Contents

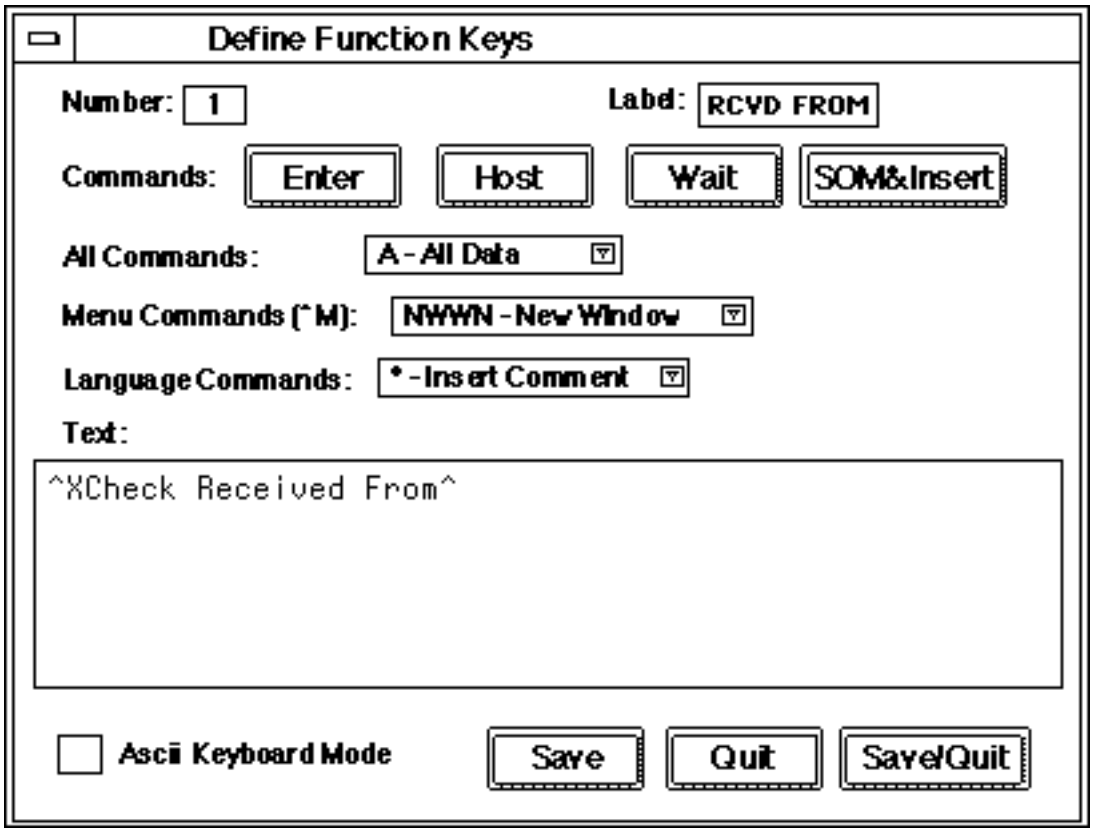
PF Keys

The PF Keys allow a user to automate commands and command sequences that would otherwise have to be entered manually. After a PF key is defined, it may be executed by pressing the appropriate key on the keyboard, by selecting it from a pull-down menu, or by double-clicking on it in the “View PF Keys” window.

PF Keys may be nested up to 12 times. On some systems, there are macro-building utilities that can be used in place of or to augment the Terminal PF Key language. In most cases, both the “macro” keys and PF Keys are accessible. However, any macro script accessed by pressing a function key makes that function key unavailable for normal PF Key usage. For this reason, it is recommended that the function keys are not used to access macros.

Defining PF Keys

PF Keys are created by using the “Define Function Keys...” selection on the Options pull-down menu. When this command is selected, the following window appears:



To define a PF Key, fill in the three fields in the PF Key definition window (the TAB key may be used to move through the fields):

Number - enter the number of the PF Key that is being defined. The valid range of numbers is from 1 to 30. (Note: It is possible create and use multiple sets of PF Keys. Please see the "Terminal Reference Manual" for more details.)

Label - enter a label that will be used to refer to this PF Key. The label may be up to 12 characters long. The label will appear next to this key number on the Key Labels pull down menu and the View PF Keys window.

Text - enter the definition of this PF Key. The text may be up to 255 characters long. (To define a PF Key with more than 255 characters, create it with a text editor and then use the PF key "X" command to run it.) There are buttons for the most commonly used commands, as well as pop-up menus of all commands, menu items, and functions which may be used in PF Key definitions. Rather than typing in a command code, the desired command code can be selected using a button or one of the pop-up menus. To view a pop-up menu, position the pointer on the "All Commands", "Menu Commands", or "Language Commands" box, and then press and hold down the mouse button.

Some users may wish to use "non-standard" keys within function key definitions. (For example, a Japanese agency may want the PF Key to display a prompt on the screen which uses Kanji characters.) Since these keys may not be available on the Terminal's keyboard map, a mechanism has been created to allow the operator to use a different keyboard map while defining PF Keys. The Terminal configuration screen includes a check-box which controls which keyboard map is to be used during PF Key definition. (See "Special Pull Down Menu" section in the Terminal Reference Manual for more detail.)

To define an additional PF Key, click on the "Save" button to save the definition that was just completed, then begin the new key definition by entering a new number in the Number field. When finished defining PF Keys, click on the "Save/Quit" button to exit the PF Key definition function to save the latest definition before exiting, or click on the "Quit" button to simply exit without saving.

PF Key Command Codes

This section lists and describes PF Key commands that may be used in programming PF Keys.

All PF Key commands include the Field Mark (displayed as an "†" or "^ ") followed by a letter or number code which represents a Host command. To type a Field Mark into a PF key definition, press the Control key (ctrl) and the "F" key simultaneously.

Another key that may be necessary as PF Keys are defined is the Option Key (displayed as "~"). Press the Control key and the "O" key simultaneously to create the Option character. On certain hosts, these keys are standard items on the keyboard map.

The following command codes are supported:

nn	Fixed field length
A	Send entire screen to host
C	Position insert point (cursor)
D	Insert optional text
E	Auto enter
F	Execute PF Key
H	Wait for reply from host
I	Search for tab character
K	Erase to end of page
L	Launch another program
O	Input optional text
P	Pause for x seconds
R	Send the next host reply to specified window
S	Move SOM to specified location
T	Display prompting text
V	Fill in variable length field
W	Wait for input
=WF	Wait for input, terminated by a Field Mark
=WE	Wait for input, terminated by the Enter key
=WB	Wait for input, terminated by either Enter or Field Mark
X	Execute commands in a specified file
#	Put a “Start Field” character at the current cursor location
.	Put an “End Field” character at the current cursor location
@	Put the terminal into “protected” mode
–	Take terminal out of “protected” mode
$	Keep Terminal window as active window

Command Code Descriptions

The following examples use formats from a variety of CRS Hosts. The correct format for a particular host may be different than is presented in the example.

nn This command copies a fixed number of characters from the screen, beginning from the SOM position, into the character string generated by pressing a PF Key. For example, if the contents of PF1 are:

↑SR1↑03LAX

and the user enters “DFW” at the SOM and then presses the PF1 key, the screen displays:

1DFWLAX

A This command is used to include all data in the terminal window in any message to the host. For example, if a form was displayed in the terminal window for the user to fill in and PF1 was defined as **↑A↑E**, pressing the PF1 key would send the form to the host.

C This command moves the Insert Point to the specified location in the terminal window. Forms include:

C* position Insert Point at the SOM
Cxx position Insert Point at column 1 of the specified line xx
C-cc position Insert Point at the specified column cc of the current line
Cxx-cc position Insert Point at the specified line xx and column cc

For example:

†C20-45DFWLAX

would position the Insert Point at line 20, column 45 and write

DFWLAX

starting from that point.

Important! Use of the SOM positioning and cursor positioning commands eliminates the requirement that PF keys be pressed only when the cursor is at the SOM. Data to fill in fields is still taken from the old SOM to the cursor when the PF Key was pressed, as it is with any PF Key command. **Some host types do not allow the use of columns with the †C command.** For these hosts, use the †\POSN† command instead.

D This command allows the user to replace data in a PF key command with data the user enters. The data in the PF Key command that may be replaced is delimited with a Field Mark. The end of the replacement data (not limited to same number of characters as are being replaced) must either be delimited by a Field Mark or be at the end of the PF Key data (that is, the PF Key was pressed with the Insert Point at the end of the data). No replacement occurs if no data is entered by the user.

For example: If the contents of PF1 are:

†SR1DFWLAX†D6P

and the user types

4P

and then presses the PF1 key, the following displays in the terminal window:

1DFWLAX4P

If user presses the PF1 key without entering any data, the following displays in the terminal window:

1DFWLAX6P

E This command sends the characters between the SOM and the current Insert Point to the host as if the user had pressed the enter key. For example, if the contents of PF1 are:

ALAXDFW†E

and the user presses the PF1 key with the Insert Point at the current SOM, the availability display command

ALAXDFW

is displayed on the screen and sent to the host.

F This command allows the user to execute a PF Key from inside another PF Key function. The PF Key number is defined as two digits (e.g. as †F03 not †F3). For example, if the contents of PF1 are:

ADFW†F24†E

and the contents of PF24 are:

LIT

and the user presses PF1 at the current SOM, the command

ADFWLIT

is displayed on the screen and sent to the host.

H This command pauses execution of the current PF Key until a response is received from the host. This function normally follows the †E PF Key function. For example, if the contents of PF1 are:

1DFWIAH†E†H1DALHOU†E

and the user presses the PF1 key at the current SOM, the command

1DFWIAH

is displayed on the screen and sent to the host. When the response is received, the second command

1DALHOU

is displayed on the screen and sent to the host. While waiting for the host reply, the message "Holding for host reply - Press reset to stop" appears in the Status Window. Pressing the Reset Key will terminate the PF key.

I This command searches for the next tab character in the terminal window or the next user-defined tab stop in the terminal window and moves the Insert Point to that location.

K This command erases all characters in the terminal window starting from the current Insert Point position and ending with the last character in the terminal window.

L This command causes the specified object to be launched. The “object” may be either: i) a fully qualified pathname leading to an .exe file, or ii) an item from the “launch” section of the iate.ini file. If no name is specified after the L command, then the standard Windows “Open File Dialog” is displayed on the screen so the user can pick an .exe file to launch.

If an .exe file is specified, a new copy is launched even if that program is already running. If an item from the “launch” section of the iate.ini file is specified, it is launched only if the program is not already running; otherwise the copy that is currently running is brought to the foreground.

As an example, if the contents of PF21 are:

†LWordProc†

and “WordProc” is defined in the [launch] section of the iate.ini file as the Microsoft Word® software, then Word is launched when PF21 is pressed. (If Word was already running, it is brought to the foreground.)

O This command optionally replaces a field in a PF Key command with a replacement field entered by the user. The O PF Key command must be followed by a field number, for example, O0, O1, O2, O3, etc. Field numbers

must be in the range 0 - 9. Each optional field in the PF Key definition is terminated by the Field Mark (↑). If the user enters an optional field number and no data, then that field is dropped from the PF text.

Replacement field data the user enters on the terminal screen must be preceded by the "lightning bolt" (~) and the number of the field to be replaced, and ended by a Field Mark.

For example, if the contents of PF1 are:

A↑O1TUL↑↑O2IAH↑↑O36P↑↑E

and the user presses the PF1 key, the command

ATULIAH6P

is sent to the host.

As another example, if the user enters

~2LAX↑

and presses the PF1 key, the command

ATULLAX6P

is sent to the host.

In another example, if the user enters

~3↑

and presses the PF1 key, the command

ATULIAH

is sent to the host.

As a final example, if the user enters

~1SMF↑~39A↑

and presses the PF1 key, the command

ASMFIAH9A

is sent to the host.

P This command pauses execution of the PF Key for the indicated number of seconds. The number of seconds is followed by a Field Mark (↑).

Rnn This command sends the next host reply to window nn. It must be followed by the ↑H command. If a host reply command ↑H is not subsequently given, the R command is ignored. If window nn does not exist, the user is asked whether to cancel the function key or send the output to the current window. For example, if the contents of PF1 are

1LAXORD↑E↑R02↑H

then the host would be sent

1LAXORD

and the reply would be sent to window #2.

S This command moves the SOM to the specified location in the terminal window. Forms include:

↑S* position SOM at the current cursor location (for some hosts, only when SOM's and carriage returns are being displayed - otherwise an

error occurs and key execution stops)

†Sxx position SOM at column 1 of the specified line xx

†Sxx-cc position SOM at the specified line xx and column cc (for some hosts, only when SOM's and carriage returns are being displayed - otherwise an error occurs and key execution stops)

†SR position SOM at the start of the line below the current line (wraps at bottom of window)

For example, if the contents of PF1 are:

†SR†C*1DFWLAX†E

then pressing PF1 with the Insert Point anywhere in a terminal window causes the SOM to be placed at the beginning of the next line; then the Insert Point is placed at the current SOM (see the definition of the "†C" PF Key command) and the command

1DFWLAX

is displayed on the screen and sent to the host.

Important! Use of the SOM positioning and cursor positioning commands eliminates the requirement that PF keys be pressed only when the cursor is at the SOM. Data to fill in fields is still taken from the old SOM to the cursor when the PF Key was pressed, as it is with any PF Key command. **Some host types do not allow the use of columns with the †C command.** For these hosts, use the †\POSN† command instead.

T This command displays, on the second line of the status area, the text that follows the T command. The text must be terminated by either the Field Mark or the end of the key definition. The "Wait for Input" (†W) PF Key command (see below) often follows the T PF Key command. For example, if the contents of PF1 are:

A†TENTER DESTINATION†SFO†W†E

If the user presses the PF1 key,

ASFO

is displayed on the screen, and

ENTER DESTINATION

is displayed on the second line of the status bar. If the user then types:

DFW†

then

ASFODFW

is sent to the host.

V This command optionally fills in fields in a PF Key command with fields entered by the user. Multiple fields may be replaced. If the user enters multiple fields, they should be separated by Field Marks. For example, if the contents of PF1 are:

1†V†V†E

If the user types:

DFW†LAX

and then presses the PF1 key, the command

1DFWLAX

is sent to the host.

W This command stops further processing of a PF Key command until the user enters data. The user must terminate the input with a Field Mark. The message "Press Reset or end input with Field Mark" appears in the Status Window, and the pointer changes to a hand until the user enters the data. Pressing the Reset key cancels the PF Key execution. For example, if the contents of PF1 are:

1DFW↑W↑E

and the user presses the PF1 key, the command:

1DFW

is displayed.

The user must then enter data, terminated by a Field Mark, before the PF Key will complete. If the user types:

LAX↑

1DFWLAX

is sent to the Host.

=WF↑ This command is identical to the **↑W** command.

=WE↑ This command is identical to the **↑W** command except that the user terminates input with the Enter key instead of the Field Mark.

=WF↑ This command is identical to the **↑W** command except that the user terminates input with either the Enter key or the Field Mark instead of only the Field Mark.

X This command executes a PF Key script that is stored in a text file. The file name immediately follows the X command, and is terminated by a Field Mark or the end of the PF Key definition. Lines in the file are executed just as if they were part of a PF Key definition. Within the file, the Field Mark should be entered as the "^" character (shift-6). All letters that are part of a message to the host should be entered as upper case letters. If no name is specified after the X command, then the standard "Open File Dialog" is displayed on the screen so the user can pick a file to run.

The first characters of the first line in the file should be the SOM/Cursor Reset command (↑SR↑C*). The first characters of each subsequent line in the file should be the Cursor Reset command (↑C*). Each host command (such as "1MSPORD") should be followed by the "enter" and "wait for host response" commands (↑E↑H). The message, "Holding for reply - Press Reset (esc) to end", appears in the Status Window while the X command is executing. Pressing the Reset key cancels the PF Key execution. For example, if the contents of PF1 are:

↑XBOSTON.PF

and the contents of the file "BOSTON.PF" are:

^SR^C*1MSPBOS^E^H
^C*1BOSATL^E^H
^C*1ATLMSP^E

and the PF1 key is pressed, the command:

1MSPBOS

is sent to the host and the response is displayed on the screen; the command:

1BOSATL

is sent to the host and the response is displayed on the screen; and the command:

1ATLMSP

is sent to the host and the response is displayed on the screen.

The comma character (“,”) may not be used in filenames that are referred to by the ^X command. (The “,” character is used to pass arguments to a PF Key file while executing it.)

This command places a “Start of Field” character in the terminal window at the current cursor location. This function is used to build formatted screens that allow a user to tab to certain locations on the screen. It is also used to mark the beginning of a field that is to be used when the terminal window is placed in protected mode.

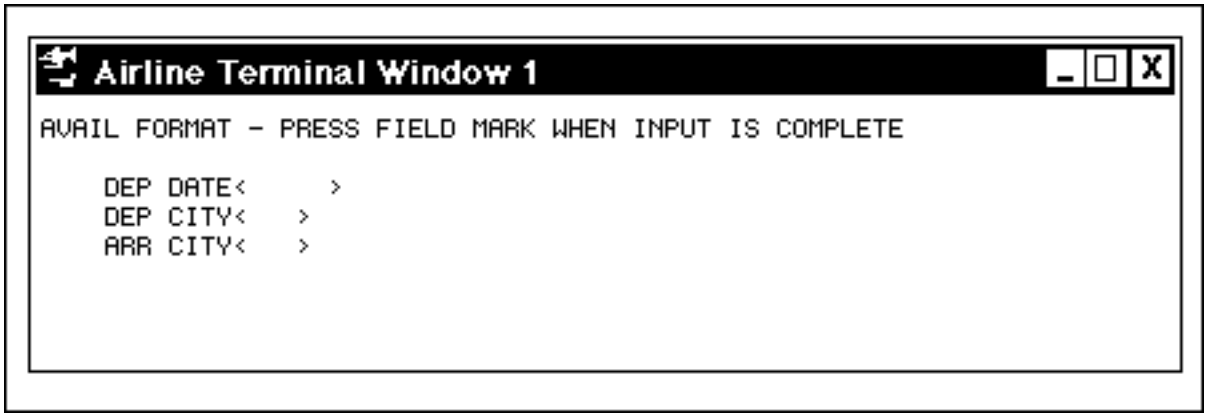
. This command places an “End of Field” character in the terminal window at the current cursor location. It is also used to mark the end of a field that is to be used when the terminal window is placed in protected mode.

@ This command forces the terminal to go into “protected mode”. Protected mode is used primarily to build masks on the screen. When in protected mode, the only place a user may type in the terminal window is between fields that start with a “Start of Field” character and end with an “End of Field” character. (For SABRE users in protected mode, when ENTER is pressed only certain data from the terminal window is sent to the host. This data includes the first two characters on the screen, plus all data found in fields that start with a “Start of Field” character and end with an “End of Field” character.)

- This command forces the terminal to go into “unprotected mode”. This is the normal mode of operation for the terminal and is the default unless the “@” function has been invoked. When in unprotected mode, the user may type anywhere in the terminal window. In unprotected mode, when ENTER is pressed all of the characters between the SOM and the current location of the cursor on the screen are sent to the host.

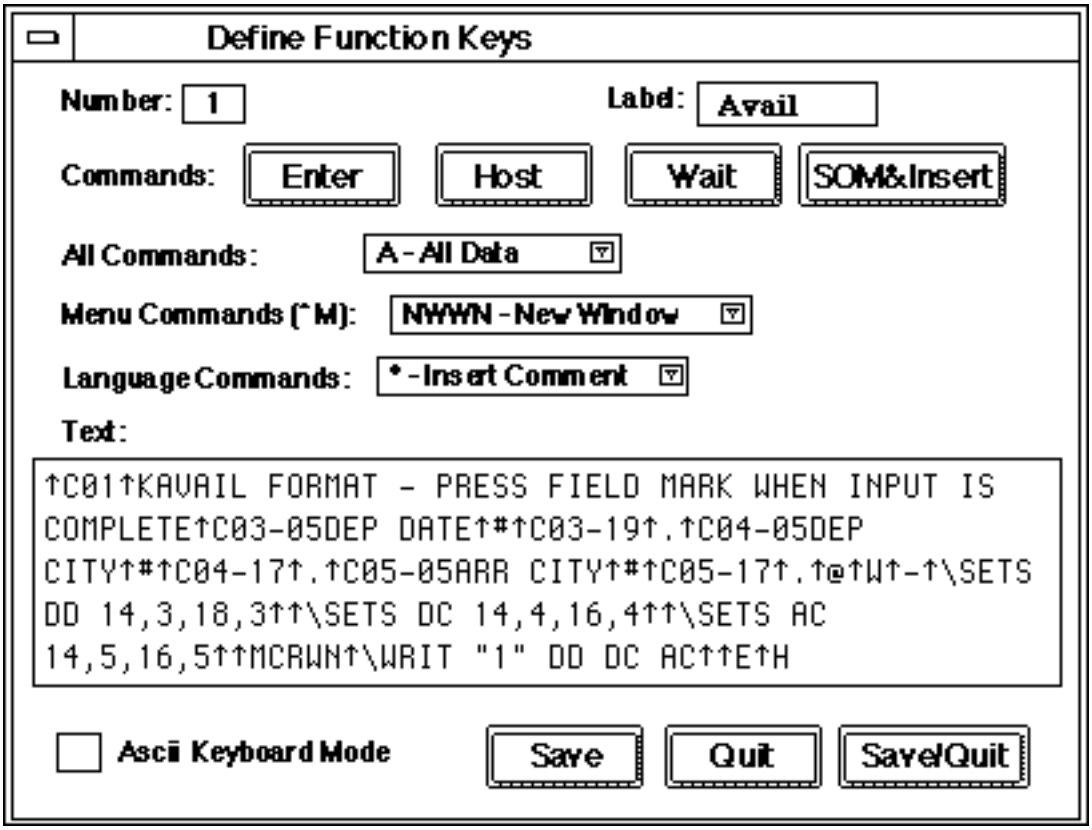
$ This command forces the terminal to keep the current terminal window in the foreground when a PF key completes. This function is intended for users who expect to execute a key from the View PF keys box, but want the terminal window to remain the active window when the key finishes instead switching back to the View PF keys window.

Example: The screen mask below is generated by the PF Key definition that follows:



Availability screen mask built with the sample PF Key script

The PF Key definition that sets up this mask is:



PF Key definition window with example PF key

The following is an explanation of each command in the PF Key script - some of the functions/commands used in this example are further explained in the section of the manual that follows, “PF Key Structured Programming Language”. (Note: This key may require

minor revisions in order to work with certain hosts):

↑**C01**: Place the cursor at the beginning of line 1 in the terminal window.
↑**K**: Clear the screen from the current cursor location to the end of the screen.
AVAIL FORMAT - PRESS...(etc.): Places this text at the current cursor position.
↑**C03-05**: Place the cursor at row 3 column 5.
DEP DATE: Places this text in the terminal window at the current cursor position.
↑**#**: Place a "Start of Field" character in the terminal window (at the current cursor position).
↑**C03-19**: Place the cursor at line 3, column 19 in the terminal window.
↑**.**: Place an "End of Field" character in the terminal window (at the current cursor position). (When the screen is placed in protected mode, this "End of Field" character and its preceding "Start of Field" character will define a field where the user enters the departure date.)
↑**C04-05**: Place the cursor at row 4 column 5.
DEP CITY: Places this text in the terminal window at the current cursor position.
↑**#**: Place a "Start of Field" character in the terminal window (at the current cursor position).
↑**C04-17**: Place the cursor at line 4, column 17 in the terminal window.
↑**.**: Place an "End of Field" character in the terminal window (at the current cursor position). (When the screen is placed in protected mode, this "End of Field" character and its preceding "Start of Field" character will define a field where the user enters the departure city.)
↑**C05-05**: Place the cursor at row 5 column 5.
ARR CITY: Places this text in the terminal window at the current cursor position.
↑**#**: Place a "Start of Field" character in the terminal window (at the current cursor position).
↑**C05-17**: Place the cursor at line 5, column 17 in the terminal window.
↑**.**: Place an "End of Field" character in the terminal window (at the current cursor position). (When the screen is placed in protected mode, this "End of Field" character and its preceding "Start of Field" character will define a field where the user enters the arrival city.)
↑**@**: Put the terminal in protected mode.
↑**W**: Wait for user to enter data. In this PF key, we are not actually soliciting input with this command. Instead, we are using this command to prevent the execution of any more PF key commands until the user presses the field mark key (See ↑W description).
↑**-**: Exit protected mode since the user has completed entering data.
↑**\SETS DD 14,3,18,3↑:** Sets a string variable called DD with the characters starting at column 14 of row 3, and ending with the character in column 18 of row 3.
↑**\SETS DC 14,4,16,4↑:** Sets a string variable called DC with the characters starting at column 14 of row 4, and ending with the character in column 16 of row 4.
↑**\SETS AC 14,5,16,5↑:** Sets a string variable called AC with the characters starting at column 14 of row 5, and ending with the character in column 16 of row 5.
↑**\MCRWN**: Clears the window and places the cursor at the home position (row 1, column 1).
↑**\WRIT "1" DD DC AC↑**: Places the text "1" (a host "availability" command code) in the terminal window at the current cursor position. It is followed by the text in the strings DD, DC, and AC.
↑**E**: Send the data between the SOM and the current cursor position to the host.
↑**H**: Once a response is received from the host, the script has finished running.

(Consequently, if the user types 22APR in the first field, LAX in the second field, and JFK in the third field, the command sent to the host would be "**122APRLAXJFK**".)

Important Note - SABRE users must either 1) select the "SOM and Returns shown" box in their terminal configuration screen to successful run any PF key, including this example key, that positions the cursor at a column other than column 1; or 2) change the ↑C commands to ↑\POSN↑ commands.

PF Key Menu Item Codes

This section lists and describes the menu item codes used with PF Key command "↑M" to build PF Key definitions. The menu item codes include:

APCT	Append Cut
APCY	Append Copy
ARTC	Add Returns to Copy
CLER	Clear
CLWN	Close Window
CMCP	Column Copy
COPY	Copy
CPDL	Copy Delimited
CRWN	Clear Window
GOWN	Go to Window
NWWN	New Window
NXWN	Next Window
PSTE	Paste
PTWN	Print Window
PVWN	Previous Window
SCUT	Cut
SLCT	Select

Brief definitions of these commands follow. For a further description, refer to the descriptions of the pull down menus in the "Using the Terminal" section of this manual.

APCT This command cuts the already selected area from the active window and appends it to the contents of the clipboard.

APCY This command copies the already selected area from the active window and appends it to the contents of the clipboard.

ARTC This command toggles the terminal in/out of Add Return to Copies mode.

CLER This command clears an already selected area.

CLWN This command closes the then active Terminal window.

CMCP This command toggles the terminal in/out of Column Copy mode.

COPY This command copies the already selected area from the active window to the clipboard.

CPDL This command toggles the terminal in/out of Copy Delimited mode.

CRWN This command clears the then active window.

GOWN This command makes the specified Terminal window the active window. The window number is specified as two digits (e.g. GOWN02 makes window 2 the

active window.)

NWWN This command opens a new Terminal window and makes it the active window.

NXWN This command makes the next (numerically sequential) Terminal window into the active window.

PSTE This command takes the contents of the clipboard and inserts it into the active window.

PTWN This command prints the contents displayed in the active Terminal window.

PVWN This command makes the previous (numerically sequential) Terminal window into the active window.

SCUT This command cuts the already selected area from the active window and places it in the clipboard.

SLCT This command selects a specified area. The area is defined by providing column,row,column,row coordinates. For example, the command "↑MSLCT1,2,3,4↑" will select an area beginning at column 1 - row 2 and ending at (just before) column 3 - row 4.

ZOOM This command is a toggle which functions the same as the "Zoom Box" in the upper right-hand corner of the terminal window.

PF Key Structured Programming Language

The *Win IATE* Terminal software PF keys include a complete structured programming language. This language allows the operator to send one or more entries to the host and use the responses from the host in a script. An example program appears in Appendix A. The language includes:

- Variables
  - Integer
  - String
  - System
- Control Structures
  - If/else/end
  - While/end
- Expressions & Functions
  - Arithmetic (integer only)
  - Boolean (i.e. evaluate to either “zero” or “not zero”)
  - String
- Assignment statements
  - Arithmetic functions
  - Moving data to and from the screen
  - String manipulation

Following are the rules about Assignment Statements and Expressions:

- String variables may be assigned a value from a string literal from any location on the terminal screen, or from another string or integer variable. A string literal is any series of characters contained within double quotes "". Note that some word processors provide both the regular double quote (") and the “smart” double quotes (“ and ”). Smart double quotes are not recognized by the PF key language. The user must turn off smart double quotes in the word processor, or use a simple text editor such as NotePad to build PF keys that include string literals.
- The maximum length allowed for string variables is the maximum number of characters that may be displayed on the terminal screen.
- The maximum length allowed for string literals is 64 characters.
- Integer variables may be assigned a value from a literal from any location on the terminal screen, or from another integer or string variable.
- Assigning an Integer Variable to a String Variable creates an Ascii (“string” or “text”) representation of the integer.
- Assigning a String Variable to an Integer Variable converts the contents of the String Variable to an integer. 0 is the result of a failed conversion.
- Expressions are evaluated from left to right. Expressions within parentheses are evaluated first. No precedence (i.e. priority of processing/evaluation) is given to the Arithmetic or Boolean operators.
- Boolean expressions must be parenthesized.
- Only the Boolean expressions for Equality and Inequality are allowed for strings and the CMPS operator must be used. Unpredictable results may occur if strings of different lengths are compared. “=”, “>”, “<“, etc. may not be used with string variables.
- The CMPS operator may only be used in Boolean expressions. Unpredictable results may occur if a string from the terminal window that was displayed on more than one line in the terminal window is used with the CMPS operator.

• If a string variable is encountered in an arithmetic expression, an attempt is made to convert the first digit(s) encountered in the string to a number and use that number to finish evaluating the expression. If the string cannot be converted to a number, then a 0 is used instead. No error is generated.

The Arithmetic operators provided are:

+	addition
-	subtraction and unary negation
*	multiplication
/	division
%	modulus

The Boolean operators provided are:

()	encloses a Boolean expression.
=	equality
<>	inequality
>	greater than
<	less than
>=	greater than or equal
<=	less than or equal
&&	Boolean and
||	Boolean or
!	Boolean not

CMPS varNameOrLiteral varNameOrLiteral - compares two strings

A function called “SLEN” is provided to return the length of a String Variable. SLEN may appear anywhere that a numeric or boolean expression may be used. SLEN may not be part of an expression or operate on an expression. An example of the use of SLEN is given below in the “Sample Program Segments” part of the manual.

Following are the rules about variables:

• Variables may be either Integer variables or String variables.
• Variables do not have to be declared before they are used; however, results of using variables that have not been initialized are unpredictable.
• Variable names must begin with a letter and may be alpha numeric. The name may not exceed 64 characters in length.
• Variable names are case-sensitive. (This means "Flight" and "flight" are two different variables.)
• Variable names may not be the same as any keyword name. The four System Global Variables also may not be used as variable names.
• Integer variables may range from 2,147,483,647 to -2,147,483,647.
• Variable names may be used in expressions.
• Every variable is local to the pf key in which it is first used (declared) unless the variable is explicitly defined to be “Global”. That is, if PF key 1 calls PF key 3, the variables used in PF key 1 are not usable in PF key 3 unless they are used as arguments to PF key 3 or are ‘globalized’.

The PF key programming language functions are invoked by using the name of the function (“keyword”) desired. The names of all keywords must be preceded by “^\” and be followed by “^”. The following section describes the function of each of the keywords and gives examples of the use of the keywords:

^**BEEP**^ Causes the system beep to sound.

^\BEEP^

^**CATS**^ Creates a new string variable (or replaces the value of an existing string variable) by concatenating string variables and string literals. The first operand following the CATS keyword must be a the name of a string variable. The values of all of the string variables and literals following the first operand are concatenated and put into the first operand. The string literals may not be longer than 64 characters.

^\SETS strnew "new"^

^\CATS str "a " strnew "string"^ The string variable str now contains “a new string”

^\CTOI^ Converts a string variable to an integer variable or truncates the rightmost characters of a string. The first operand is a string variable. The second operand may be a string or an integer variable. Only the leftmost character of the first operand is used. If the second operand is an integer, the character’s ascii value is the new value. If the second operand is a string, the new value is a string composed of the single character. Any string expression which is legal for SETS is also legal for CTOI, including a position on the current terminal screen. THE FIRST OPERAND MUST HAVE BEEN PREVIOUSLY USED IN A SETI OR SETS STATEMENT.

^\CTOI intVarName “A”^ -> intVarName = 65

^\CTOI strVarName “A”^ -> strVarName = “A”

^**DARG**^ Declares a series of argument names when calling a numbered pf key or pf file. Once declared, whenever a ^F or ^X for the function is encountered, any arguments following the number or file name are turned into variables which may be accessed and changed during the run of the function. Once the function ends, the argument variables no longer exist. A function key’s arguments need only be declared once during a run of the program.

^\DARGfunctionKeyOrFileName, arg1,... argN^

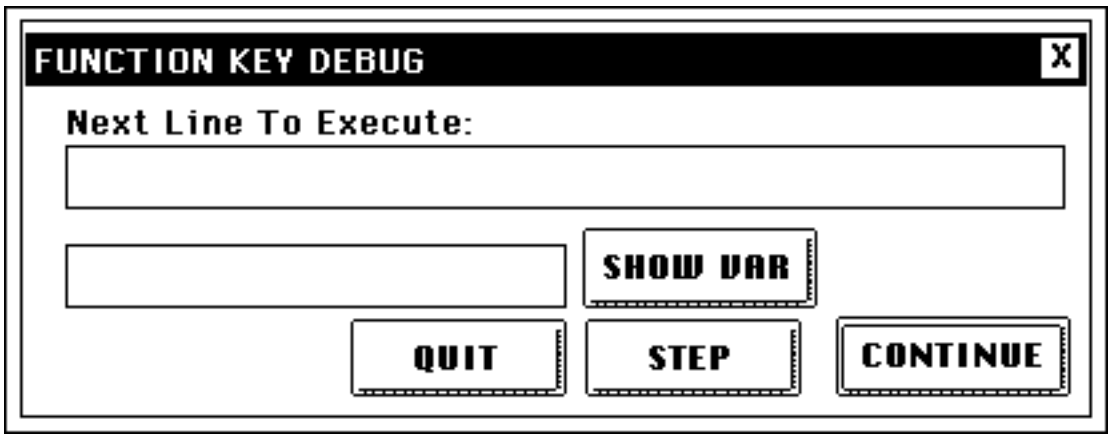
For example, if the user wants to pass a string into function key 1, they would first declare it:

^\DARG1, myStringVar^

Now, when calling F1, the user may pass any variable or literal string and it can be accessed through myStringVar. The syntax for calling is ^F1, “this will be the contents of myStringVar”^. Once F1 begins to run, myStringVar will contain “this will be the contents of myStringVar”. Commas must be used to separate arguments (spaces are not significant). All types of expressions will be evaluated, however, numerical or Boolean expressions must be enclosed in parentheses for proper evaluation. The type of the variable is determined by the type of the passed value.

^**DBUG**^ Turns on the PF key debugging function. Once a DBUG command is executed, the PF key processor will display the PF key debugging window every time a PF Key function is encountered. The PF key debugging window gives the operator the option to display any variable; to step to the next “^\” command and then pause again; to continue executing the PF key without any more debugging pauses; or to quit the PF key.

^\DBUG^



Example of Debugging Window

Type the name of any variable (the variable name is case sensitive) in the box to the left of the "Show Var" button and then click on the button to display the value contained by that variable. The "Step" button is used to move through the script, function by function. The "Continue" button causes the Debugging Window to close and the script to finish running. The "Stop" button closes the Debugging Window and the script stops running.

^**ELSE**^ May follow an IF statement. PF key text after the ELSE and before and ENDI is executed as a normal PF key text when the expression after the IF evaluates to zero.

^\IF (expression = 0)^ ...pfKeyTextThatIsNotExecuted... ^\ELSE^
...pfKeyTextThatIsExecuted... ^\ENDI^

^**ENDI**^ Marks the end of text to execute from a preceding IF or IF/ELSE.

^\IF (expression not = 0)^ ...pfKeyTextThatIsExecuted... ^\ENDI^

^**ENDW**^ Marks the end of text to execute from a preceding WHIL.

^\WHIL (expression)^ ...pfKeyTextThatIsExecuted..^\ENDW^

^**EXIT**^ Causes the PF key script to stop executing and returns the user to the normal terminal mode.

^\EXIT^

^\FCLS^ Closes a file. The only operand is the name of the file to be closed (including the path name if the file is not in the current directory). The file name can be expressed as any legal string expression.

^\SETS MyFileName "NewProfiles"^
^\FCLS MyFileName^

^\FDEL^ Deletes a file. The only operand is the name of the file to be deleted (including the path name if the file is not in the current folder). The file may not be open The file name can be expressed as any legal string expression.

^\SETS MyFileName "Profiles"^
^\FDEL MyFileName^

^\FOPN^ Opens a file. The only operand is the name of the file to be opened (including the path name if the file is not in the current directory). If the file is found, the

file is opened. If the file is not found, a new file is opened for output. The file name can be expressed as any legal string expression. Up to ten files may be open at the same time by a script.

^\SETS MyFileName "NewProfiles"^
^\FOPN MyFileName^

^\FRED^ Reads from a file. The first operand following the FRED keyword is the name of the file to be read (including the path name if the file is not in the current directory). FRED requires two additional arguments, (1) a string variable name (does not have to be created yet) to read the text into, and (2) an integer expression which determines how much text to read. The read happens from wherever the last read stopped since opening the file. If the integer expression is 0, then the next line is read from the file, including the end of line character. Continuing with 0 length reads will read a file line by line. A single FRED command will read up to the next 255 characters. Lines longer than this will be broken into 255 character segments. When the string variable is empty (CMPS myStringVar ""), then the complete contents of the file have been read (at end of file). FRED commands with the read length expression set to 0 and FRED commands calls with the read length expression set to a specific length cannot be mixed. If this is attempted, the returned text is undefined and the PF key may encounter system file errors.

^\SETS MyFileName "NewProfiles"^
^\FRED“MyFileName MyStringVariable 0^

^\FWRT^ Writes to a file. The first operand following the FWRT keyword is the name of the file to be opened (including the path name if the file is not in the current directory). The second operand is a string variable name which contains the text to be written to the file. A carriage return is added to the end of the data written to the file to indicate the end of this record. The string variable is the same as the string variable for a SETS: a variable name, a literal or a position on the current terminal window. The first time a write is performed, any contents of the file are deleted.

^\SETS MyFileName "NewProfiles"^
^\FWRT MyFileName MyStringVariableOrLiteralOrPosition^

^**\GLOB**^ Establishes the listed variable(s) as global. This means that the variable retains its value until the terminal program quits. Global variables are available to any PF key, once defined. A variable that the user wishes to be “Global” variables must **first** be defined with a ^\SETI^ (or ^\SETS^) statement, **followed** by a ^\GLOB^ statement to give the variable its global scope. One variable name is allowed for each ^\GLOB^ statement.

^\GLOB varName1[, varName2, .. varNameN]^

^**\GPOS**^ Places into the two specified integer variables the number of the column and the number of the row where the cursor is currently positioned. If the variables have not previously been declared, they are created.

^\GPOS Int1,Int2^ The integer variable Int1 now contains the current cursor column. The integer variable Int2 now contains the current cursor row.

^**\IF**^ Begins an “If” expression. It must be followed by parentheses enclosing an expression and it must be closed by an ENDI keyword. Text between IF and ENDI may be empty. Text in between IF and ENDI is executed as normal PF key text when the expression following the IF evaluates to non-zero.

^\IF (expression not = 0)^ ...pfKeyTextThatIsExecuted... ^\ENDI^

^\ITOC^ Converts an integer variable to a character or does a modulus 256 function, depending on whether the first operand is a string variable or an integer variable. The second operand can be any legal integer expression. If the first operand is an integer variable, ITOC returns in the first operand the modulus 256 of the second operand. If the first operand is a string variable, ITOC returns in the first operand the Ascii character that is the modulus 256 of the second operand. THE FIRST OPERAND MUST HAVE BEEN PREVIOUSLY USED IN A SETI OR SETS STATEMENT.

^\ITOC strVarName 64+1^ -> strVarName = “A”

^\ITOC intVarName 64+1^ -> intVarName = 65

^**POSN**^ Positions the cursor to the column and row specified by the two integer variables Int1 and Int2 or Expressions Expr1 and Expr2. If either Int1 or Int2 specifies a location less than one or is greater than the maximum configured values for the terminal window, the cursor is not moved..

^\POSN Int1 Int2^ positions the cursor to the column and row specified by Int1 and Int2.

^**RSLT**^ Returns a result from a pf key file or pf key that is called from another. It sets a global variable called “THERESULT” which can then be queried or used in an expression. THERESULT changes type to accommodate whatever type is being returned.

^\RSLT (integerExpressionOrStringVariableOrLiteral)^

^**SETI**^ Defines or sets an integer variable. It must be followed by a variable name. The first SETI creates a new integer variable. Subsequent SETI commands change the contents of the variable. Capitalization is significant, so “Var” and “var” are two different variables.

^\SETI varName 1 + 3 * (3 - 1)^ sets varName to 8.

^\SETI varName 10,3,15,3^ sets varName to the value of the characters on the screen between column 10, line 3 and column 15, line 3. If this string does not contain a number, varName equals 0.

^\SETI varName1 varName2^ sets the contents referred to by varName1 to be a copy of the contents of varName2.

The script can also use a variable name for a location, as in:

^\SETI varName 10,lineVar,15,lineVar^. If lineVar contained 10, this would put the value of the characters on the screen between 10,10 and 15,10 into varName.

^**SETS**^ Defines or sets a string variable. It must be followed by a variable name. The first SETS creates a new named string variable. Subsequent SETS commands change the contents of the variable. Capitalization is significant, so “Var” and “var” are two different variables.

^\SETS varName "TheString"^ sets varName to TheString.

^\SETS varName 10,3,15,3^ sets varName to the string on the current terminal screen between column 10, line 3 and column 15, line 3 inclusive.

^\SETS varName1 varName2^ sets the contents referred to by varName1 to be a copy of the contents of varName2.

The script can also use a variable name for a location, as in:
^\SETS varName 10,lineVar,15,lineVar^. If lineVar contained 10, this would put the screen contents between 10,10 and 15,10 into varName.

^**TRIM**^ Trims a string variable of leading and trailing blanks; it also trims a string variable to a particular substring. The results are stored back into the same string variable.
^\SETS str " a short string "^
^\TRIM str^ The string variable "str" now contains "a short string"
^\TRIM str 3,7^ The string variable "str" now contains "short"

^**WHIL**^ Begins a "While" expression. It must be followed by parentheses enclosing an expression and closed by an ENDW keyword. Text is not required between the WHIL and ENDW keywords. Text between WHIL and ENDW is executed as normal PF key text until the expression following the WHIL evaluates to zero.
^\WHIL (expression)^ ...pfKeyTextThatIsExecuted.. ^\ENDW^

^**WRIT**^ Places the values of the listed variables onto the terminal screen at the current cursor location. String literals may be interspersed in the list of variables and they are written in the stream. However, the string literals may not be longer than 64 characters. Using the "RETURN" constant in a ^\WRIT statement will clear the line to the right of the Return character.
^\WRIT varNameOrLiteral [varNameOrLiteral...]^

^*****^ Defines a comment. None of the text between the "^*" and the following "^" is executed. Because it is the delimiter that signals the end of the comment, the Field Mark key ("^" or "^") may not be used in a comment except as the last character.
^* This text is a comment and is not executed ^

System Global Variables:

COLUMNS contains the number of columns the terminal window is configured for.

HOSTAVAILABLE is True (1) when the "Sys Avail" indicator is displayed in the Status Line, False (0) otherwise. It is most commonly used in startup scripts to prevent the terminal from hanging because a command has been sent to the host before the connection is completely up. Sample code to perform this test is:
^\WHIL (HOSTAVAILABLE=0)^^P1^^\ENDW^

HOSTWAITTIME contains the number of seconds the terminal should wait for a host reply before returning control to a script after the ^H command. The default value is 0, which means "wait forever".

HOSTWAITTIMEDOUT contains a 0 when the the response from the host was returned in a time less than or equal to the value specified in **HOSTWAITTIME** for the most recent ^H command. Otherwise a 1 is returned.

LASTHOSTMSG contains the last complete host message. Any attribute characters from the host are removed, although they may take a place on the screen. If the host has sent more characters than the buffer allows, then only the last portion of message is displayed. There are no carriage returns in this variable.

LASTHOSTMSGLEN contains the number of characters in the last host message.

LASTHOSTMSGLINES contains the number of lines in the last host message after it was written on the terminal screen. (Some of the lines may end in a carriage return and some may not. This does not refer to the System Global Variable LASTHOSTMSG, which does not contain lines). Note that if the host sends back a one-line error message and then a SOM on the next line, LASTHOSTMSGLINES will contain 2.

RETURN contains the return character. It is most commonly used in the "^\WRIT .. ^" statement when it is desirable to display text on the next line down.

ROWS contains the number of rows the terminal window is configured for.

THERESULT contains the result returned from a pf key or file that was called from another pf key or file.

TOTALTERMINALWINDOWS contains the number of terminal windows currently configured.

WINDOWNUMBER contains the number of the terminal window that is "in front".

Sample Program Segments

• SETI Keyword:

```
^\SETI line 3^^\SETI line line+1^^\WRIT line " "^
```

• Loop using WHILE. Before running this example, you need a terminal with at least windows 1 and 2 open. In terminal window 2, columns 4-9 of line 1 should be "AAAAAA", columns 4-9 of line 2 should be "BBBBBB", and so on through line 9 and "IIIIII":

```
^\SETI line 1^^\WHIL line<10^^\SETS str 4,line,9,line^^MGOWN01^\WRIT "line "
line " "  str^^MGOWN02^\SETI line line+1^^\ENDW^^\BEEP^
```

• TRIM Keyword:

```
^\SETS str "   Has Leading and Trailing spaces     "^^\WRIT str RETURN^^\TRIM
str^^\WRIT str RETURN^^\TRIM str 5,31^^\WRIT "Does not have " str^
```

• CATS Keyword:

```
^\SETS str "concatenated"^^\CATS str "a  " str "  string"^^\WRIT str^
```

Result: The variable str now contains "a concatenated string".

• System variables:

```
^MGOWN01^\WRIT "msg lines " LASTHOSTMSGLINES " len "
LASTHOSTMSGLEN " contents " LASTHOSTMSG^
```

• SLEN function:

```
^\SETS STRa "FIVE SIX"^
^\SETI STRlen SLEN STRa^
^MCRWN
^\WRIT "String Length is " STRlen^
```

Result: “String Length is 8” is displayed in the terminal window.

• Example of looping through the alphabet using ITOC:

```
^\SETS currentLetter ""^
^\SETI loopctr 1^
^\WHIL (loopctr <= 26)^
  ^\ITOC currentLetter loopctr+64^
  ^\WRIT currentLetter RETURN^
  ^\SETI loopctr loopctr +1^
^\ENDW^
```

Result: The alphabet will display in the terminal window, one letter per line.

• Example of the use of a global variable:

```
^\* Script 1 of 2 to demonstrate the use of Global Variables^
^\SETI VARA 32251^
^\GLOB VARA^
^MCRWN^\WRIT "Original value of the variable is: " VARA RETURN^
^XSCRIPT2^
^\WRIT RETURN "New value of the Global Variable is: " VARA^

^\* Script 2 of 2 to demonstrate the use of Global Variables^
^\* Note that the Global Variable is NOT declared global (“GLOB”) in this script!! ^
^\WRIT RETURN "Into the second script using global variable: " VARA RETURN^
^\SETI VARA 32296^
```

• File I/O commands, example 1:

```
^\* open a file, read the first line, replace the contents of the file with the first line read,
then close the file^

^\FOPN MyTextFile^
^\FRED MyTextFile myStringVariable 0^
^\FWRT MyTextFile myStringVariableOrLiteralOrPosition^
^\FCLS MyTextFile^
```

• File I/O commands, example 2:

```
^\* open a file, read every line and close the file ^

^\FOPN MyFile^
```

```
^\FRED MyFile fileVar 0^
^\WHIL !(CMPS fileVar "")^
  ^\WRIT fileVar^
  ^\* do something with the data in the file^
  ^\FRED MyFile fileVar 0^
^\ENDW^
^\FCLS MyFile^
```

- File I/O commands, example 3:

```
^\* open 2 files, and copy one to the other in 1000-character blocks^

^\FOPN MyFirstFile^
^\FOPN MySecondFile^
^\FRED MyFirstFile fileVar 1000^
^\WHIL !(CMPS fileVar "")^
  ^\FWRT MySecondFile fileVar^
  ^\FRED MyFirstFile fileVar 1000^
  ^\ENDW^
^\FCLS MyFirstFile^
^\FCLS MySecondFile^
```

Appendix A contains a sample program which illustrates the use of many of the PF key functions.

Programming Notes

When PF key text is entered into a file, the Field Mark character (“^”) is entered as the “^” character (Shift 6).

Functions cannot be nested. For example, ^\IF (CMPS ^\TRIM STR1^ "AA")...^ is not allowed because the TRIM function is nested inside the CMPS function.

If a variable is needed that tells script whether this is the first time the script has run since the terminal was launched, initialize a variable and declare it global in a Start-up script for the terminal emulator.

If a PF key is not running properly and it is difficult to figure out why, or where the error is occurring, insert a ^\DBUG^ statement or some ^\WRIT^ statements into the PF key. This will help in figuring out where problems are occurring in the PF key and what the values of the variables are.

A very common error in ^\IF^ statements is to try to compare one string to another without using the CMPS function. The normal Boolean operators such as “=”, “>”, “<“, etc. do not work with string variables.

If a form of SETI (SETS) is used that creates a variable from data in the terminal window, the starting and ending row and/or the starting and ending column must be different. For example, to access the first character on the screen, use “^\SETI varName 1,1,2,1^”, not “ ^\SETI varName 1,1,1,1^” or “^\SETS varName 1,1,2,1^”, not “ ^\SETS varName 1,1,1,1^”.

Be very careful about the use of spaces in PF keys. Any character (including the space character) which is not part of a PF key command is interpreted to be something that should be displayed in the terminal window.

Do not use any of the system global variable names as variable names or literals.

If there is the choice of using either an IF statement or a WHILE statement, use a WHILE statement if speed is the primary consideration.

Be very careful in the use of the space character (“ ”) and the quote (") character. They are used in different commands and will cause syntax errors if used incorrectly.

Special notes regarding carriage returns:

1) Carriage returns are ignored in a PF key unless they are in a string literal. For ^\WRIT^ or ^\SETI^ statements, if a carriage return is placed between double quotes like any other literal, it is written to the screen properly. This is the only way to get a return on the screen. To send a carriage return to the host, use the end item symbol (∑). This will not force a new line on the screen, but will be translated to a carriage return when transmitted to the host. All keyboards have ∑ on them. Usually this character is where the vertical bar character (shift \) is. On some host types it is in other places.

2) Carriage returns at the end of a line in PF key scripts are ignored.

Appendix A: PF Key Structured Program Example

The following PF Key script illustrates the structured programming language embedded in the PF key script. It is an example of a script to display all the United flights in a PNR. The PNR Record Locator, followed by a Field Mark, must be entered when the script prompts for it.

To run this script,

1) Copy the following script as text into a text file of any name,

2) Build a PF Key in the terminal that executes the script from a file (using the “^Xsss^” PF Key function, where nnn is either null or the name given to the file in which the PF Key script was saved), and

3) Run the PF Key. If a name was not entered for sss, when the PF Key runs it will prompt for a file name. When it does, select the file that contains the PF Key script. It may be necessary to make a few changes to this script to make it run properly, because this script was written for an airline host that i) displays the air itinerary at the top of the screen and ii) sends back one more line than the number of flights in the itinerary.

```
^C01^K^C02-25Display Flight Numbers
^\* Position the cursor at home, clear the screen, and display the screen title^
^C04-12PNR Record Locator ^#
^\* Put the prompting message on the screen and start a field using the “#” function ^
^C04-38^ . ^@
^\* Put the terminal in protected mode, position the cursor and end the field^
^W
^\* Make the PF key pause until the user presses field mark^
^-
^\* Take the terminal out of protected mode^
^\SETS PNR 32,4,37,4^
^\* Put the data the user entered into the field called PNR^
^\TRIM PNR^
^\* Strip off any leading or trailing spaces from PNR^
^\IF (CMPS PNR "")^
^\* If the PNR field is empty (which means nothing was entered )^
    ^C01
    ^\* Move the cursor to the home position^
    ^SR
    ^\* Put a SOM on the screen^
    ^\EXIT^
    ^\* Exit the key^
^\ELSE^
^\* If the user did enter something...^
    ^SR^C**
    ^\* Put a SOM on the screen and put a “*”(the “PNR display” character) after the SOM^
^\ENDI^
^\* End of the IF statement^
^\WRIT PNR^^E^H
^\* Put the contents of the variable PNR on the screen, send it to the Host, and suspend further
processing by the PF key script until the Host responds^
^\IF (LASTHOSTMSGLINES<3)^
^\* If the Host sent back less than 3 lines, assume it didn’t find the PNR, so^
    ^C01^K^\WRIT "Can't find PNR " PNR " , try again"^
```

```
    ^\* Clear the screen, and tell the user what happened^
    ^SR^C*
    ^\* Put a SOM on the screen^
    ^\EXIT^
    ^\* Exit the PF key^
^\ENDI^
^\*End of the IF statement^
^C01^K^SR*IA^E^H
^\* Clear the screen, put “*IA” on the screen, send it to the Host, and suspend further
processing until the Host responds^
^\IF (LASTHOSTMSGLEN<20)^
^\* Assume there are no flight segments if the Host sent back less than about 20 characters,
so^
    ^C01^K^\WRIT "No Air Itinerary"^
    ^\* Put a message on the screen for the user^
    ^SR^C*
    ^\* Put a SOM on the screen^
    ^\EXIT^
    ^\* Exit the PF key^
^\ELSE^
^\* The message is long enough to assume there are flight segments^
    ^\SETI FLTLINES LASTHOSTMSGLINES-1^
    ^\* Save the number of flight segments, the Host also sent back a line with a
    SOM on it^
^\ENDI^
^\* End of the IF statement^
^\CATS ANS "   The United Flight Numbers are:" RETURN RETURN^
^\* Create a string called ANS that includes a heading line and 2 carriage returns^
^\SETI UAFLT 0^
^\* Initialize a counter for the number of United Flight Segments^
^\SETI LINENBR 0^
^\* Initialize a variable that keeps track of which line number in the Host display we are
working on^
^\WHIL (FLTLINES > LINENBR)^
^\* Set up a loop that runs as long as the number of flight segments is greater that the line
from the Host we are working on^
    ^\SETI LINENBR LINENBR + 1^
    ^\* Increment the line number^
    ^\SETS AIRLINE 4,LINENBR,5,LINENBR^
    ^\* Create a variable that contains the airline code on the line in the Host
    display that we are currently working on^
    ^\IF (CMPS AIRLINE "UA")^
    ^\* If the airline code is “UA”, then^
        ^\SETS FLT 6,LINENBR,9,LINENBR^
        ^\* Put the flight number into a variable^
        ^\TRIM FLT^
        ^\* Trim off any leading spaces^
        ^\CATS ANS FLT RETURN^
        ^\* Add the flight number and a carriage return to our “Answer” variable^
        ^\SETI UAFLT 1^
        ^\* Indicate that we have found a UA flight^
    ^\ENDI^
    ^\* End of the IF statement^
```

```
^\ENDW^
^\* End of the While statement^
^\IF (UAFLT = 0)^
^\* If we did not find any United flights, then^
    ^\WRIT "Sorry, no United Flights"^
    ^\* Put a message on the screen^
    ^\BEEP^
    ^\* Make the terminal beep^
^\ELSE^
^\* If we did find United flights^
    ^C01^K^\WRIT ANS^
^\*Clear the screen and display the “answer” string on the screen^
^\ENDI^
^\* End of the IF statement^
^SR^C*
^\* Put a SOM on the screen^
```

WinIATE ™

Version 2.5

PF Keys and Scripting

Reference Manual

InnoSys
INCORPORATED
3095 Richmond Parkway, Suite 207
Richmond, CA 94806
+1 510 222-7717

InnoSys Incorporated
3095 Richmond Parkway, Suite 207
Richmond, CA 94806
(510) 222-7717 Voice
(510) 222-0323 FAX
info@innosys.com