe out any old user-defined menu.

The Procedure Accelerator

The Procedure Accelerator program serves two purposes:

- 1) To speed up a Procedure by removing all labels and substituting an infinitely faster form of the JUMPTO command.
- 2) To provide a good degree of security against prying eyes. This can be used to protect any information in the Procedure you would not like others to see.

I won't go into much detail about the specifics of how this program works, but will rather concentrate here on how to use it. First, since it does a lot of disk accessing during the first of the two passes, using a RAM or hard disk rather than floppy can greatly speed up its throughput speed (on a floppy, it's about 1 line per second).

Second, DO NOT EDIT THE FILE the Accelerator produces, as it will then most likely not run properly. It will not speed up a file that contains no JUMPTO commands, and is therefore useless on those sorts of files unless you need to take advantage of the "secure" feature.

Third, DO NOT DISPOSE of the "source" file you run through the Accelerator to produce the optimized version. If you make any edits (or take advantage of the "secure" feature), you must make the changes to the original file and then run it through the Accelerator again. There is no way you can decode a "secured" file, and I will refuse to do so for anyone no matter what the excuse (dog ate the original, etc.). I recommend you do your testing with an unoptimized version, and only use this program to produce the "final product".

There are several ways a savvy hacker could find out what a "secured" Procedure sends out with TYPE statements, by switching quickly to HALF duplex, hooking up a terminal and acting like the desired host, etc. There's really only one way to prevent that. The beginning of your Procedure should contain a brief routine like the following:

```
ERASE ~7
ERASE ~9
COPYINTO ~8,Do you wish to produce a Log File for this session (Y/N)?
QUERY1 ~7
CONVUP ~7
CONTAINS FOOBAR
IF YES JUMPTO (START)
ERASE ~7
COPYINTO ~8,Sorry the Log File driver couldn't be loaded.
QUERY3
QUIT
(START)
(This is where your Procedure actually starts)
```

By doing something crafty like this (with your personal password replacing the "FOOBAR" above), the unauthorized user never has the opportunity to see exactly what the Procedure does.

A "secured" Procedure can not be converted back to TEXT, and it turns off and disables the "Monitor Procedure" menu choice (you can override that with a LOUD command in your Procedure, so be careful).

This is not the most elegant program, but it does provide two very necessary functions in an acceptable manner until I can finish the Procedure Editor/Compiler I'm working on.

Since I have not yet publically stated the following, it's time I do so. Any Procedures you write are your intellectual property, and can be sold, given away, copyrighted, etc. without any license or permission necessary from the FreeSoft Company.