# CP-6

# 6Edit Screen Editor Reference Manual

Worldwide
Information
Systems

Bull

CP-6

# 6EDIT SCREEN EDITOR REFERENCE MANUAL

SUBJECT

Reference Information for the Bull CP-6 6Edit Screen-Oriented Editor

SPECIAL INSTRUCTIONS

This edition supersedes CE70-01, dated December 1988.

SOFTWARE SUPPORTED

6Edit Version A03 under CP-6 Operating System E00

DATE

June 1990

ORDER NUMBER

CE70-02

Worldwide
Information
Systems

Bull

# Preface

This reference describes the concepts, features, and commands for the A03 version of the CP-6 6Edit screen editor, running on the E00 version of the CP-6 operating system. This reference is intended for the experienced user of 6Edit.

UNIX is a registered trademark of AT&T.

The Bull Los Angeles Development Center Documentation Group authors, edits, reviews and creates laser print masters with integrated text and graphics using CP-6 CAP (Computer Aided Publication).

Readers of this document may report errors or suggest changes through a STAR on the CP-6 STARLOG system.

# Table of Contents

**Figures:**

# About This Manual

This document describes the CP-6 6Edit screen editor.

This manual is intended to be a complete reference for the experienced user, and is intended for reference rather than tutorial use. The manual is organized as follows:

SECTION 1, OVERVIEW OF 6EDIT, describes the conceptual model upon which 6Edit is based: the objects it manipulates and the operations it performs on those objects. No details of actual use are given in Section 1, however.

SECTION 2, USING 6EDIT, presents the details of how to run 6Edit, the environment it requires, and how it appears to the user.

SECTION 3, 6EDIT COMMANDS, presents all of the 6Edit commands, their options, and examples of their use.

SECTION 4, STRING EXPRESSIONS, describes string expressions, how to form them, and how they are evaluated by 6Edit.

SECTION 5, BLOCK EXPRESSIONS IN 6EDIT, describes block expressions, how to form them, and how they are evaluated by 6Edit.

APPENDIX A, PREDEFINED NAMES, lists 6Edit's predefined names and describes their values and uses.

APPENDIX B, PREDEFINED STRING FUNCTIONS, describes the built-in functions supported by 6Edit in string expressions.

APPENDIX C, CONTEXT FILES IN 6EDIT, describes the standard context file and its components, and presents examples of terminal-specific context files.

APPENDIX D, CUSTOMIZING THE 6EDIT USER INTERFACE, discusses how to tailor the 6Edit user interface to your individual needs.

APPENDIX E, INPUT EDITING FUNCTIONS, lists all input editing functions available to the 6Edit user.

APPENDIX F, THE ASCII CHARACTER SET, lists the ASCII character set with decimal conversions.

## Changes Since Last Release

There have been several changes to this manual to answer STARs and to correct spelling, format, and the HELP facility. These changes have been marked with change bars. Significant content changes are:

Overview

o The new predefined string function $KEY (or <key>) is explained in 'Function Key and Control Key Notation', Section 1. $KEY (or <key>) is provided to represent function key names in KEYIN commands or EQUALS commands.

o The parsing of $CONTROL-functions in string expressions is now performed only when the value of the string is actually used. It is no longer done when simply assigning the string expression to an EQUALS variable. In previous versions, parsing was performed at both times. So, for example, if a terminal had a function key sending <ESC><%>, an EQUALS command might assign that string to a variable with the key's name, for later use in a KEYIN command. In previous versions, the EQUALS command would require four percents. In A03, it only needs two. See '$CONTROL Identifier', Section 1.

Using 6Edit

o 6BUILD and 6X are alternate command names to invoke 6Edit.

o 6Edit chooses unused UC streams for command and editing windows. Normally, it will use UC10 for the command window and UC11 for the first editing window. Additional editing windows normally start at UC12. These points are discussed in 'Command Window' and 'Editing Window' in Section 2 (per star 32148).

o 'Editing Window' in Section 2 explains that it is now possible to enter a command in the command window and have the cursor remain there for another command, by terminating the command with <CNTL-C> instead of <CR>.

o 'Multiple Editing Windows' has been added to Section 2.

o 6Edit can now be used conveniently in batch or on-line modes, or with non-CRT terminals. 'Serial Editing' in Section 2 and the subheadings 'Switching between Serial and Screen Editing' and 'Manipulating Data in Serial Editing Mode' have been added.

6Edit Commands

- o The DELETE FILE form of the DELETE command has been added to Section 3 (per STAR 23879).
- o DO or ! commands can now be used to call other processors from within 6Edit (per STAR 23353).
- o The EQUALS command syntax in Section 3 is changed. The IN phrase now permits identification of a specific editing window.
- o An example showing the $KEY predefined string function has been added to the description of the EQUALS command in Section 3 (per STAR 28846).
- o The IF and ELSE commands, which provide conditional execution of multiple commands in a single line, have been added to Section 3.
- o The KEYIN command syntax in Section 3 is changed. The IN phrase now permits identification of a specific editing window.
- o The KEYIN command description in Section 3 has been expanded to explain and illustrate the use of the $KEY predefined string function in KEYIN commands (per STAR 28846).
- o A new command, SHOW, displays 6Edit name definitions and selected records (per STAR 26518); see Section 3.
- o A new command, WINDOW, has been added to Section 3 (per STAR 26429).

String Expressions

- o The discussion of string expressions has been expanded.

Block Expressions in 6Edit

- o The BY option is changed for cases in which a block expression also creates a new edit block: that key increment overrides the value in the $BY predefined name for the rest of the life of that edit block. See See the 'BY Option' example, Section 5.
- o It is now possible to create and manipulate consecutive files and keyed files with text keys instead of 3-byte binary edit keys, as explained in 'FILE Block Operand', Section 5.
- o A new $CONTROL function, W, for wildcard pattern searching has been added to Section 5 (per STAR 32349).

- String block expressions can now be constructed from the corresponding wildcard portions of pattern expressions. See 'STRING Block Operand', Section 5.

Predefined Names

- The following new predefined names have been added to Appendix A:

  $BY, $CONTEXT, $DIRECTION, $END_MARK, $EXIST, $FILEORG, $FILERECORDS, $FILETYPE, $INITIALIZE, $KEY_GENERATION, $POINT, $PROTECT, $REKEY, $SCREEN, $SCROLL, $TEXTEDIT, $WI_BORDER, $WI_PERCENT, ANY, AO, BACKWARD, BIN10, BINHLF, BIN521, BO, DONT_PROTECT, EO, FORWARD, NEW, OLD, PROTECT, STRING (per STARs 36719, 30049, 26518).

- $BY existed previously but was not documented. The default key increment for new records, $BY, is now set and displayed as an edit key. If $BY was used with previous versions of 6Edit, the values used will now have to be divided by 1000.

- The description of the $MATCH_LIMIT predefined name in Appendix A has been modified.

Predefined String Functions

- Three new predefined string functions have been added to Appendix B: $CMDVAR and $KEY (per STAR 28846), and $CNTL.

- The $INPUT string function described in Appendix B now reads from the command window instead of M$UC. This change, the result of an improvement to the X$EVAL routines, allows a more convenient interface for a "string substitution" KEYIN or SYNONYM command. Any other new features and bug fixes in the X$EVAL host library package as of 3/1/90 are also available in A03 6Edit.

Context Files in 6Edit

- Sample context files -- for DECTV100, PCTV7800 and PCTX364 -- have been added to Appendix C.

Customizing the 6Edit User Interface

- A discussion of function key definitions on a personal computer, using the KEYIN command with the $KEY predefined string function, has been added to Appendix D.

Input Editing Functions

o These functions have been added: <ESC> <CNTL-D>, <ESC> <E>, <ESC> <n> <H>, <ESC> <n> <M>.

Additional Changes

In addition to the changes specific to this manual, the following software changes and corrections have also been made to 6Edit:

o String expressions may now include predefined names that start with a dollar sign (per 32020). For example, SHOW LOCATIONS $NOT_COPIED .

o When REKEY (the synonym for COPY CURRENT OVER CURRENT) is used to rekey a file, the cursor now returns to the current position in the editing window (per STAR 25698).

o $CONTROL functions with more than one parameter, such as %V(m,n) and %W(min,max), now work properly in pattern-matching strings. For example, PATTERN %V(75,80). (Per STAR 32349.)

o Moving a string to the left within the same record now works properly (per STAR 37411/25800).

o When you attempt to edit a file that someone else has open in PROTECT mode, 6Edit now lets you open the file, still keeping it in PROTECT mode. The message "File filename is busy; open for reading only" is displayed. (Per STAR 26824.)

o Per star 31671, 6Edit now stays in serial mode long enough for context file changes to control the building of the command window.

## On-Line HELP Facility

The 6Edit processor supports an on-line HELP facility. 6Edit users can display syntax formats, parameter descriptions, and examples at the terminal.

The 6Edit HELP facility includes information about the 6Edit processor and commands.

For a list of HELP topics from the system command level (!) or from another processor, enter the following at the terminal:

```
HELP (6EDIT) TOPICS
```

From within the 6Edit processor, enter:

```
HELP TOPICS
```

## Related Manuals

Following is the list of manuals supporting the CP-6 Operating System, AR 1.0 Release:

| ORDER NUMBER | TITLE | PUBLICATION DATE | | |
|---|---|---|---|---|
| End User Facilities | | | | |
| HA03-01 | CP-6 Introduction to MAIL | October | 1986 | * |
| HA04-01 | CP-6 MAIL Reference | September | 1986 | * |
| HA09-00 | CP-6 Introduction to ARGENT | August | 1985 | * |
| HA10-00 | CP-6 ARGENT Reference | October | 1985 | * |
| HA12-01 | CP-6 ADAPT Reference | December | 1988 | * |
| HA13-01 | CP-6 FORGE Reference | December | 1988 | * |
| HA15-00 | CP-6 PC Terminal (PCT) Facility Reference | July | 1986 | * |
| CE30-02 | CP-6 IDP Reference | July | 1982 | * |
| CE70-02 | CP-6 6Edit Screen Editor Reference | June | 1990 | * |
| CE73-00 | CP-6 Introduction to 6Edit Screen Editing | April | 1985 | * |
| Database Management | | | | |
| HA01-01 | CP-6 Introduction to ARES | August | 1985 | * |
| HA02-03 | CP-6 ARES Reference | June | 1990 | * |
| CE35-03 | CP-6 I-D-S/II Reference | December | 1988 | * |
| CE36-03 | CP-6 I-D-S/II DBA Reference | December | 1988 | * |
| CE54-01 | CP-6 I-D-S/II Guide | December | 1988 | * |
| Publishing | | | | |
| HA07-00 | CP-6 CAP Administrator Guide (A00) | February | 1987 | * |
| HA08-00 | CP-6 CAP Reference (A00) | February | 1987 | * |
| HA18-00 | CP-6 Introduction to CAP (A00) | April | 1987 | * |
| HA19-00 | Getting Started with CAP (A00) | August | 1987 | * |
| HA27-00 | CAP DSL Reference (B00) | February | 1989 | S |
| HA28-00 | CAP ADSL Reference (B00) | February | 1989 | S |
| HA29-00 | CP-6 CAP User Guide (B00) | February | 1989 | S |
| HA30-00 | CP-6 CAP Administrator Guide (B00) | February | 1989 | S |
| CE48-02 | CP-6 TEXT Processing Reference | February | 1987 | * |
| CE53-00 | CP-6 TEXT Processing Primer | June | 1981 | * |
| CE59-00 | CP-6 FASTEXT Guide | December | 1982 | S |

Transaction Processing

| | | | |
|---|---|---|---|
| CE49-01 CP-6 TP Applications Programmer Guide | August | 1985 | * |
| CE50-02 CP-6 TP Administrator Guide | December | 1988 | * |
| CE51-02 CP-6 FPL Reference | December | 1988 | * |

Application Programming

| | | | |
|---|---|---|---|
| HA17-00 CP-6 C Language Reference | June | 1990 | * |
| CE28-01 CP-6 SORT/MERGE Reference | February | 1983 | * |
| CE29-02 CP-6 COBOL Reference (COBOL-74) | February | 1983 | * |
| CE46-02 CP-6 COBOL Programmer Guide (COBOL-74) | February | 1983 | * |
| CE68-00 CP-6 COBOL Reference (COBOL-85) | April | 1986 | * |
| CE69-00 CP-6 COBOL Programmer Guide (COBOL-85) | April | 1986 | * |
| CE31-05 CP-6 FORTRAN Reference (FORTRAN-77) | December | 1988 | * |
| CE47-05 CP-6 FORTRAN Programmer Guide (FORTRAN-77) | December | 1988 | * |
| CE32-03 CP-6 BASIC Reference | March | 1985 | * |
| CE37-00 CP-6 RPGII Reference | December | 1979 | N |
| CE38-05 CP-6 APL Reference | December | 1988 | * |
| CE40-04 CP-6 Programmer Reference | December | 1988 | * |
| CE42-03 CP-6 Programmer Pocket Guide | August | 1989 | * |
| CE55-01 CP-6 Application Programmer Handbook | January | 1984 | * |
| CE72-00 CP-6 DIGS (Graphics) Reference | March | 1985 | * |

System Programming and Support

| | | | |
|---|---|---|---|
| HA11-00 CP-6 FEP Programming Concepts | May | 1985 | S |
| HA20-01 CP-6 System Support Reference (A-P) | December | 1988 | * |
| HA21-01 CP-6 System Support Reference (Q-Z) | December | 1988 | * |
| HA22-01 CP-6 System Support Reference (Appendices) | December | 1988 | * |
| CE34-05 CP-6 Operations Reference | December | 1988 | * |
| CE39-04 CP-6 DELTA Reference | December | 1988 | * |
| CE44-03 CP-6 PL-6 Reference | December | 1988 | * |
| CE60-00 CP-6 System Manager Handbook | March | 1985 | * |
| CE61-01 CP-6 Customer Support Handbook | October | 1987 | * |
| CE62-00 CP-6 System Programmer Guide | January | 1984 | * |
| CE64-03 CP-6 Operations Pocket Guide | August | 1989 | * |
| CE65-00 CP-6 FEP Assemblers Reference | March | 1985 | * |
| CE66-02 CP-6 FEP Monitor Services Reference | December | 1988 | * |
| CE67-01 CP-6 FEP Library Services Reference | December | 1988 | * |
| CE71-02 CP-6 Host Library Services Reference | December | 1988 | * |
| CE74-01 CP-6 Host Monitor Services Reference (Desc.) | December | 1988 | * |
| CE75-01 CP-6 Host Monitor Services Reference (Struc.) | December | 1988 | * |

General Purpose

|  |  |  |  |  |
|---|---|---|---|---|
| HA16-01 | CP-6 X Account Pocket Guide | August | 1989 | * |
| CE45-01 | Getting Started with Timesharing | November | 1987 | * |
| CE56-03 | CP-6 Pocket Guide to Documentation | February | 1987 | * |
| CE58-00 | CP-6 Monitor Error Message Reference | December | 1988 | * |

Hardware

|  |  |  |  |  |
|---|---|---|---|---|
| DH03-01 | DPS 8 Assembly Instructions |  |  | N |
| DX20-00 | DPS 90 Assembly Instructions |  |  | N |
| DZ51-00 | DPS 8000 Assembly Instructions |  |  | N |

Legend

* = order from Newton Highlands (also supplied on software release tape)
S = system-supplied only (not currently available from Newton Highlands)
N = available only from Newton Highlands
~ = electronic update available on release tape (E00 or prior release)

Ordering

Manuals may be ordered using Form No. HB-2808 from:

Bull HN Information Systems Inc.
Customer Services Operation
Publications Order Entry
141 Needham Street
MA35/219
Newton Highlands, MA 02161 U.S.A.

or may be ordered by telephone:

|  |  |
|---|---|
| Internal orders: | (617) 552-5199 (fax) |
|  | (617) 552-5374 |
| Customer orders*: | (800) 343-6665 |
|  | (617) 552-5199 (fax) |

* publications and supplies

# Notation Conventions

Notation Conventions used in command specifications and examples are listed below.

| Notation | Description |
|---|---|
| Brackets | Brackets are used to enclose an optional element. For example: |
| | [movement] indicates that a value for movement may be entered. |
| | When enclosing keywords, brackets signify that all of the bracketed portion may be entered or omitted. For example: |
| | DI[SPLAY] indicates that the command DISPLAY can be entered as either the word DISPLAY, or simply as the first two characters, DI. |
| | Note: A slash (/) may replace brackets in a figure or list of commands. For example: |
| | AC/CEPT SEND means that the first two characters of the ACCEPT name may be entered. |
| | If more than one element is enclosed in brackets, the notation indicates an optional choice. Multiple elements in brackets are separated by an OR bar or listed on separate lines. For example, |

| Notation | Description |
|---|---|
| | `[ ALL | ldevlist ]`<br><br>and<br><br>`[   ALL    ]`<br>`[ ldevlist ]`<br><br>have identical meaning: either ALL or a list of logical device names is permitted. |
| OR Bar | The OR bar separates elements enclosed in braces or brackets from which one must or may be chosen. For example:<br><br>`{END|XIT}`      indicates that either END or XIT may be entered. |
| Braces | Braces around words separated by \| (OR bars) indicate a required choice. For example:<br><br>`{E[ND]|XI[T]|QU[IT]}`     means either END, XIT, or QUIT must be selected. |
| Lowercase | Lowercase letters identify an element that must be replaced by a user-selected value. For example:<br><br>`DE[LETE] BL[OCK] block_expression`   indicates the user supplies a value for block_expression. |

| Notation | Description |
|---|---|
| Careted Letters | Letters inside carats (<>) identify physical keys on the terminal. Carats are not typed. The indicated keys are pressed. For example:<br><br><ESC>     indicates touch the Escape key.<br><br>Another example:<br><br><CNTL-T> means press and hold down the Control key while pressing the "T" key. |
| Horizontal Ellipsis | Horizontal ellipsis indicates that a previous bracketed element may be repeated or that elements have been omitted. For example:<br><br>movement[[movement]...]   indicates that one or more movement expressions may be entered. |

# Section 1

# Overview of 6Edit

This section describes the major concepts behind the operation of the CP-6 6Edit file editor. Actual procedures are covered later in the manual.

## What is 6Edit?

6Edit is a screen-oriented file editor. It inserts, deletes, and replaces data in a file. Although 6Edit accesses the file by reading and writing records, when using 6Edit you do not have to give instructions in terms of records.

You normally interact with 6Edit using "full-screen editing," wherein 6Edit displays a portion of the file on the screen and you can update it directly by simply moving the cursor to the desired data on the screen, and typing in new data.

When full-screen editing, you press keys on your terminal. Some of these keys cause 6Edit to perform actions, and some of the keys simply insert data into the file being edited. As you move the cursor, 6Edit automatically scrolls the file data up or down, using the terminal screen as a movable "window" on the data in the file.

6Edit allows you, the user, to edit data in a variety of ways. For example:

- o Interactive editing, using the editing window. Single keystrokes on your terminal keyboard allow you to perform many common editing functions, such as ad hoc or repeated changes (e.g., change all x's to y's).

- o Interactive editing, using the command window. Verbal commands typed in 6Edit's command window enable more complex changes and additions, often involving data at arbitrary locations throughout the file.

- o Customization. The ability to change commands with user-defined synonyms, to alter key function definitions, and to customize the 6Edit context file for individual editing needs, makes 6Edit a highly versatile editing processor.

Most of the editing that occurs in 6Edit consists of selecting the data to be changed, and actually changing the data. 6Edit offers flexibility in both of these steps. Users can perform them:

- o Interactively, for ad hoc editing.
- o Semi-automatically, visually selecting information, then either skipping or changing the selection.
- o Automatically, using any combination of commands, which can be automatically repeated throughout the file.

## Files, Record Keys, and Records

6Edit is used to edit disk files only. It can only access edit-keyed, string-keyed, consecutive, and unit record (UR) files. Edit-keyed files are CP-6 keyed files with 3-byte binary keys. Each record has a record key associated with it. The record keys are numbers between 0 and 99999.999, inclusive, with at most three digits to the right of the decimal point.

### File Pointer

6Edit keeps track of a "file pointer" which "points" to a location in a file. You can move the file pointer in three ways:

- o When full-screen editing, the cursor is effectively the file pointer. When you move the cursor (with, for example, the backspace key on the terminal), you are moving 6Edit's file pointer.
- o You can use the terminal's function keys to move the file pointer, by moving forward or backward in a file in units of characters, words, records, matches of a pattern string, etc. You must first tell 6Edit what the terminal's function keys mean, usually in a "context file." (See Appendix C, Context Files in 6Edit.)
- o You can enter commands to move the file pointer in the same way as the terminal function keys. (Actually, the terminal function keys simply generate commands; anything you can do with commands, you can make your terminal function keys do). You can also enter commands to move the file pointer to a specific file, or a record within a file, by typing the specific file name and/or record key in a command.

## Blocks

The 6Edit editing commands work with "blocks" of data. A block can be an entire file, or any portion of a file. As shown in Figures 1-1 through 1-4, a block is usually a string of characters taken from one or several records in a file.

A block can be a portion of one record, excluding the record boundary at the end of the record.

Example:

```
  The elevator doors in the lobby
      ----------------------
of the|Second National Bank|Build-
      ----------------------
ing downtown were open late Wed-
nesday afternoon, but the elevator
car was not there.  Tom Hall, 42,
```

Figure 1-1. Example of a Block

A block can be all or part of a record, including the end-of-record boundary.

Example:

```
  The elevator doors in the lobby
      ------------------------------
of the|Second National Bank Build- |
      ------------------------------
ing downtown were open late Wed-
nesday afternoon, but the elevator
car was not there.  Tom Hall, 42,
```

Figure 1-2. Example of a Block

A block can be a set of records, with portions of the first and last records (excluding the final end-of-record boundary).

Example:

```
     The elevator doors in the lobby
         ------------------------------
   of the|Second National Bank Build- |
----------                            |
|  ing downtown were open late Wed-   |
|                   -------------------
|  nesday afternoon,|but the elevator
---------------------
   car was not there.  Tom Hall, 42,
```

Figure 1-3. Example of a Block

A block can be a set of records, with all or part of the first record and all of the last record, including the final end-of-record boundary.

Example:

```
     The elevator doors in the lobby
         ------------------------------
   of the|Second National Bank Build- |
----------                            |
|  ing downtown were open late Wed-   |
|  nesday afternoon, but the elevator |
---------------------------------------
   car was not there.  Tom Hall, 42,
```

Figure 1-4. Example of a Block

## Specifying a Block

You specify a block of data to edit by "selecting" the block. There are two ways to select a block:

- o When full-screen editing, you can direct the cursor to the beginning and the end of the block, moving it with the arrow keys or the function keys on the terminal. Moving the cursor is equivalent to moving the file pointer.

- o You can enter commands which direct the file pointer to the beginning and the end of the block to be selected.

## What You Can Do With a Block

Once a block is selected, you can enter an editing command (or use a function key on your terminal) to manipulate the block of data. The things you can do with a block of data include:

- o Delete the selected block of data.

- o Copy the block of data to another location, either in the same file or a different file.

- o Move the block of data to another location (same as copying the block, except that after copying it, it is deleted from its original location) either in the same file or a different file.

- o Insert previously selected data after the selected block.

- o Replace the selected block with another block of data (which you previously selected).

- o Give a name to the location of the selected block. Thereafter, using that name in a 6Edit command tells 6Edit to use the current data at that location.

- o Edit the selected block. (See "Edit Block" below.)

There are two special blocks used in 6Edit: the edit block and the selected block.

## Edit Block

The implicit subject of all editing operations is the "edit block." The edit block is any set of records in any file. It can comprise one record or an entire file. Usually the edit block is all or a large portion of a file.

You can edit less than an entire file, simply by telling 6Edit which portion of the file, or block, you want to edit. This is useful when you want to limit editing operations to one portion of the file.

You can set the edit block explicitly with the EDIT command; you can refer to it explicitly using the "CURRENT" predefined name.

The default for all editing commands is to edit a block which is located entirely inside the edit block. You can override this limitation, and specify a block anywhere in any file. (See Section 5, Block Expressions in 6Edit.)

When you specify a file, or some other block operand which is located outside the current edit block, 6Edit automatically changes the edit block to the specified file. This is called an "implicit EDIT" because it is as though 6Edit automatically inserted an EDIT command in front of the command you entered. An implicit EDIT can occur during the following commands:

```
    AFTER
    DELETE
    DISPLAY
    LOCATION
    OVER
```

For example, assume you have typed the following command:

```
    EDIT FILE ACCTS_PAYABLE
```

The edit block is now all of file "ACCTS_PAYABLE". If you type the command:

```
    DISPLAY 23
```

it displays record 23 from file ACCTS_PAYABLE. Your search is limited to the material within the existing edit block only; 6Edit does not search for record 23 before the first record, or beyond the last record of the current edit block.

Now, you type this command:

```
DISPLAY FILE INVENTORY 23
```

Because you explicitly specified a file, it becomes the new edit block; the command displays record 23 from file "INVENTORY", and the new edit block is all of file INVENTORY.

Note that the following two commands have the same effect on the edit block:

```
EDIT FILE ACCTS_PAYABLE
DISPLAY FILE ACCTS_PAYABLE
```

Contrast with this the very different effects of the following two commands on the edit block:

```
EDIT FILE ACCTS_PAYABLE 23
DISPLAY FILE ACCTS_PAYABLE 23
```

The EDIT command in this example sets the edit block to just record 23 of file ACCTS_PAYABLE; the DISPLAY command in the example sets the edit block to all of the ACCTS_PAYABLE file, then displays record 23.

### Edit Block Stack

When you change to a new edit block, the previous edit block is not forgotten. 6Edit maintains an "edit block stack", which holds the specifications of previous edit blocks.

The edit block stack has a pointer, which refers to the current edit block specification (the "CURRENT" predefined name). When the edit block changes, either explicitly (EDIT command) or implicitly, 6Edit adds an entry to the edit block stack, and stores the specification of the new edit block in that entry.

You can easily refer to the block specification in the previous entry in the edit block stack by using the PREVIOUS block operand in a block expression. The PREVIOUS block operand always changes the edit block, but instead of appending the new edit block specification to the end of the edit block stack, 6Edit simply moves the edit block stack's pointer backward one entry. The new edit block is the previous edit block specification from the edit block stack. Once you have used PREVIOUS, you can similarly use the NEXT block operand to advance the edit block stack's pointer. The following examples illustrate responses generated by the edit block stack.

Note that when you change to a new edit block, thus adding a new entry to the stack, any entries from the current entry to the end of the stack are removed from the stack. Thus, you may not EDIT NEXT after introducing a new entry into the stack, since there is no longer a NEXT file in the stack.

Table 1-1. Operations in the Edit Block Stack

```
You type:                  Edit Block Stack:

                            -----------
EDIT  FILE  ABC            | FILE  ABC |
                            -----------
                             current

                            -----------------------
EDIT  FILE  DEF            | FILE  ABC | FILE  DEF |
                            -----------------------
                                         current

                            -----------------------
EDIT  PREVIOUS             | FILE  ABC | FILE  DEF |
                            -----------------------
                             current

                            -----------------------
EDIT  NEXT                 | FILE  ABC | FILE  DEF |
                            -----------------------
                                         current

                            -----------------------------------
EDIT  FILE  GHI            | FILE  ABC | FILE  DEF | FILE  GHI |
                            -----------------------------------
                                                     current

                            -----------------------------------
EDIT  PREVIOUS             | FILE  ABC | FILE  DEF | FILE  GHI |
                            -----------------------------------
                                         current

                            -----------------------------------
EDIT  PREVIOUS             | FILE  ABC | FILE  DEF | FILE  GHI |
                            -----------------------------------
                             current
```

Table 1-1. Operations in the Edit Block Stack (cont)

```
EDIT  FILE  XYZ          -----------------------
                         | FILE  ABC | FILE  XYZ |
                         -----------------------
                                      current
```

## Selected Block

The "selected block" is the last block you specified, either by moving the file pointer (cursor) or by entering commands. You manipulate the selected block using the editing commands: it can be deleted, moved, replaced, etc.

For example, if you move the cursor to the beginning of a sentence, type THRU in the command window, then move the cursor to the end of the sentence, you have designated the new selected block. If you then type DELETE THAT in the command window, 6Edit deletes the sentence from the file.

## Compound Blocks

In complex editing situations, you can specify "compound blocks" in 6Edit commands. There are two types of compound blocks: repeated blocks and enclosed blocks.

### Repeated Blocks

Sometimes you want an editing operation to be performed at several locations in a file. For example, you may want to replace all appearances of the word "horse" with the word "giraffe." Rather than find each location and perform the operation manually each time, 6Edit allows you to enter a command which specifies a "repeated block."

Specification of a repeated block in a command causes the entire command to be repeated as many times as needed, each time at a different location in the file. A block expression tells 6Edit how to move the file pointer to find the location of each repetition of the operation. (See below.)

#### Enclosed Blocks

"Enclosed blocks" are used in commands to limit movement of the file pointer temporarily. For example, you can specify that a pattern search be limited to a single record. This single record is called the "enclosing block."

## Expressions

6Edit commands involve two types of expressions: string expressions and block expressions.

### String Expressions

A string expression specifies either a number or a string of characters, depending on the context in which it is used. A string expression can combine different types of strings into one string by concatenating them. Also, a string expression can combine strings and numbers with the usual logical, relational, and arithmetic operators, producing a string of characters which represent logical (true or false), or numeric results.

The types of strings which can be used in string expressions are:

o String constants: strings of characters enclosed in apostrophes ('), or decimal numbers (not enclosed in apostrophes). The strings or decimal numbers represent constants. Character data (including non-displayable characters) may be entered using ASCII characters, their decimal codes, or their ASCII names. (See $CONTROL-Value Function later in this section.)

o EQUALS-names: names of variables which have been assigned string values with the EQUALS command. These may be names you create or predefined names. 6Edit uses the value of a name in place of the name when evaluating the string expression.

o Predefined string functions: functions analogous to IBEX system functions that yield values (such as the date, the current directory account, the absolute value of an expression), and functions such as $CNTL and $KEY that yield strings that identify control keys, function keys, and special keystroke sequences.

## Block Expressions

A block expression specifies a block of data in a file. All editing commands have a single block expression as a parameter. This expression specifies the block of data upon which the editing command is to operate (to delete the block, to copy the block, to replace the block, etc.)

Block expressions can indicate whether the block resides in a specific file, or if it resides in the edit block (the block currently being edited). The boundaries of a block can be given in absolute terms, such as a specific location in a file, or in relative terms, such as forward or backward from the current file pointer location, or in a combination of these.

The LOCATION command allows you to give a name to the location of a block. Later, you can use that name in a block expression to refer to the designated block.

Block expressions can be enclosed and repeated. When a block expression specifies a repeated block, the entire editing operation of which the block expression is a parameter is repeated.

## Keywords and Abbreviations

6Edit commands use a keyword-style notation. All keywords may be spelled out fully, or abbreviated. You can choose any editing function by simply using the alphabetic characters, the numerals, and seven special characters. The following figure summarizes the keywords and non-alphabetic characters used.

Keywords

| | | | |
|---|---|---|---|
| 6BUILD | * DONT | MOVE | * REPEAT |
| 6EDIT | EDIT | NEW | RESET |
| 6X | EDITING | * NEXT | * RESTORE |
| ADJUST | END | NUMBER | * SAVE |
| AFTER | EO | OF | SELECT |
| ALL | EQUALS | OLD | * SESSION |
| ANY | ** ERASE | ON | * SETUP |
| AO | EXIT | *** OUTPUT | SHOW |
| BACKWARD | FILE | OVER | SKIP |
| BEGINNING | FORWARD | PATTERN | STRING |
| BLOCK | * HELP | PATTERNS | SYNONYM |
| BO | * HERE | PERCENT | SYNONYMS |
| BY | IN | POSITION | * THAT |
| COMMAND | IS | POSITIONS | THROUGH |
| COPY | * KEY | * PREVIOUS | THRU |
| * CURRENT | KEYIN | ** PRINT | * TIME |
| * DATE | KEYINS | PROTECT | TO |
| DELETE | LINES | QUIT | ** TOPICS |
| *** DIRECTORY | LOCATION | READ | WINDOW |
| DISPLAY | LOCATIONS | RECORD | WINDOWS |
| | | RECORDS | XIT |

All of the above may be entered in upper or lower case and may be abbreviated to exactly the first two characters, except those preceded by one or more asterisks.
* - must be abbreviated to exactly the first four characters.
** - abbreviation is not permitted.
*** - may be abbreviated with three or more characters.

Non-alphabetic Characters

Numerals: 0 through 9
Special Characters: ' ( ) , . ; ?

Figure 1-5. Keywords and Reserved Special Characters

## Command Lines and Continuation

A "command line" is one record from the "command stream." The command stream usually originates from the keyboard of your terminal. (With the READ command, you can switch the command stream to any file. Command lines will then be read from that file. See Section 3, 6Edit Commands.)

Command lines contain one or more commands. Multiple commands in one command line are separated by a semicolon (;).

If the command you type is longer than one line, type a semicolon as the last character of each ongoing line (except the last line of the command).

Example:

```
CO ST 'This drawn-out line renders the command lengthy' AFTER;
FI XYZ 20.5
```

## Lexical Functions

You can instruct 6Edit to perform certain lexical functions during command entry. You can invoke these functions by using identifier characters, which are the values of two predefined names: $CONTROL and $COMMENT.

Initially, 6Edit uses the following default identifier characters for the $CONTROL and $COMMENT functions:

| | |
|---|---|
| % | invokes the $CONTROL function |
| " | invokes the $COMMENT function |

6Edit looks for the $COMMENT identifier in command input and READ-file records, and for the $CONTROL identifier in quoted strings. It does not look for these identifiers (nor perform their functions) in records read from data files.

It is possible to use the $CONTROL or $COMMENT identifier without invoking its lexical function. This can be accomplished in two ways. One is to disable the name. To do this, you assign an empty string to the predefined name $CONTROL or $COMMENT. Thereafter, 6Edit does not support the functions invoked by $CONTROL or $COMMENT, and will not look for the identifier in command lines, READ-file records, or quoted strings.

Or, when using the $CONTROL identifier, you can enter it twice (see $CONTROL-Literal Function, below.)

The functions invoked by the $CONTROL and $COMMENT identifiers are described below.

## $CONTROL Identifier

The $CONTROL identifier is used to invoke several functions which are called "$CONTROL functions". Initially, the value of $CONTROL is the percent sign (%).

The character following the $CONTROL identifier is called the "function identifier".

Some of the $CONTROL functions require an operand to be included along with the function identifier. Normally, a $CONTROL function is specified as follows:

```
    $CONTROL_identifier  function_identifier
```

These characters are to be entered together; there is no space between them.

Parameters:

$CONTROL_identifier    represents one or two characters designating the current value of the $CONTROL predefined name. (See Lexical Functions, above.)

function_identifier    specifies the function 6Edit is to perform.

For example, assume that the $CONTROL identifier is %:

```
    %%    invokes the $CONTROL-Literal function
    %R    invokes the $CONTROL-Record function
    %V    invokes the $CONTROL-Value function
```

Note: $CONTROL functions used in EQUALS commands are not interpreted when assigned, but only when the resulting variable is used in a different command.

Certain $CONTROL functions require one or two operands along with the function identifier. Enclose both in parentheses following the function identifier:

```
    $CONTROL_identifier function_identifier ([operand])
```

For example:

```
    %V()        invokes the $CONTROL-Value function with no operands
    %V(23)      invokes the $CONTROL-Value function with one operand: 23
    %V(23,255) invokes the $CONTROL-Value function with two operands:
                23 and 255
```

To change the identifier which invokes $CONTROL functions, you assign a different character value to the $CONTROL predefined name. The characters assigned to the $CONTROL name must reside in the following set:

```
    !   "   #   $   %   &   *   +   -   @   :   <
    =   >   /   [   \   ]   ^   _   '   {   |   }   ~
```

However, the characters which represent the current value of the $COMMENT predefined name (initially "), cannot be used in the value of the $CONTROL predefined name.

The following paragraphs describe most of the $CONTROL functions available. See Section 5, Block Expressions in 6Edit, for descriptions of additional $CONTROL functions which may only be used in pattern strings.

## $CONTROL-Literal Function

Function Identifier: the $CONTROL identifier itself.

This $CONTROL function allows you to include the $CONTROL identifier itself in quoted strings. 6Edit recognizes two adjacent occurrences of the $CONTROL identifier as representing a single $CONTROL identifier; no other lexical function is performed.

Example:

```
COPY STRING '48%% are qualified' AFTER HERE
```

inserts the string "48% are qualified" into the file.

## $CONTROL-Record Function

Function Identifier:  R

The $CONTROL-Record function represents an "end-of-record boundary."

This function may be invoked in quoted strings.  The $CONTROL-Record function is the only way to represent an end-of-record boundary in quoted strings.

Example:

```
COPY STRING 'The quick brown fox%Rjumped over the lazy dog%R' AFTER 10
```

inserts two records after record 10.0 in the file being edited.  The first record contains "The quick brown fox"; the second record contains "jumped over the lazy dog".

```
COPY STRING 'whenever available ' AFTER 10 SELECT 'respond '
```

inserts two words into record 10.0 after the word "respond".  Note the absence of any $CONTROL-Record function:  the quoted string does not represent a record, but just a string of characters.  This command creates no new records. Record 10.0 is simply lengthened.

Figure 1-6, below, illustrates the quoted string before the $CONTROL-Record function.

```
We would appreciate an itemized inventory sheet from your
                                -----------
department.  You can | respond | stock
                                -----------
arrives from the Omaha branch.  We will expect a complete

list, signed and dated by the area manager, at that time.
```

Figure 1-6.  CONTROL-Record Function Before

```
COPY STRING 'whenever available%R' AFTER 10 SELECT 'respond '
```

inserts two words into record 10.0 after the word "respond".  Also, because the $CONTROL-Record function is included in the quoted string, a new record is inserted after record 10.0.

Figure 1-7, below, illustrates the quoted string after two $CONTROL-Record commands.

```
CO ST 'whenever available ' AF 10 SE 'respond '


 We would appreciate an itemized inventory sheet from your
                                  ---------------------
 department.  You can respond | whenever available | stock
                                  ---------------------
 arrives from the Omaha branch.  We will expect a complete

 list, signed and dated by the area manager, at that time.

CO ST 'whenever available%R' AF 10 SE 'respond '


 We would appreciate an itemized inventory sheet from your
                          -----------------------------
 department.  You can | respond whenever available |
                          -----------------------------
 stock

 arrives from the Omaha branch.  We will expect a complete

 list, signed and dated by the area manager, at that time.
```

Figure 1-7. CONTROL-Record Function After

## $CONTROL-Value Function

Function Identifier: V

Parameters:

value     is a decimal number between 0 and 511, inclusive.

The $CONTROL-Value function allows you to enter any 9-bit byte value in quoted strings.

This function requires an operand following the function identifier.

For example, assuming that $CONTROL is %, to enter a quoted string which contains a single byte with a value of 23, you would type:

```
    '%V(23)'
```

Example:

```
ESC EQ '%V(27)'
```

defines a name, ESC, with the value of the escape character. The number 27 is the decimal ASCII code for the Escape character.

```
KEYIN '%V(27)A' IS 'harbinger of glad tidings'
```

redefines the terminal key sequence Escape A. Hereafter, whenever <ESC> <A> is typed at the terminal keyboard, it appears as the phrase "harbinger of glad tidings".

## $COMMENT Identifier

The $COMMENT identifier is used to denote commentary in command lines. The quotation mark (") is the initial $COMMENT identifier.

When the $COMMENT identifier appears in a command line outside of a quoted string, 6Edit ignores the following text, until the next appearance of the $COMMENT identifier or the end of the command line. This allows you to append commentary to command lines by separating the commentary from the actual command(s) with the $COMMENT identifier.

Note that the $COMMENT identifier is not recognized inside quoted literal strings.

Example:

```
CASE EQ OFF  "This is commentary
```

is not concluded by a quotation mark, because the commentary occurs at the end of the command line.

```
LOCATION CHAPTER_1 IS "Definition follows:" BO CURRENT THRU  '.brp'
```

differs from the former command line. Here, the commentary occurs in the middle of the command line, and is therefore enclosed by quotation marks.

To change the identifier which invokes $COMMENT functions, you assign a different character value to the $COMMENT predefined name. The characters assigned to the $COMMENT name must reside in the following set:

```
    !   "   #   $   %   &   *   +   -   @   :   <
    =   >   /   [   \   ]   ^   _   '   {   |   }   ~
```

However, the characters which represent the current value of the $CONTROL predefined name (initially %) cannot be used in the value of the $COMMENT predefined name.

## Function Key and Control Key Notation

6Edit recognizes function keys, control keys, and ASCII control codes as string expressions. The key or code name must be enclosed in angle brackets ("<", ">") or be the argument of the $KEY function, for example:

```
    KEYIN <F1> IS ...
    KEYIN <UPARROW> IS ...
    KEYIN <ENT> IS ...
    KEYIN <F1>||'A' IS ...
```

Note that abbreviations accepted by IMP (such as "ENT" for "ENTER" above) will also be accepted by 6Edit.

Also note the last example above: function keys may be used as introducers for multi-key sequences.

6Edit accepts the name of any key on the terminal keyboard inside the angle brackets. This includes not only the FEP/profile-defined function keys, but also the ASCII control characters (plus the DEL character) as mnemonics:

| | | | | | | |
|---|---|---|---|---|---|---|
| NUL | ENQ | LF | SI | DC4 | EM | RS |
| SOH | ACK | VT | DLE | NAK | SUB | US |
| STX | BEL | FF | DC1 | SYN | ESC | DEL |
| ETX | BS | CR | DC2 | ETB | FS | |
| EOT | HT | SO | DC3 | CAN | GS | |

Also, ASCII control keys can be included inside the angle brackets, using the identifier "CNTL":

```
    <CNTL-x>
```

where "x" is any one of: @, A-Z, [, \, ], ^, _
'. a-z, {, |, }, ~

The ASCII mnemonics evaluate to the corresponding ASCII code (0 - 31 and 127). However, use of the function key elements in a numeric expression yields an error; use of the function key elements in an expression not used (eventually) by the KEYIN command yields unpredictable results.

The angle bracket notation is intended to reinforce the notation used in the manuals when referring to keystrokes at the terminal keyboard. See CE70 and CE73, Notation Conventions.

Additionally, the $CNTL predefined string function is permitted in string expressions:

```
    $CNTL
```

$CNTL is much like the <CNTL-x> expression element; it is included as a computational form of <CNTL-x>: $CNTL takes a string argument possibly longer than one character, whereas the angle bracket notation is limited to a single character. $CNTL returns a string which contains the ASCII control-codes for the characters in the original string.

Examples:

```
KEYIN <ESC>||'A' IS <CNTL-C>||'BA SK 20 RE'||<CR>
```

```
KEYIN ESC||'A' IS <ETX>||'BA SK 20 RE'||CR
```

The above two examples are identical; the second assumes that the names "ESC" and "CR" have been defined by the user, probably in the context file as they are now.

```
DOITTOIT EQ $INPUT('Type control char to define: ')
KEYIN $CNTL(DOITTOIT) IS <ESC>||'5R'||$CNTL('[W')
```

Note that <CNTL-[> is Escape, so "$CNTL('[W')" is the sequence <ESC> <CNTL-W>.

## Names

In 6Edit, you can create your own names for things. This makes it easier to remember specific details, such as the location of data. Names also make it easier to refer to something repeatedly, such as an often-used sequence of keywords.

Each name has a value associated with it; the value is the location or object to which the name refers.

You create names with the assignment commands; these commands are also used to change the value of a name. The assignment commands are:

```
EQUALS
LOCATION
SYNONYM
KEYIN
```

Some names are "predefined" by 6Edit; that is, whenever you run 6Edit the names have already been created. These names are used to communicate details between you and 6Edit. You can change the value of some of the predefined names; others may not be changed by you, their values are changed internally by 6Edit.

Names are used in several ways in 6Edit:

- o The EQUALS command assigns a reasonably short value (up to 511 characters) to a name. The value is stored as a string of characters, which may represent numeric or character data, depending on how the name is later used. The name can be used in the expression component of a command; 6Edit uses the name's value in place of the name when evaluating the expression.

- o The LOCATION command assigns to a name the boundaries of a block of data in a file. To later refer to that location in the file, include the name in an editing command; 6Edit treats the name as a reference to the block.

- o The SYNONYM command assigns a fragment of a 6Edit command line to a name. If 6Edit later sees the name in a command line, it replaces it with the fragment which is the name's value.

- o The KEYIN command redefines a key on your terminal. Here, the name being assigned is the actual character or sequence of characters generated by the terminal key, or an encoded representation of the name of the key. When a terminal key is redefined, the CP-6 system will cause it to generate any sequence of characters that you specify. When you type the key (or sequence of keys), the system will echo on your terminal the sequence of characters which represent the name's value.

## Value Types

When assigning a value to a name, you must be aware of the "type" of the value. The type of a value is determined by the assignment command you use to assign the value to a name.

Each of the assignment commands assigns a different type to the name's new value. The types of values supported by 6Edit are:

- o EQUALS-value: a string of up to 511 characters.
- o LOCATION-value: a specific block of data in a specific file.
- o SYNONYM-value: a fragment of a 6Edit command-line.
- o KEYIN-value: an arbitrary sequence of characters as they would be typed at a terminal, including escape and other control characters which may invoke editing functions (such as backspace or carriage return).

Beware of the required "type" of the predefined names. Each of the predefined names may only be assigned values of a specific type. Appendix A, Predefined Names, lists the predefined names along with the command which should be used to assign values to each of them. If you try to assign a value to a predefined name using the wrong command, 6Edit rejects the command.

## Substitution

"Substitution" refers to the substitution of the value of a name for the name itself. Substitution may occur when you enter command lines and data to 6Edit; substitution may also be performed on input from your terminal and from command files.

In 6Edit, there are four methods of substitution. These correspond to the four types of values which a name may have.

The table below contrasts the different methods of substitution.

| Assignment Command | Lexically | Semantically | Evaluated | Type |
|---|---|---|---|---|
| name EQUALS string_exp | checked | checked | now | string |
| LOCATION name IS blk_exp | checked | checked | now | location |
| SYNONYM name IS fragment | checked | unchecked | later | -- |
| KEYIN name IS string_exp | unchecked | unchecked | later | -- |

For the EQUALS and LOCATION commands, the expression for the value is evaluated when the assignment command is processed; the result of the evaluation is the value given to the name. However, for the SYNONYM and KEYIN commands, the value is saved as a sequence of characters. Later, when the name is used, 6Edit evaluates the saved sequence of characters.

EQUALS-Values and LOCATION-Values

The EQUALS and LOCATION commands operate similarly, with the exception of the type of the value they assign to a name. For the EQUALS command, the value is a character string (which may represent a number). For the LOCATION command, the value is the location of a block of data in a file.

Example:

```
START_POS EQUALS END_POS
```

is legal. END_POS is evaluated now.

```
ANSWER EQUALS 'ABC
```

is illegal, and will be rejected. The string has unbalanced quotes.

```
LOCATION CHAPTER_1 IS FILE MY_BOOK SKIP 3
```

is illegal, and will be rejected. The block_expression must be complete. It is evaluated now. The SKIP 3 option must be followed by a block operand. (See Section 5, Block Expressions in 6Edit, for details.)

### SYNONYM-Values

When processing a SYNONYM command, the fragment which is the value is lexically checked (parsed), and any comments are removed. However, no further processing of the fragment is performed. The value assigned to the name is simply the decommented text of this fragmented command line.

A name whose value is a SYNONYM-value can be used anywhere in any 6Edit command line, including the SYNONYM-value of a SYNONYM command. 6Edit substitutes the fragment which is its value for the name. 6Edit semantically checks the fragment only when the name is used, for example, when a fragment is substituted into a command line.

Example:

```
SYNONYM START_POS IS END_POS
```

is legal. END_POS will not be evaluated until START_POS is used.

```
SYNONYM ANSWER IS 'ABC
```

is illegal, and will be rejected. The string has unbalanced quotes.

```
SYNONYM CHAPTER_1 IS FILE MY_BOOK SKIP 3
```

is legal. The block_expression will not be evaluated until CHAPTER_1 is used in a command line.

### KEYIN-Values

Normally, values in assignment commands must at least be lexically correct (parsable into legal fragments). A lexically unchecked value (enclosed in string delimiters) is allowed in the KEYIN command only. When redefining a terminal key, it is useful to be able to generate any sequence of characters. A KEYIN-value, then, is any arbitrary sequence of characters; it may include displayable and non-displayable characters, control-characters, and input-editing functions recognized by the CP-6 system.

KEYIN-values are only useful when running 6Edit on-line. A name whose value is a KEYIN-value is recognized by the CP-6 system whenever you type the keystrokes which form the name. When the keystrokes are received by the system, the sequence of characters that represents the name's value is substituted for the keystrokes that comprise the name.

Note that 6Edit substitutes KEYIN-values for their names in any input from the terminal (command lines as well as data records). KEYIN-values are not substituted, however, in input from command files (initiated with the READ command).

Example:

```
KEYIN '%V(10)' IS  '''ABC'
```

is legal. When you type linefeed (ASCII 10), the characters

```
            'ABC
```

appear on the terminal.

## Copy and Move Operations

In 6Edit, you can copy or move a block of data from one location to another. This is done in two steps:

1. First, you specify the block you want to copy or move; this is called the "source block." The COPY and MOVE commands specify the source block for the operation.
2. Then, you specify the "destination" for the copy/move. Use the AFTER command to insert the source block at a location in a file. Or, use the OVER command to replace a destination block with the source block.

The COPY and MOVE commands do not make any changes to the file; they just specify the source block for the copy/move operation. Only when you enter an AFTER or OVER command is the copy/move operation performed.

Both the source and destination blocks can be any sequence of characters, records, or parts of records. 6Edit uses the same commands to copy/move one characters, an entire file, a sentence, or a few words.

When copying a block of data, 6Edit maintains any "record boundaries" in the block; that is, characters which were together in one record in the source block remain together in one record after the operation, at the destination location.

### Record Key Generation

In a copy/move operation, the source block may or may not include record boundaries. If the source block does include a record boundary, 6Edit must insert new records into the file at the destination location.

### Unkeyed Files

Records may not be inserted into consecutive files or unit record (UR) files except at the end of the file. 6Edit does not need to generate any record keys in this case.

### Edit-Keyed Files

If the destination file uses edit keys, new keys must be generated for any inserted records. 6Edit adds the value of the BY option to the key of the destination record to compute the new record keys. The BY option may be specified in the block_expression of the AFTER or OVER commands. If no BY option is given, 6Edit uses the value of the predefined name $BY.

If a key created with this increment conflicts with a key already in the file, 6Edit divides the increment by 10. This division is repeated until a new key, one which does not conflict with any already in the file, is created.

If no such key can be created, 6Edit tries to move some records nearby to make room for the insertion, governed by the $REKEY predefined name. If $REKEY is zero, 6Edit discontinues the copy/move operation, informs you of the problem, and no further processing of the source block, such as deletion for a MOVE command, occurs.

### String-Keyed Files

Records may only be inserted into a string-keyed file individually and by explicitly specifying the key to be used.

## Rekeying a Keyed File

When 6Edit tries to create a new record key and is not able to and $REKEY is zero, it will abort the operation. If the file is an edit-keyed file, you can rekey the file and resume the operation using the following command:

```
    COPY CURRENT OVER CURRENT
```

The synonym REKEY, defined in the standard context files, can be used instead:

```
    REKEY
```

This command can only be used when you are editing the entire file.

Optionally, you can add the BY option to control the record keys used when rekeying the file as follows:

```
    COPY CURRENT OVER CURRENT BY 5
```

or

```
REKEY BY 5
```

uses 5.0 as the first record key in the rekeyed file. Each successive record will have a key which is larger by five more increments than the previous key (i.e., the rekeyed file's keys will be 5.0, 10.0, 15.0, 20.0, ...).

# Section 2

# Using 6Edit

This section describes the environment which 6Edit requires for operation, the invocation of the 6Edit processor, and the user interface.

## DCBs

6Edit uses eleven DCBs. Before starting 6Edit, you can set some of the DCBs to other files or devices, using the IBEX SET command. The following list describes each DCB.

M$SI

can be set to a file containing 6Edit commands. 6Edit executes these commands after it reads the context file.

M$6E

can be set to the window which 6Edit is to use as the terminal screen. If it is not set, 6Edit uses all but the top line of UC01.

M$FILE

can be set to correspond to the file to be initially edited. You may also designate the initial file on the IBEX command line.

M$DO

receives error messages from 6Edit in non-interactive mode (BATCH or XEQ), or if set to a file.

M$EI, M$E9, M$6E00, M$6E01, M$EW

are reset (or "scrubbed") by 6Edit. You should not set them.

M$ME

receives information messages in non-interactive mode, and accepts command input in interactive but not full-screen mode (e.g., $SCREEN EQ 0).

M$LO

is used in non-interactive mode to echo commands if echoing is enabled (in IBEX), to display error messages, and to display information from the SHOW command. It is normally assigned to the command window, but may be redirected via the OUTPUT command.

## Context File

When 6Edit is invoked, it first looks for a "context file" containing 6Edit commands. These commands set up the environment in which to do your editing.

The file 6Edit looks for is called:

```
    :6EDIT_CONTEXT_profilename
```

where profilename is the name of your current terminal profile. If 6Edit cannot locate that name, it looks for:

```
    :6EDIT_CONTEXT
```

6Edit looks for these two file names first in your !DIR account, then in your logon account, and finally in .:LIBRARY.

6Edit's standard context file is :6EDIT_CONTEXT.:LIBRARY. It supplies common assignments. This context file is terminal independent, because it uses Escape sequences and control-keys for all functions, and does not assume that your terminal has any function keys. There are other context files, designed for use with specific terminal profiles, in the .:LIBRARY account.

As an alternative to these context files, you can build your own personal context file in your logon account. Because 6Edit reads the context file every time it is invoked, the context file should contain only those commands which you want executed every time you run 6Edit. This eliminates the need to retype them each time you invoke 6Edit.

For example, you may want to include KEYIN commands in the context file to redefine the keys of your terminal. This is particularly useful if your terminal has function keys or unlabeled keys: you can define them to generate any sequence of characters, commands and/or input editing functions you wish. (See Section 3, 6Edit Commands, for a discussion of the KEYIN command.)

You may also want to include SYNONYM commands in the context file. These define words or special characters to be synonyms for 6Edit keywords or phrases. (See Section 3, 6Edit Commands, for a discussion of the SYNONYM command.)

Before you alter the contents of a context file, you should copy the standard context file into your account, and then change it. (Appendix C, Context Files in 6Edit, provides examples of 6Edit context files. Appendix D, Customizing the 6Edit User Interface, provides some guidelines to consider before designing a context file.)

## Searching for a Context File

You can override the fixed name and directory search list which 6Edit uses to find a context file at start-up by using the IBEX LET command:

```
    !LET SETUP_6EDIT='fid'
```

where fid can be any of the following:

- o Only a file name (no account, packset name, or password can be given). 6Edit looks for just this file name in your !DIR account, then in your logon account, and finally in .:LIBRARY.
- o Only an account. 6Edit looks for the :6EDIT_CONTEXT_profilename file name, then for :6EDIT_CONTEXT. However, it looks for these file names only in the account specified in fid.
- o Both a file name (which may include an optional packset name and password) and an account. 6Edit looks for just that file name in only the specified account.

In the file name portion of the fid, you can use two special character strings:

>U - If this appears in the fid, it is replaced with the current user's name.

>P - If this appears in the fid, it is replaced with the current terminal profile name. (If 6Edit is running in batch mode, >P is simply removed from the fid.)

In both of these substitutions, an underscore (_) separates the substituted text from the rest of the fid.

The following table gives examples of different values of SETUP_6EDIT and files 6Edit searches for. The current terminal profile used is VIP7801, and the user name is EMC2EINSTEIN.

Table 2-1. 6Edit Setup Files

| If SETUP_6EDIT is set to: | then 6Edit looks for one of these files: |
|---|---|
| :6EDIT_CONTEXT>P | :6EDIT_CONTEXT.dir_account<br>:6EDIT_CONTEXT_VIP7801.dir_account<br>:6EDIT_CONTEXT.logon_account<br>:6EDIT_CONTEXT_VIP7801.logon_account<br>:6EDIT_CONTEXT.:LIBRARY<br>:6EDIT_CONTEXT_VIP7801.:LIBRARY |
| WP>U>P | WP_EMC2EINSTEIN.dir_account<br>WP_EMC2EINSTEIN_VIP7801.dir_account<br>WP_EMC2EINSTEIN.logon_account<br>WP_EMC2EINSTEIN_VIP7801.logon_account<br>WP_EMC2EINSTEIN.:LIBRARY<br>WP_EMC2EINSTEIN_VIP7801.:LIBRARY |
| 6E_DOCUM>P.EINSACCT | 6E_DOCUM.EINSACCT<br>6E_DOCUM_VIP7801.EINSACCT |
| .EINSACCT | :6EDIT_CONTEXT.EINSACCT<br>:6EDIT_CONTEXT_VIP7801.EINSACCT |

## Starting 6Edit

6Edit is entered from IBEX using the following command:

```
    !{6E[DIT]|6B[UILD]|6X} [fid] [(command list[)]]
```

Parameters:

fid    identifies the initial file to be edited.

command list    specifies one or more initial commands. (See Section 3, 6Edit Commands, for a command summary.) If a READ command is included in the command list, the trailing right parenthesis must be omitted.

Description:

Note that the context file is read and executed before the IBEX command line's command list is executed. This means, for example, that synonyms defined in the context file can be used in the command list.

If a command list includes more than one command, the commands must be separated by semicolons (;).

When started, 6Edit displays a greeting:

```
    6Edit A03 Here.
```

There is a delay while 6Edit reads the context file appropriate for this session, based on your terminal profile name and your current file management (DIR) account. If fid is included in the IBEX command line, 6Edit displays it after clearing the terminal screen. If fid is omitted from the IBEX command line, 6Edit prompts for a command. When waiting for a command to be entered, 6Edit displays an asterisk (*) on the terminal screen.

Examples:

```
    !6EDIT DAILY_JOB
```

or

```
    !6EDIT (FILE DAILY_JOB
```

or

```
    !6E DAILY JOB
```

starts 6Edit, editing the file DAILY_JOB, the beginning of which is displayed in the editing window.

```
    !6EDIT (NEW FILE WEEKLY_JOB.MYACCT
```

or

```
    !6BUILD WEEKLY_JOB.MYACCT
```

starts 6Edit, creating a new file named "WEEKLY_JOB" in account "MYACCT". The editing window is initially empty, ready for you to enter text into the file.

```
    !6EDIT (PROTECT FILE YEAR_END_JOB)
```

or

```
    !6X YEAR_END_JOB
```

starts 6Edit, editing the file YEAR_END_JOB. This file is "protected", that is, you cannot make changes to the file, you can only view the file in the editing window.

```
    !6EDIT DB_UPDATES (28.4)
```

starts 6Edit, editing file DB_UPDATES. The command list specifies that 6Edit is to put the cursor on the record with key 28.4. Instead of displaying the beginning of the file, 6Edit displays the records near record 28.4 in the editing window.

## Faster 6Edit Invocation

6Edit processes a file created by 6Edit's SAVE command much faster than a normal textual READ-file. To start 6Edit faster, you should first SAVE an encoded version of the context file you wish to use.

For example, if you plan to use the standard context file, run 6Edit and use the SAVE command as follows:

```
    !6EDIT
    * 6Edit A03 Here.
    * Reading :6EDIT_CONTEXT.someaccount.
    *SAVE SETUP TO :6EDIT_CONTEXT.youraccount
```

Now, the next time you start 6Edit, it will use the encoded version of the context file.

## Screen Appearance

6Edit typically divides the screen as illustrated in Figure 2-1 and explained in the following paragraphs. Additional capabilities affecting screen appearance -- multiple editing windows and serial editing -- are described later in this section.

6Edit normally divides the terminal screen into three "windows": the "IBEX window" in the top line of the screen, the "command window" in the upper half of the screen, and the "editing window" in the lower half of the screen. A straight line across the screen forms a "border" which separates the IBEX and command windows from the editing window.

```
---------------------
|   IBEX window      |
|                    |
|*  command window   |
|                    |
|                    |
|--------------------|
|                    |
|                    |
|   editing window   |
|                    |
|                    |
---------------------
```

Figure 2-1. The 6Edit Screen

You can edit the characters appearing in any window using the input editing functions. Input editing functions (entered at the keyboard) are key sequences beginning with the control (<CNTL>) or escape (<ESC>) key. Some input editing functions are introduced later in this section. (See Appendix E for a list of all input editing functions supported by 6Edit.)

## IBEX Window

The IBEX window is represented by one line above the command window at the top of your terminal screen. The IBEX window processes certain global commands when you are in 6Edit. It is an option.

To remove the IBEX window from your screen, enter the following command at the IBEX prompt before starting 6Edit:

```
    !SET M$6E UC01
```

or

```
    !ADJUST M$6E UC01
```

before any editing command.

If you remove the IBEX window, you have an extra line for editing. Also, since what you've really done is to combine the IBEX window and the command window, any commands that need to use the IBEX window will probably have more lines available. The disadvantage of combining the IBEX and command windows is that since keyins and IMPs are really the same thing, any keyins that you define in 6Edit for the command window will remain in the IBEX window after you exit 6Edit. You will have to restore any needed IMPs either manually or via the $RESTORE predefined name. This problem can be eliminated if you do not need to use keyins in the command window by specifying "IN EDITING" for all your keyin definitions.

If you do not remove the IBEX window, any IMPs or keyins used in your context file during that session are removed when you exit 6Edit. 6Edit restores your terminal environment to its former status.

By not removing the IBEX window, you give up screen space. Since the window uses a line at the top of your screen, you are short one editing window line at the bottom.

For example, if you normally define <ESC> <1> to issue a CHECK command, and you redefine Escape 1 during a 6Edit session to generate a COPY THAT command, then when you leave 6Edit:

- o If you had an IBEX window during the 6Edit session (the default), <ESC> <1> reverts back to generating the CHECK command.
- o If you did not have an IBEX window during the 6Edit session (i.e., you !SET M$6E UC01 before starting 6Edit), <ESC> <1> still generates the COPY THAT command. (See the $RESTORE name in Appendix A, Predefined Names.)

## Command Window

Lines at the top of your screen (above the border line) comprise the command window. The number of lines in this window changes depending on the window's usage. When the command window displays messages, it grows larger; when you type commands in this window, it remains small.

When the cursor is in the command window, you can type 6Edit commands. (See Section 3, 6Edit Commands, for details.)

You can control the minimum and maximum size of the command window by assigning values to the predefined names "$MIN_COMMAND" and "$MAX_COMMAND". (See Appendix A, Predefined Names.)

The stream name for the 6Edit command window is the lowest unused UC stream greater than UC09 (usually UC10).

## Editing Window

The editing window occupies the lower portion of your terminal screen (below the border line). It displays records from the file being edited.

Besides using the input editing functions, you can edit the data in the editing window using the editing commands. (See Section 3, 6Edit Commands.)

To move the cursor from the editing window to the command window, type <CNTL-C> (hold down the Control key, while typing the letter C once). To return to the editing window without typing a command, type <CR> on an empty command line. To type a command and leave the cursor in the command window, terminate the command with <CNTL-C>. These functions may differ according to your context file.

The stream name for the 6Edit editing window is the second lowest unused UC stream greater than UC09 (usually UC11). Additional editing windows start at the next unused UC stream (usually UC12 and higher).

## Vertical Scroll Margins

The editing window has two vertical "scroll margins." These margins are the top and bottom subportions of the editing window:

```
 -----------------------
|*                      |   command window
|                       |
|-----------------------|
|    scroll margin      |
|-----------------------|
|                       |   editing window
|                       |
|-----------------------|
|    scroll margin      |
 -----------------------
```

Figure 2-2. Vertical Scroll Margins

6Edit keeps the cursor out of the scroll margins whenever possible. (An exception is when the editing window displays the beginning or end of the file, so you cannot scroll any more in that direction.) Thus, text surrounds the cursor at all times. If you try to move the cursor into a scroll margin, 6Edit scrolls the data so the cursor is no longer in the scroll margin.

You can change the size of the scroll margin, from a minimum of zero lines to a maximum of half the editing window height. To do this, assign a number to the predefined name "$VSCROLL_MARGIN". (See Appendix A, Predefined Names.)

## Horizontal Scroll Margins

In addition to scrolling vertically, the 6Edit window can also scroll horizontally. Horizontal scroll margins act on the same principle as vertical scroll margins; however, horizontal margins scroll from right to left, or from left to right.

These margins are located at the right and left of the editing window:

```
-------------------------
|*                      |   command window
|                       |
|-----------------------|
| s  m |          | s  m |
| c  a |          | c  a |
| r  r |          | r  r |   editing window
| o  g |          | o  g |
| l  i |          | l  i |
| l  n |          | l  n |
-------------------------
```

Figure 2-3. Horizontal Scroll Margins

6Edit keeps the cursor out of the scroll margins whenever possible. If you try to move the cursor into the margin, 6Edit scrolls the data so that the cursor is no longer in the margin.

You can change the size of a horizontal scroll margin, from a minimum of zero columns to a maximum of half the editing window width. To do this, assign a number to the predefined name "$HSCROLL_MARGIN". (See Appendix A, Predefined Names.)

## Wordwrap

6Edit supports a special feature called "wordwrap." As you type text, 6Edit allows you to type as far as the right margin; when you cross that margin, wordwrap automatically replaces the last word of the record at the beginning of a new record. Thus you can enter text without ever pressing the Return key.

You can enact the wordwrap mode with the $WORDWRAP predefined name. (See Appendix A, Predefined Names.) The value assigned to $WORDWRAP is the right margin column number you choose. A value greater than 11 implements the wordwrap mode. To discontinue the wordwrap mode, set $WORDWRAP to 0 or 1; this is the default.

## Multiple Editing Windows

Multiple editing windows are permitted. The WINDOW commands, explained in Section 3, allow you to create and remove editing windows. (There will always be just one command window.) At any one time, you're working with just one editing window, called the "current" editing window.

Each editing window remembers its own instances of:

the edit block ("CURRENT");
the edit block stack (for "EDIT PREVIOUS" and "EDIT NEXT");
the file pointer ("HERE");
the selected block ("THAT").

Different editing windows can be editing the same file or different files. You can select a block in one window, switch to a different editing window, and insert the data in the new window.

The editing windows must be created either side-by-side (horizontal windows), or stacked one above the other (vertical windows). The horizontal and vertical modes of window creation are illustrated in the following figure.

```
  ----------------------------     ----------------------------
  | command window           |     ! command window           |
  |--------------------------|     |--------------------------|
  !   ed.  | ed.  | editing  |     | editing window #1        |
  |   win. | win. | window   |     |--------------------------|
  |   #1   | #2   | #3       |     | editing window #2        |
  |        |      |          |     |                          |
  ----------------------------     ----------------------------

  Horizontal editing windows         Vertical editing windows
```

Figure 2-4. Two Styles of Multiple Editing Windows

The "mode" for creating new editing windows (horizontal or vertical) is determined when the second editing window is created. To switch "modes", you must remove all but one editing window, and create a second editing window in the desired mode. Only vertical mode is implemented in A03 6Edit, however.

The minimum window size is: for horizontal windows, ten positions; and for vertical windows, one line plus the border, if present. The only practical limit on the number of windows which can be created is the size of the 6Edit screen.

Editing windows are numbered starting at one, top-to-bottom or left-to-right. The predefined name $WI_BORDER controls borders between windows. The value of $WI_BORDER is a single character, or no characters (the empty string); the initial value is "-". This character will be used to form the border between windows; if $WI_BORDER is set to the empty string, then borders are not used.

To move the cursor from the current editing window to the next one, type <CNTL-N> (hold down the Control key, while typing the letter N once). The cursor can be in the current editing window or the command window. If the current editing window is the last one, <CNTL-N> moves to the first one.

## Input Editing Functions

Generally, input editing functions affect characters under or near the cursor by moving, deleting, etc.

These key sequences are those in effect at logon, before any new IMP keys are designated. You, the user, may redefine these keys on your terminal keyboard. (See Appendix D, Customizing the 6Edit User Interface, for more information.) If you decide to redefine the standard key sequences, those listed here (and in Appendix E) may not work as described.

Input editing functions may be typed at the keyboard directly, by typing a sequence of one, two, or three keys.

The following paragraphs describe the key sequences you can use to move the cursor, replace, insert, and delete characters, and split and join records. These input editing functions are executed by the Front End Processor. (See Appendix E for a complete list of input editing functions.)

### Moving the Cursor

The following key functions move the cursor (and, when necessary, scroll the data in the editing window). These functions do not change the data in the file.

Table 2-2. Cursor Key Functions

| Command | Function and Description |
|---|---|
| <BS> or <CNTL-H> | Backspace. |
| <CNTL-R> | Forward space. |

Table 2-2. Cursor Key Functions (cont)

| Command | Function and Description |
|---|---|
| <TAB> or <CNTL-I> | Tab. |
| <CNTL-W> | Move to right/left word.<br>Moves the cursor to the right or left word. The direction in which the cursor moves depends on the last <BS> or <CNTL-R> key typed.<br><br>A "word" represents a contiguous string of non-space characters. Any number of spaces may appear between words. All spaces are skipped when moving to the next word. If the recordwrap mode is disabled, this key function is limited to the current record; when you reach the beginning or end of the record, this function changes direction and moves to the next word in the opposite direction. If the recordwrap mode is enabled, this key function will move to the next or previous record when it hits the edges of the current record. (See the $RECORDWRAP predefined name in Appendix A, Predefined Names.)<br><br>You can use the KEYIN command to define keys that can move either left or right to the previous or next word. (See the examples for the KEYIN command in Section 3.) |
| <ESC> <CR> | Move cursor to beginning of current record. |
| <ESC> <N> | Move cursor to the end of current record. |

Table 2-2. Cursor Key Functions (cont)

| Command | Function and Description |
|---|---|
| <CR> | Move cursor to beginning of next record. If the cursor is in the command window, <CR> returns the cursor to the current editing window. |
| <ESC> <A> | Move cursor up one record. |
| <ESC> <B> | Move cursor down one record. For <CR>, <ESC> <A>, and <ESC> <B>, the cursor moves by records, not by lines on your terminal. If a record occupies more than one line of the display, "move cursor up" moves the cursor to the same position of the previous record, (not the previous line). This movement may appear to jump over terminal lines. The same is true of "move cursor to beginning of next record" and "move cursor down". |
| <CNTL-C> | Moves the cursor to the command window. This function may also be used to terminate a command and keep the cursor in the command window. |
| <CNTL-N> | Moves the cursor from the current editing window to the next one. If the current editing window is the last one, it moves the cursor to the first one. From the command window, <CNTL-N> is equivalent to <CR><CNTL-N>. |

The context file that 6Edit reads initially may have defined other keys on your terminal keyboard, to move the cursor over groups of records, or to search the file for specific data.

## Replacing and Inserting Characters

There are two modes available for typing data in 6Edit: the replacement mode and the insertion mode. (There is another mode, the overstrike mode, but its use is more specialized, and is not described here.) These modes are available whenever you use CP-6; they are described here only because they are particularly helpful when using 6Edit. The following paragraphs provide an introduction to these modes. (For detailed information on their use, see the CP-6 Programmer Reference (CE40), Section 6, Terminal Control.)

You may do all your editing in either mode, or you may switch between modes. The technique you use is a matter of personal preference as well as the nature of your editing.

### Replacement Mode

When you logon to CP-6, you are initially in the replacement mode. In this mode, each character you type replaces the character under the cursor. You can insert characters into a record at the end of the record. To insert characters anywhere else in the record, you must define an "insertion window" by typing <ESC> <J>, or <ESC> <>>; then, characters you type at the right end of the insertion window are inserted into the record at that point.

The insertion window "encloses" the action of many input editing functions. For example, the <ESC> <CR> sequence moves the cursor to the beginning of the insertion window. If you move the cursor beyond the insertion window, the window expands, encompassing the entire record.

### Insertion Mode

As an alternative to the replacement mode, you can use the insertion mode. To tell the system tc use the insertion mode, enter <ESC> <'>. Later, to return to the replacement mode, enter <ESC> <M>. Or use the $INSERT predefined name.

The insertion mode inserts characters into the record; it never replaces characters. To replace characters in the insertion mode, you must first delete the characters you want replaced, then insert the new characters.

If you define an insertion window in the insertion mode, it maintains many of the same effects as in the replacement mode. For example, many of the input editing functions are limited to the insertion window. However, regardless of whether the cursor is at the end of the insertion window, all characters typed are inserted into the record, they never replace characters already there.

## Deleting Characters

You can delete characters in several ways.

Table 2-3. Delete Functions

| Command | Function and Description |
|---|---|
| <DEL> | Delete character. Usually deletes the character under the cursor. However, in the replacement mode only, if the cursor is at the end of the insertion window, <DEL> deletes the character to the left of the cursor. |
| <ESC> <DEL> | Delete left character. Always deletes the character to the left of the cursor. |
| <ESC> <CNTL-K> | Delete characters from the beginning of the record up to the character just to the left of the cursor. |
| <ESC> <K> | Delete the character under the cursor and all characters to the right to the end of the record. |
| <ESC> <CNTL-L> | Delete the record from the file. This is the same as <ESC> <X>, but it also deletes the record from the file. |

Table 2-3. Delete Functions (cont)

| Command | Function and Description |
|---|---|
| <ESC> <CNTL-W> | Delete word.<br>Deletes the character under the cursor, and all characters to the right up to the start of the next word.<br><br>You can use the KEYIN command to define keys that can delete either the previous or the next word. (See the examples for the KEYIN command in Section 3.) |
| <ESC> <X> | Delete the entire record.<br>This does not remove the record key from the file, but does delete all characters in the record. |
| <CNTL-X> | Same as <ESC> <X>. |

## Splitting and Joining Records

Splitting records allows you to insert blank lines, and add new records to the file. Type <ESC> <LF> to "split" a record, i.e. break it up into two separate records. The characters under and to the right of the cursor are removed from the current record and placed in a record which is inserted into the file following the current record. To insert new records in the middle of a file, type <ESC> <N> to go to the end of the record, and then type <ESC> <LF>. This creates a new record after the initial record.

Another way to insert records in the middle of a file is from the beginning of the record. Type <ESC> <CR> to go to the beginning of the record. Then type <ESC> <LF>. This creates a new record at the cursor line.

To add data at the end of a file, go to the end of the file, using the EO CURR command in the command window. This command positions the cursor on a blank line. Type the new record. Then type either <CR> or <ESC> <LF>.

Using <ESC> <BS>, you can join two records together into one record. This function works only when the cursor is at the beginning or the end of the record. If the cursor is at the beginning of the record, it joins this record to the previous record. If the cursor is at the end of the record, the following record attaches to the last character of the cursor record when you type <ESC> <BS>.

The presence of a blank space between words when joining or splitting lines is controlled by the $TEXTEDIT predefined name. See Appendix A for details.

## Serial Editing

6Edit provides the serial editing mode for circumstances when full-screen editing is inappropriate or inefficient. These cases are as follows:

o When using 6Edit non-interactively (i.e., as a result of the IBEX BATCH or XEQ commands)

o When using 6Edit online at a non-CRT terminal that permits neither full screen display nor scrolling

o When using 6Edit to perform well-defined editing tasks such that it is more efficient not to wait for the editing window to be filled with a full screen of data every time the file pointer is pointed elsewhere within the file.

Note that in serial editing, window-specific values for EQUALS-names and KEYINs are ignored, as is the size and position of a new window created with the WINDOW command.

When serial editing, a block expression by itself as a command (with no editing verb such as COPY or AFTER) moves the file pointer without displaying any data. To display data, an explicit DISPLAY command is necessary.

### Switching between Serial and Screen Editing

When 6Edit is started, it determines if the user is running online and if the terminal can support screen editing. If the user is not online or if the terminal profile reveals that the terminal cannot screen edit, 6Edit forces the user into serial editing mode. That is, it sets the predefined name $SCREEN to 0 (the height of the terminal "screen").

The online CRT user choose between serial editing and screen editing by setting $SCREEN back to the number of lines that 6Edit should use as the screen size. Setting $SCREEN to ON (or 1) restores the current maximum screen size allowed. Setting $SCREEN to OFF (or 0) enters serial mode.

## Manipulating Data in Serial Editing Mode

In serial editing mode, with $COMMAND set on (the default), 6Edit commands are used to manipulate the data. With $COMMAND off, 6Edit prompts with the key of the "here" record, and uses reread mode for editing the data.

Special activation characters perform special functions:

<LF> terminates editing of the current record, and moves to the next one. <LF> is <CNTL-J>.

<EOT> terminates editing of the current record, and moves to the previous one. <EOT> is <CNTL-D>.

<SYN> splits the current record at the cursor position, and possibly moves to the new one, depending on whether it is empty. If wordwrap is set, typing at the end, past the wordwrap position, will cause the insertion of a new record just as it does in screen editing mode. <SYN> is <CNTL-V>.

Other activation characters (i.e., all normal ones plus <ETX>) cause a command prompt, just as in screen editing.

# Section 3

# 6Edit Commands

This section describes in detail each of the commands supported by 6Edit.

## Entering Commands When Full-Screen Editing

When you are full-screen editing in 6Edit, your terminal's display screen is divided into two "windows." Commands for 6Edit may only be entered when the cursor is in the upper window, which is called the "command window." If the cursor is in the lower window, you can move it to the command window by typing <CNTL-C> on your terminal. Hold down the Control key, as if shifting, then type the "C" key. This keystroke may differ according to your context file. (See Appendix C, Context Files in 6Edit, for standard key definitions.)

When the cursor is in the command window of the screen, you can enter command lines normally. You can use all the features described in this section.

## Command Summary

There are three categories of commands in 6Edit: editing, assignment, and housekeeping. The general syntax for all 6Edit commands is shown in Table 3-1.

Table 3-1. Command Summary

| Type | Syntax | Description |
|---|---|---|
| Editing Commands | | |
| | 6E[DIT] file | Begin editing (by displaying the specified block) |
| | 6B[UILD] file | Create a new file |
| | 6X file | Edit block in protected mode |
| | ED[IT] block | Begin editing (sets edit block) |
| | [DI[SPLAY]] block | Display a block |
| | CO[PY] {block\|string} | Set source for copy operation |

Table 3-1. Command Summary (cont)

| Type | Syntax | Description |
|---|---|---|
| | MO[VE] block | Set source for copy; delete after copy |
| | AF[TER] block | Copy source, inserting after block |
| | OV[ER] block | Copy source, replacing block |
| | DE[LETE] block | Delete a block |
| Assignment Commands | | |
| | name EQ[UAL[S]] string | Assign string value to name |
| | LO[CATION] name [IS] block | Make name refer to block |
| | SY[NONYM] name [IS] fragment | Make name a synonym for fragment |
| | KE[YIN] str_1 IS str_2 | Redefine a terminal key |
| Housekeeping Commands | | |
| | TIME | Display current date and time |
| | DA[TE] | Display current date and time |
| | DIR[ECTORY] [fid\|R[ESET]] | Change default account and packset name |
| | {DO\|!} command | Call another processor |
| | H[ELP] [(processor)] [topic] | Display specified on-line information |
| | IF str [cd];cds[;ELSE [cd];cds] | Conditionally execute commands. |
| | READ fid | Read 6Edit commands from fid |
| | OUT[PUT] [location] [optlist] | Send output to specified location |
| | PRINT [ALL\|ldevlist] | Direct accumulated output to destination |
| | ERASE [ALL\|ldevlist] | Delete specified output |
| | EN[D]\|X[IT]\|Q[UIT] | Exit 6Edit |
| | REST[ORE] fid | Restore context information from fid |
| | SAVE [option] {TO\|ON\|OVER} fid | Save current context information |
| | SH[OW] option | Displays strings, assigned names, records in a block, keyin definitions, and windows |
| | [function] WI[NDOW] [option] | Creates a new editing window, changes current editing window, or switches to another editing window |

## AFTER Command

Syntax:

```
AF[TER] [BL[OCK]] block_expression
```

Parameters:

block_expression    specifies the location at which to insert the source block.

Description:

The AFTER command is used to insert a block of data into a file. The block_expression specifies the destination location for the insertion operation. The current source block is inserted into a location just after the end of the block specified by block_expression. (The source block was set by the last COPY or MOVE command.)

The source and destination blocks may be whole records, parts of records, or any combination of these.

Example:

```
COPY 1 THRU 5 AFTER 10
```

inserts a copy of records 1 through 5 after record 10.

```
COPY 1 THRU 5 AFTER FILE OCTOBER_BILLS 42
```

copies records 1 through 5 to record 42 of the specified file.

Related Topics:

Copy/Move Operations
COPY Command
MOVE Command
OVER Command

## COPY Command

Syntax:

```
CO[PY] {[BL[OCK]] block_expression}
       {ST[RING] string_expression}
```

Parameters:

| | |
|---|---|
| block_expression | is the block of data to be copied. |
| string_expression | is the string of characters to be copied. |

Description:

The COPY command specifies the source block for the next AFTER or OVER command.

The COPY command is a passive command. To have any effect, it must be followed by an AFTER or OVER command.

6Edit remembers the location of the source block for use later by the AFTER and OVER commands.

If block_expression is given, then 6Edit simply remembers the location of the block specified, it does not keep a separate copy of the data. If string_expression is given, then 6Edit keeps a copy of the value of string_expression for later use by the AFTER and OVER commands.

Example:

```
COPY 2 PO 10 THRU EO 4
```

specifies a source block which begins at position 10 of record 2, and ends at the end of record 4. 6Edit ignores this data until it receives the next AFTER or OVER command.

```
COPY ST 'X' OVER REPEAT SELECT 'Y'
```

replaces all 'Y's with 'X's. This command searches for 'Y's starting at the current file pointer location, and proceeds to the end of the edit block.

Related Topics:

Copy/Move Operations
MOVE Command
AFTER Command
OVER Command

## DATE Command

Syntax:

```
DATE
```

Parameters:

None

Description:

DATE (a synonym for the TIME command) displays the current date and time.

Example:

```
DATE
```

requests a display of the current data and time. A sample display is:

```
MAY 22  '81  14:39
```

Related Commands:

TIME

## DELETE Command

Syntax:

```
DE[LETE] [BL[OCK]] block_expression
```

Parameters:

| | |
|---|---|
| block_expression | is the block of data to be deleted. |

Description:

The DELETE command deletes its operand. After deleting the block, the location vacated by the block becomes the new selected block. Note that this is an "empty" block: it contains no data characters.

Note that "block_expression" can explicitly specify an entire file (e.g., "FILE filename"). In this case, the file itself is deleted, not just the records in it.

If a file is open for editing when DELETE FILE is entered, the editing window is cleared, and the following messages appear in the command window:

```
* File filename deleted.
* EDIT PREVIOUS will restore the Edit Block.
```

If the file open for editing was the one deleted and no other files are in the edit block stack, the EDIT PREVIOUS message is not issued. A deleted file cannot be restored by 6Edit.

If the file open for editing was a file other than the one deleted, then entering EDIT PREVIOUS will return that file to the screen.

Example:

```
DELETE 5
```

means delete record 5.

```
DELETE BO CURRENT THRU 5 PO 20
```

means delete everything from the beginning of the current edit block through record 5 position 20.

```
DELETE FILE TRANS
```

means delete the file TRANS and all the records in it.

## DIRECTORY Command

Syntax:

```
DIR[ECTORY] [fid|R[ESET]]
```

Parameters:

| | |
|---|---|
| fid | is a fid containing only an account name, and an optional packset name. |

Description:

The DIRECTORY command changes the default account and packset for fids. These defaults are used if a fid in any later command does not include an account and packset name. If neither the fid nor the RESET option is included, 6Edit simply displays the current directory.

The RESET keyword specifies that the default account and packset name are to be reset to the user defaults (i.e., the defaults in effect when you logged on to the system).

Example:

```
DIR .SYSLIB
```

directs subsequent fids that do not include an account to default to the .SYSLIB account and to the packset associated with that account.

## DISPLAY Command

Syntax:

```
[DI[SPLAY]]  [BL[OCK]] block_expression
```

Parameters:

| | |
|---|---|
| block_expression | is the block of data to be displayed. |

Description:

The DISPLAY command displays the value of its operand.

DISPLAY SKIP 20 RECORDS

means move the cursor to the record which is 20 records beyond the current cursor record. A section of the file surrounding this record is displayed in the editing window.

SKIP 20 RECORDS

performs the same function as the above command, except that 6Edit does not forget the block selected before the command; it simply changes the block's boundaries.

DISPLAY 428.5

means move the cursor to the record in the current edit block with key 428.5.

The verb DISPLAY may be omitted from the command line. However, there is a variation in the command's effect: when the verb DISPLAY is given, 6Edit forgets any block selected prior to the command, and the block_expression in the DISPLAY command becomes the new selected block.

On the other hand, when the verb DISPLAY is omitted, block_expression only changes part of the specification of the block being selected (if any) prior to this command. For example:

| Form | Result |
|---|---|
| 5 | Moves the cursor to record 5.0 |
| THRU | Begins selecting a block |
| 10 | The selected block is now 5.0 THRU 10.0 |
| SKIP 3 RECORDS | The selected block is now 5.0 THRU the third record after 10.0 |

Compare the above sequence of commands with the following:

|  |  |
|---|---|
| 5 | Moves the cursor to record 5.0 |
| THRU | Begins selecting a block |
| 10 | The selected block is now 5.0 THRU 10.0 |
| DI SKIP 3 RECORDS | Forgets the former selected block; the selected block is now the third record after record 10.0 |

In serial editing mode, the DISPLAY command displays only the complete records constituting the selected block (through M$LO), and only if the verb DISPLAY is used explicitly.

## DO Command

Syntax:

{DO|!} command

Parameters:

| | |
|---|---|
| command | specifies an IBEX-level command. |

Description:

The DO command is used to call another processor from within 6Edit.

## EDIT Command

Syntax:

ED[IT] [BL[OCK]] block_expression

Parameters:

| | |
|---|---|
| block_expression | is the block of data to be edited. |

Description:

The EDIT command causes its operand to become the new "edit block." The edit block is used by 6Edit in block expressions; it limits pattern searches, and is the implicit file in which you move the file pointer. (See Section 5, Block Expressions in 6Edit.)

The previous edit block is not forgotten. Its specification remains in the edit block stack.

The block_expression is not required to specify an entire file; however, the block it specifies must begin and end on record boundaries, that is, block_expression must specify some number of complete records.

Sometimes it is convenient to limit file pointer movement to some sub-portion of the file. EDITing a sub-portion of a file also limits the display of data around the file pointer, which can be useful when your terminal operates at a low speed (thus displaying data takes longer).

The predefined name "CURRENT" is set by 6Edit automatically whenever the EDIT command is given. The value of "CURRENT" is the location of the current edit block. For example, the block expression "EO CURRENT" moves the file pointer to the end of the current edit block.

You use the EDIT command when creating a new file. Type the command

```
    EDIT NEW FILE fid
```

at the asterisk prompt.

Example:

```
EDIT FILE OPPORTUNITY_KNOCKS
```

begins editing the file "OPPORTUNITY_KNOCKS". An entry for this new edit block is added to the edit block stack.

```
EDIT NEW FILE ACCEPTANCE_LETTER
```

creates a new file "ACCEPTANCE_LETTER", and begins editing it. An entry for this new edit block is added to the edit block stack.

```
EDIT 50 THRU 15286
```

edits a sub-portion of the current edit block. Specifically, the new edit block starts at record 50.0 and continues through (and including) record 15286.0. An entry for this new edit block is added to the edit block stack.

```
EDIT PREVIOUS
```

reverts to editing the block whose specification was saved in the edit block stack immediately prior to the current edit block. The edit block stack is not altered, but the "CURRENT" entry changes.

Related Topics:

Edit Block
Edit Block Stack

## END, EXIT, QUIT and XIT Commands

Syntax:

```
{E[ND]|EX[IT]|Q[UIT]|X[IT]}
```

Parameters:

None

Description:

This command terminates 6Edit. Control is returned to the command processor (IBEX).

The keywords, END, EXIT, QUIT, and XIT perform exactly the same function. All editing changes you made during an editing session take effect when you change the record. Therefore, you can terminate 6Edit at any time without losing any of the editing work you have done.

However, any name settings (such as KEYINS, SYNONYMS, EQUALS, and LOCATION names) that you made during the session will be lost when you type the END command and leave 6Edit.

## EQUALS Command

Syntax:

```
                                 [   {CO[MMAND] [WI[NDOW]] }]
name EQ[UALS] string_expression [IN {ED[ITING] [WI[NDOWS]]}]
                                 [   {WI[NDOW] window_num  }]
```

Parameters:

name    is any name; it must be between 1 and 31 characters in length. The "name" may include:

- o Alphabetic characters
- o Digits (except as the first character of name)
- o $ _ # and @

string_expression    is any string expression.

window_num    is a number between 1 and the number of editing windows.

Description:

The EQUALS command assigns the value of string_expression to name. The name may be a predefined name, a name you previously created with an assignment command, or a new name you are creating now.

When you type an EQUALS command, 6Edit evaluates string_expression to yield a string of characters. (If numbers were used or computed in string_expression, 6Edit converts them to character form using decimal representation.) 6Edit assigns this character string to name. Thereafter, name can be used in string or block expressions; 6Edit replaces it with the character string which was assigned to it by this EQUALS command.
Note: Except when name is $INTRO_1 or $INTRO_2, $CONTROL functions in the character string are not interpreted when assigned but only when the resulting variable is used in a different command.

The value of name will not change until you explicitly assign a new value to it. (Certain predefined names disobey this rule; 6Edit updates their values internally at certain points in the processing of commands.)

The IN-clause can be appended to the EQUALS command only when name is one of the following predefined names:

| | |
|---|---|
| $AUTOTAB | $INTRO_2 |
| $END_MARK | $KEY_GENERATION |
| $EZ_APPEND | $RECORDWRAP |
| $HSALL | $SCROLL |
| $HSCROLL | $TEXTEDIT |
| $HSCROLL_MARGIN | $VSCROLL |
| $INSERT | $VSCROLL_MARGIN |
| $INTRO_1 | $WORDWRAP |

If you include the IN-clause when assigning a value to one of these predefined names, then the value affects only the command or editing window, depending on which is specified. If you omit the IN-clause when assigning a value to one of these predefined names, then the value affects both the command and editing windows. Note that $SCROLL works only in the command window. "IN EDITING WINDOW" causes all editing windows to be affected. "IN WINDOW window_num" affects only the specified window.

For details on the predefined names listed above, see Appendix A, Predefined Names.

Example:

```
MODEL EQUALS MODEL .PLUS. 3
```

increments the value of MODEL (a hypothetical name you have already defined) by 3.

```
$WORDWRAP EQUALS 76
```

sets the value of the predefined name $WORDWRAP to 76 (i.e., enables wordwrap mode, limiting lines to 76 characters). See Appendix A, Predefined Names, for details on the $WORDWRAP name and wordwrap mode.

```
$HSCROLL_MARGIN EQUALS 5 IN COMMAND WINDOW
```

enables horizontal scrolling in the command window only. It does not affect horizontal scrolling in the editing window.

```
$INTRO_1 EQUALS $KEY(F4)
```

sets the primary introducer for KEYIN names to $KEY(F4). This permits definitions of keyins such as KEYIN $KEY(F4)||'R' IS ... to establish the actions taken when you type F4 followed by R. See the KEYIN command, later in this section, for details on the use of KEYIN and the $KEY(keyname) predefined string function.

Related Topics:

Substitution

## ERASE Command

Syntax:

```
ERASE [ALL|ldevlist]
```

Parameters:

ALL specifies that the accumulated outputs for all logical devices are to be deleted. This is the default.

ldevlist specifies that the accumulated outputs for the specified logical device or devices are to be deleted. The list is entered in the format

```
    ldevname[,ldevname]...
```

ldevname is a logical device name established through the LDEV command.

Description:

ERASE deletes the accumulated output for logical devices.

Example:

```
ERASE ALL
```

deletes all output accumulated for all logical devices defined for the session or job.

Related Commands:

PRINT

## HELP Command

Format:

```
H[ELP] [(fid)] [TOPICS] [keyword1 [-] [keyword2]
```

Note: The following elements can be specified in any order:

```
(fid)
TOPICS
[keyword1] [-] [keyword2]
```

For example, HELP (fid) keyword1 - keyword2 TOPICS is acceptable.

Parameters:

(fid) specifies the processor name (for example, 6EDIT). If (fid) is omitted, the current processor is assumed.

TOPICS requests a list of topic or subtopic names, rather than an information message.

| Form | Result |
|---|---|
| HELP (fid) TOPICS | Lists all topics |
| HELP (fid) TOPICS keyword1 - keyword2 | Lists all topics in the range specified by keyword1 - keyword 2 |
| HELP (fid) TOPICS keyword1? | Lists all topics beginning with the prefix specified by keyword1 |
| HELP (fid) TOPICS keyword1 | Lists all subtopics for the topic specified by keyword1 |

keyword1 [- [keyword2]]    specifies a topic, a range of topics, or a topic and subtopic to identify what HELP information is requested.

| Form | Result |
|---|---|
| HELP (fid) keyword1 | Displays the first level information message for the topic keyword1 |
| HELP (fid) keyword1 keyword2 | Displays the information message for keyword1, but only the level identified by the subtopic keyword2 |
| HELP (fid) TOPICS keyword1 [-] [keyword2] | Lists topic or subtopic names. See TOPICS parameter. |

keyword1 may include the wildcard (?) character as the rightmost character, if TOPICS is specified.

Description:

HELP displays information.

HELP messages have levels. Once the initial level has been displayed, entering a question mark displays the next level, usually containing greater detail. Entering two question marks displays the entire message.

The standard HELP command is provided by most processors. The HELP facility for each processor is available only to users with sufficient privilege to use the processor itself. The HELP command can be used either within the processor or in IBEX.

## IF and ELSE Commands

Format:

```
IF string_expr [command] ; commands [; ELSE [command] ; commands]
```

Parameters:

string_expr specifies an expression that evaluates to a number.

command specifies any 6Edit command.

commands specifies any 6Edit commands.

Description:

The IF and ELSE commands permit conditional execution of multiple commands in a single command line. If the string expression evaluates to a number greater than zero, any commands preceding the corresponding ELSE are executed, and the rest of the line (ELSE and beyond) is ignored. If the string expression evaluates to zero, the commands following ELSE are executed.

The IF/ELSE construct can be nested to any depth.

Examples:

```
IF $AUTOTAB PO NU $AUTOTAB; ELSE PO NU 1
```

positions to the autotab column of the current record.

```
RE 'bird'; IF $MATCHES COPY STRING 'bath' AFTER BO '%R'
```

appends the word "bath" to the next record containing the word "bird".

## KEYIN Command

Syntax:

```
KE[YIN] string_expression_1 IS string_expression_2

    [   {CO[MMAND] [WI[NDOW]] }]
    [IN {ED[ITING] [WI[NDOWS]]}]
    [   {WI[NDOW] window_num  }]
```

Parameters:

string_expression_1   specifies the character or character sequence to be typed on the terminal keyboard. It must evaluate to one, two, or three characters, or a function key name possibly followed by one character. Additional rules governing string_expression_1 are explained under "Description" below.

string_expression_2   specifies the characters to be substituted for string_expression_1 when the latter is typed at the terminal keyboard. Information on expressing non-displayable characters in string_expression_2 are explained under "Description" below.

window_num   is a window number between 1 and the number of editing windows.

Description:

The KEYIN command redefines the keys on your terminal. It is only effective when 6Edit is being used in the screen-editing mode, and the command stream is originating from your terminal. 6Edit uses the CP-6 IMP facility to perform substitution.

After using this command, whenever the CP-6 system sees the string_expression_1 character(s) coming from your terminal, it acts as though you had typed string_expression_2 instead. The system even displays string_expression_2 on your terminal, instead of the characters you had typed (that is, instead of string_expression_1).

Any character value may appear in either string_expression_1 or string_expression_2, including non-displayable characters. In string_expression_1, non-displayable characters represent control keys; in string_expression_2, they represent input editing functions. This distinction is important. For example:

> 18 is the ASCII decimal code for <CNTL-R>. When "%V(18)" appears in string_expression_1, it represents the <CNTL-R> key on the keyboard, the meaning of which you can define using the KEYIN command. However, when "%V(18)" appears in string_expression_2, it always represents the "move right one character" input editing function, no matter how you have redefined <CNTL-R>. (See Appendix E for a list of input editing functions.)

The IN-clause can be appended to any KEYIN command. If you include the IN-clause, the string_expression_1 key or key sequence is defined only in the command or editing window, depending on which is specified. If you omit the IN-clause, then string_expression_1 is defined in both the command and editing windows.

NOTE: Typing an immediate-type function before 6Edit has IMPed the key sequence to a typeahead-type function invokes an immediate-type function rather than the 6Edit definition of the key sequence.

This situation occurs only when using immediate-type input editing functions. The immediate-type input editing functions are:

| ESC Sequences | | Control Characters |
|---|---|---|
| <ESC><A> | <ESC> <CNTL-A> | <CNTL-Q> |
| <ESC><B> | <ESC> <CNTL-B> | <CNTL-S> |
| <ESC><G> | <ESC> <ESC> | <CNTL-X> |
| <ESC><H> | | <CNTL-Y> |
| <ESC><Q> | | |
| <ESC><W> | | |
| <ESC><Y> | | |

To avoid this situation, use FCNTBL=CP5S2 with the IBEX TERMINAL command before invoking 6Edit.

Defining a key sequence to be immediate-type in one window but typeahead-type in another window may give unpredictable results. Either key sequence definition could be used by the FEP, depending on where 6Edit is in its processing of the last terminal input.

Several rules apply to string_expression_1.

o If string_expression_1 evaluates to exactly one token (character or function key name), it may be any token value which can be generated at your terminal keyboard.

o If string_expression_1 evaluates to two tokens the first must equal either the "primary introducer" or the "secondary introducer." These one- or two-token values are set by the predefined names "$INTRO_1" and "$INTRO_2"; initially, $INTRO_1 is the Escape character and $INTRO_2 has no value. (See Appendix A, Predefined Names. Also, refer to the CP-6 Programmer Reference Manual (CE40) Section IMP, for an explanation of introducers.)

In addition, the following rules apply. If string_expression_1 is

o 1 token long: string_expression_1 need not match either $INTRO_1 or $INTRO_2.

o 2 tokens long: the first character must equal either $INTRO_1 or $INTRO_2 and the second must be a character (not a function key name).

- 3 tokens long: the first two characters must equal $INTRO_1 or $INTRO_2. Only the first can be a function key name.

The last or only character of string_expression_1 may be any character value which can be generated at your terminal keyboard.

Non-displayable tokens may be included in either string_expression_1 or string_expression_2. You can enter these tokens in string_expression_1 or string_expression_2 in one of these ways:

- Using the $KEY(keyname) predefined string function or its equivalent notation: <keyname>. The parameter for this function is a keyname, such as ESC. For example, the F1 function key may be represented as $KEY(F1) or as <F1>. To obtain the spelling for any non-displayable key, use the IMP command SPELL.

  A unique feature of the $KEY predefined string function is that it can be used to represent a function key that sends one, two, three, or even four characters. These keys may be functions keys such as F1 or SF1 (i.e., Shift F1). For PCs operating with PCT, $KEY may be used to assign meaning to Alt-Fn or Cntl-Fn keys (see Appendix C and D for details).

- Using the $CNTL('char') predefined string function or its equivalent notation: <CNTL-char>. The parameter for this function, "char", is any character which when pressed with the CNTL key sends a control character.

- Using the $CONTROL-Value function. The parameter for the function is the ASCII decimal code for the control key (e.g., <CNTL-D>), or the ASCII control character (e.g., <ESC>) that you are defining. (See Appendix F for the ASCII names, their control characters, and their decimal codes.) In the example,

  ```
  KEYIN '%V(27)0' IS ...
  ```

  %V(27) represents the <ESC> control character; %V is the $CONTROL-Value function and (27) is the ASCII decimal code for <ESC>, which in this example is defined as being followed by the zero character.

- Using EQUALS names that you have previously defined, usually in your context file. The standard context file (see Appendix C) defines names for the most common non-displayable characters: ESC for <ESC> and CR for <CR>. Then, use the CONCATENATE operator to combine the name with other characters to comprise string_expression_1. For example:

  ```
  KEYIN ESC||'L' IS...
  ```

$KEY and $CNTL or their angle bracket notations provide advantages over the other two methods of expressing non-displayable characters. The chief advantages are as follows:

- o Self-documenting commands.

  Expressing keystrokes as you see them on the keyboard makes it easier to remember how to invoke the keyin by looking at the command again, or displaying the available keyins with the SHOW command.

- o Expanded number of special sequences available at the terminal.

  Your terminal probably has keys that send two-, three-, or even four-character sequences when pressed; you can assign meanings within 6Edit to many of these keys. Without $KEY, you would need to use both $INTRO_1 and $INTRO_2 if you assign both two- and three-character sequences, and you would find it quite difficult, if not impossible, to assign meanings to the four-character keys.

  Using the default single-character introducer, <ESC>, you might not be able to define enough functions to make editing convenient in your environment. You would then need to define some that require more than one keystroke to enter. Since most of the two-stroke sequences using <ESC> are already defined for input editing functions in the FEP, you would have to either give up some of those or use less-convenient three-stroke keyins.

  With $KEY, however, you can define all the single-stroke keyins without using either $INTRO_1 or $INTRO_2 at all. With the introducer names still free, you can then use any key as the introducer for up to two sets of two-stroke keyins. You can also choose as the introducer a key that sends more than one character. This is much more convenient to type than a control character, and does not force you to use a printing character. You then have the equivalent of four sets of keyins available instead of two.

Examples:

The examples below are numbered for ease of use.

Example 1a.

```
KE <ESC>||'L' IS <CNTL-C>||'SKIP 20 BO RECORDS'||<CR>
```

causes the key sequence Escape-L to generate the characters ETX, SKIP 20 BO RECORDS, and CR. When <ESC> <L> is typed at the keyboard, three things happen:

1. The <CNTL-C> character causes the cursor to move to the command window. This occurs because <CNTL-C> is 6Edit's input activation function to move the cursor to the command window.

2. The command "SKIP 20 BO RECORDS" is entered in the command window.

3. The <CR> character causes the command to be executed by 6Edit.

Example 1b.

```
KE ESC||'L' IS CMD||'SKIP 20 BO RECORDS'||CR
```

is identical to the previous command, but it takes advantage of EQUALS names. This command assumes that you have already entered the following command:

```
ESC EQUALS <ESC>; CMD EQUALS <CNTL-C>; CR EQUALS <CR>
```

Note that since string_expression_1 is two characters long in both of the above examples, $INTRO_1 or $INTRO_2 must be set to <ESC> for the key definitions to work.

Example 1c.

By using the $KEY function in the above examples, $INTRO_1 or $INTRO_2 do not have to be used. With $KEY, any available single function key that sends a character sequence (usually ESC and something else) can be used instead of the Escape-L sequence to perform the "SKIP 20 BO RECORDS" function.

Suppose F2 is the desired key for this function. The following KEYIN command can then be entered:

```
KE <F2> IS <CNTL-C>||'SKIP 20 BO RECORDS'||<CR>
```

With the EQUALS names defined as above, this command could also be entered as:

```
KE <F2> IS CMD||'SKIP 20 BO RECORDS'||CR
```

If the F2 key on your keyboard sends <ESC><L>, these commands perform the same functions as the first two KEYIN commands above, but with a single keystroke, F2, and without having to first set an introducer character with $INTRO_1 or $INTRO_2.

Example 2:

```
KEYIN '%V(23)' IS '%V(27)5R%V(23)'

        or

KEYIN $CNTL(W) IS '%V(27)5R%V(23)'
```

defines the <CNTL-W> key, specified by %V(23), to move the cursor left by one word. In this command:

- o %V(27)5R is the ASCII decimal code for escape (<ESC>), followed by "5R". (See Appendix F for the ASCII character codes.) This input editing function sets the "left" direction for the "move one word" function to follow. (See Input Editing Functions in Appendix E for more information.)

- o %V(23) is the ASCII decimal code for <CNTL-W>. <CNTL-W> is the "move one word" input editing function.

If all the "one-stroke" function keys in your environment were already defined, and you might want to use a shifted function key instead of <CNTL-W>, so you could enter the following KEYIN command:

```
KEYIN $KEY(SF3) IS '%V(27)5R%V(23)'
```

This performs the "move left one word" function with two keystrokes, Shift-F3. Single and shifted function keys can be defined in a similar manner for the rest of the examples below.

Example 3:

```
KEYIN <CNTL-E> IS '%V(27)5S%V(23)'
```

defines the <CNTL-E> key to move the cursor right by one word. In this command, similarly to Example 2, %V(27)5S sets the "right" direction for the "move one word" function to follow.

Example 4:

```
KEYIN <CNTL-D> IS <ESC>||'5R'||<ETB>||<ESC>||<ETB>
```

defines the <CNTL-D> key to delete the word to the left of the cursor. In this command:

- o <ESC>II'5R' is an input editing function that sets the "left" direction for the "move one word" function to follow. (See Input Editing Functions in Appendix E for more information.)
- o <ETB> is the ASCII character code for <CNTL-W>. This is the "move one word" input editing function; the cursor moves to the beginning of the previous word in the record.
- o <ESC>II<ETB> are the ASCII character codes which comprise the "delete word" input editing function.

Example 5:

```
KEYIN <FF> IS <ESC>II<CNTL-W>
```

defines the <CNTL-L> key to delete the word under and to the right of the cursor. <FF> is the ASCII character code for <CNTL-L>.

Example 6:

```
KEYIN <CR> IS <ESC>II<LF> IN EDITING
```

causes the carriage return key to generate the characters Escape and Linefeed, in the editing window only; it does not affect the carriage return key when the cursor is in the command window.

This KEYIN is useful when you want to add several lines of text in the middle of a file, or if you are used to PC or UNIX editors where <CR> represents a record boundary. Normally, the carriage return key simply moves the cursor to the beginning of the next record; to add a new record, you must type <ESC> <LF> (the Escape and Linefeed keys). However, this KEYIN command allows you to use the carriage return key to generate the Escape Linefeed sequence. You must include an IN-clause in this KEYIN command because you do not want the carriage return key to generate <ESC> <LF> when you type it in the command window. By adding "IN EDITING" to the KEYIN command, the definition of the carriage return key in the command window is unchanged.

Related Topics:

Substitution

## LOCATION Command

Syntax:

```
LO[CATION] name [IS] [BL[OCK]] block_expression
```

Parameters:

name    is any name; it must be between 1 and 31 characters in length. A name may include:

- o Alphabetic characters
- o Digits (except as the first character of name)
- o $ _ # and @

block_expression    specifies the block description of the new value of name.

Description:

The LOCATION command remembers the boundaries of its block operand as the value of name. Thereafter, you can use name in a block expression, and 6Edit interprets it to refer to the block specified by block_expression in this LOCATION command.

The name parameter is required. It must be a name you create.

The operand of the LOCATION command is called a "named block."

Between the time the LOCATION command is given and the time of a reference to name, you should be aware that changes to the contents or location of the named block will effectively change the value of name as well, since name simply refers to the "location" of the named block, it does not hold a copy of the block's contents.

Example:

```
LOCATION C1 IS BO 1 THRU 4 POSITION 15
```

finds the specified block, and assigns the location of that block to the name C1. In subsequent commands, 6Edit evaluates C1 as representing this location. For example:

```
COPY C1 AFTER 7
```

copies from the beginning of record 1 through column 15 of record 4. The data is inserted after record 7. C1 still refers to the block located at BO 1 THRU 4 POSITION 15. It did not move with the data.

Related Topics:

Substitution

## MOVE Command

Syntax:

```
MO[VE] [BL[OCK]] block_expression
```

Parameters:

| | |
|---|---|
| block_expression | is the block of data to be moved. |

Description:

The basic operation of the MOVE command is the same as for the COPY command. The operand becomes the new source block.

For the MOVE command, however, 6Edit remembers that the source block is to be deleted after use. Later, when an AFTER or OVER command refers to the source block, it will delete the source block after copying it to the specified destination.

After receiving this command, the operand becomes the new selected block.

Example:

```
MOVE 2 THRU 22 AFTER 40
```

moves the records 2 through 22 to a new position after record 40. The source block (records 2 through 22) is deleted after being copied to its new location (after 40).

Related Topics:

Copy/Move Operations
COPY Command
AFTER Command
OVER Command

## OUTPUT Command

Syntax:

```
         [[ON  ] LP[@location]]
OUT[PUT] [[OVER] fid          ] [(option[,option]...[)]]
         [[INTO] ME           ]
         [[TO]                ]
```

Parameters:

{ON|OVER|INTO|TO}    directs output processing. OVER causes an existing file to the overwritten. INTO causes file extension. ON and TO are synonyms used to create a new file. If the file exists, an error will occur. The default is ON.

fid    any valid CP-6 file identifier.

LP    directs output to the default line printer. @location identifies a specific line printer.

ME    redirects output to the user's terminal.

option    is one of the following:

F[ORM]={formname|'formname'}    FORM applies to unit record files and unit record devices. Specifies the name of a form to be mounted on a unit record device. formname is a 1-6 character string, which can be quoted. The name must have been defined to the system by the system manager (via the Form Definition File). The default is blank, which means the default form for a unit record device and form 'STDLP' for unit record files.

I[XTNSIZE]=value    Specifies an initial disk storage allocation (in blocks) for the output file. value must be in the range of 1 through 134217727. The default is 2.

O[RGANIZATION={C[ONSECUTIVE]|U[NIT]R[ECORD]}    Specifies the organization of the output file as either CONSECUTIVE or UNIT RECORD. If this option is not specified, the output file is created as a UR file.

X[TNSIZE]=value    Use of this option causes an automatic M$EXTEND when needed. value specifies a secondary disk storage allocation (in blocks) for the output file, and must be in the range 1 through 32000. The default is 2.

Description:

This command sends subsequent output to the specified file.

## OVER Command

Syntax:

```
OV[ER] [BL[OCK]] block_expression
```

Parameters:

| | |
|---|---|
| block_expression | is the block of data to be replaced by the source block. |

Description:

The OVER command is used for replacement. It is generally equivalent to first deleting the destination block by using block_expression, then inserting the source block into the location vacated by the destination block. (The source block was set by the last COPY or MOVE command.)

The source and destination blocks may be whole records, parts of records, or any combination of these.

Example:

```
COPY 5 THRU 45 PO 10 OVER 230 THRU 460
```

deletes any data characters that previously existed in records 230 through 460. It then inserts the new data (from the beginning of record 5 through position 10 of record 45) at record 230.

Related Topics:

Copy/Move Operations
COPY Command
MOVE Command
AFTER Command

## PRINT Command

Syntax:

```
PRINT [ALL|ldevlist]
```

Parameters:

ALL    specifies that the accumulated outputs for all logical devices are to be sent to their destinations immediately. This is the default.

ldevlist    specifies that the accumulated outputs for the specified logical device or devices are to be sent to their destination(s) immediately. The list is entered in the format

```
    ldevname[,ldevname]...
```

ldevname is a logical device name established through the LDEV command.

Description:

PRINT directs that output accumulated for logical devices be sent to its destination immediately.

Example:

```
PRINT LP01,LP02
```

causes the accumulated output associated with logical devices LP01 and LP02 to be sent immediately to the associated destinations.

Related Commands:

ERASE

## READ Command

Syntax:

READ fid

Parameters:

fid is the fid of a file containing valid 6Edit commands.

Description:

This command directs 6Edit to read the specified file (fid). The file specified by fid must contain one 6Edit command line in each record. 6Edit reads the command lines and executes them as if they had been typed at the terminal, except that KEYIN substitutions are not performed. 6Edit stops reading command lines from the file when it reaches the end of the file; then it reverts to reading command lines from the command stream.

You may include a READ command as one of the commands in the file specified by fid. You may nest READ commands in this way indefinitely.

When using multiple commands on a line, the READ command can appear at any place on the command line.

## RESTORE Command

Syntax:

REST[ORE] fid

Parameters:

fid specifies the fid of a file which was created with 6Edit's SAVE command.

Description:

The RESTORE command restores context information from a file which was created with the SAVE command. It merges the information from the SAVE-file with the current context in the following way:

- SYNONYMs and KEYINs from the SAVE-file are added to those already defined when the RESTORE command is given.

- o Predefined EQUALS-names from the SAVE-file replace the values of all predefined EQUALS-names at the time the RESTORE command is given. That is, the values of all predefined names (see Appendix A, Predefined Names) are saved in the SAVE-file; therefore, when the RESTORE command is given, all operating parameters are restored to their values as saved in the SAVE-file.

- o User-defined EQUALS-names from the SAVE-file are added to the EQUALS-names already defined when the RESTORE command is given.

- o LOCATION-names from the SAVE-file are added to those already defined when the RESTORE command is given. If the files to which these names refer no longer exist, or if they have been rekeyed, then using such LOCATION-names may give unexpected results.

- o If the SAVE-file contains window information (i.e., if SAVE SESSION or SAVE ALL was used to create the SAVE-file), then when the RESTORE command is given, any editing windows and their edit block stacks are discarded. In addition, the editing window information from the SAVE-file is used to create a new editing window(s) and edit block stack(s). Again, if the files to which this window information refers no longer exist, or if they have been rekeyed, then the RESTORE command may give unexpected results.

Related Commands:

SAVE

## SAVE Command

Syntax:

```
     [ AL[L]      ] {  TO  }
SAVE [ SETU[P]    ] {  ON  }  fid
     [ SESS[ION] ] { OVER }
```

Parameters:

fid    is the fid of the file to save the requested context information in.

ALL    saves all of the context file information.

SETUP    saves the value of all EQUALS-names, KEYIN-names, and SYNONYM-names. It does not save the value of LOCATION-names or any window information.

SESSION    saves the value of all LOCATION-names, as well as information about the windows currently on your terminal screen. For the editing windows, this includes the window dimensions, the files you are currently editing, the current locations of the cursor, and the entire edit block stacks (the files you edited previously), for each editing window.

Description:

The SAVE command directs 6Edit to save current context information in a file. The file created by the SAVE command is called a "SAVE file". The SAVE file can be used later to restore the saved context using the RESTORE command (or using the file as the context file which is read when 6Edit is started, see Context Files). If you do not specify the type of information to be saved, all the current context file information is saved.

If "TO fid" or "ON fid" is used, the file specified by fid must not yet exist; the SAVE command creates the file. If "OVER fid" is used, then the SAVE command replaces the file if it already exists, or creates it if it does not yet exist.

The file created by a SAVE command is a special encoded "workspace" file, with file type "We". It cannot usefully be edited using EDIT or 6Edit, and should not be altered in any way by the user. 6Edit can detect most changes to the file, and will not restore an encoded context file which has been modified.

Related Topics:

RESTORE Command
Context Files

## SHOW Command

Syntax:

SH[OW] option

Parameters:

option    is one of the following:

ST[RING] string_expression    displays the string represented by string_expression.

KE[YINS]    displays the keyin definitions.

[EQ[UALS]] [name [TH[RU] name2]]    displays one, some, or all EQUALS definitions.

LO[CATIONS] [name [TH[RU] name2]]   displays one, some, or all LOCATIONS definitions.

SY[NONYMS] [name [TH[RU] name2]]   displays one, some, or all SYNONYMS definitions.

[BL[OCK]] block_expression   displays complete records selected by a block expression, preceded by the record key in an appropriate format.

[WI[NDOWS]]   displays window definitions.

Description:

The SHOW command is used for terminal display of string operands, the values of names created with assignment commands (KEYIN, EQUALS, LOCATION, and SYNONYM), and records within a selected block.

The SHOW command displays its output through M$LO, which defaults to the command window. Unless the command is activated with <LF> (instead of a carriage return), the command window is first expanded to occupy the space used by the editing window and the IBEX window, and one additional read from the command window is forced after the display. With <LF>, the display is presented in the command window and the cursor returns immediately to the editing window.

In SHOW displays of KEYIN definitions or EQUALS names for which different definitions exist for different windows, the window for which a definition applies is indicated in parentheses after the name. (edt) represents the first editing window. (cmd) represents the command window. (edn) represents the nth editing window.

Example:

```
SHOW STRING $KEY(ESC)||'[36m'
```

On a color ANSI-compatible terminal, changes the display to a light-blue color.

```
SHOW STRING HERE||', '||$DATE(TY='LOCAL')<LF>
```

displays the current file FID, record key, column, and date in the command window, without interrupting the editing window.

```
SHOW $AUTOTAB
```

displays the current setting of the $AUTOTAB EQUALS variable (predefined name).

SHOW 'A'

displays the next record in the current file that contains an "A".

SHOW FILE :6EDIT_CONTEXT REPEAT SELECT RECORD SELECT 'A'

displays all records in the file :6EDIT_CONTEXT that contain an "A".

SHOW $NOT_COPIED

displays the records that were not included in a failed copy operation.

SHOW LO $NOT_COPIED

displays a block expression describing the records that were not included in a failed copy operation.

SHOW $INSERT

displays the current setting of the $INSERT EQUALS variable. If $INSERT is set to ON only in the command window, the display would be "$INSERT(cmd)      ON".

SHOW WINDOWS

displays the current window definitions.

## SYNONYM Command

Syntax:

SY[NONYM] name [IS] fragment

Parameters:

name    can be any character other than $CONTROL, $COMMENT, semi-colon (;), or have a digit as the first character.

fragment    is any set of words, quoted strings, or special characters, up to the end of the command line.

Description:

The SYNONYM command assigns to name any fragment of a 6Edit command line. It is used to create your own variations of the basic syntax.

Because name can be either a normal name (which looks like a keyword) or a string of special characters (which looks like an operator symbol), you can create your own syntax by defining name to be some series of already-defined names or symbols.

The fragment parameter specifies any set of words, quoted strings, or special characters, up to the end of the command line.

The following restrictions apply to the fragment parameter:

o Quoted strings must be completely specified (that is, you cannot have unbalanced string delimiters in a fragment).

o The command separator character semicolon (;) has no effect in a fragment when the SYNONYM command is processed. That is, it does not mark the end of the SYNONYM command; instead, it is included in a fragment. Later, when name is given in a command line, any semicolons included in the fragment (along with the other characters in a fragment) are recognized, and demarcate separate commands in the command line.

  A semicolon appearing at the end of a command line normally indicates that the command is continued on the next line. However, this is not true when a SYNONYM command appears in the command line. As for the command separator, the trailing semicolon is included in the fragment. Later, when the SYNONYM name is given in a command line, the trailing semicolon in a fragment is treated as a command separator, if more characters follow the SYNONYM name on the command line, or as a command line continuation indicator, if no characters follow the SYNONYM name.

o Other substitutions can be used in a fragment. Specifically, names defined with previous SYNONYM commands may appear. Substitutions for them take place when this SYNONYM command is processed and before the new value is assigned to name. Names defined with the EQUALS and LOCATION commands may also appear in a fragment; however, substitutions for them do not occur until name is used later in a command line.

o The current $COMMENT character is recognized in a fragment, and has its usual effect at the time the SYNONYM command is processed.

o 6Edit does not recognize $CONTROL characters in quoted strings until the synonym is used. Then, only the current value of $CONTROL is acknowledged. Be careful when using $CONTROL functions in a fragment: do not change the value of $CONTROL, or unexpected results might occur.

Example:

```
SYNONYM EP IS EDIT PREVIOUS
```

makes EP a synonym for EDIT PREVIOUS. When EP is typed at the command prompt, 6Edit will perform the EDIT PREVIOUS command.

```
SYNONYM - THRU
```

renders the dash (-) a synonym for THRU. When you type <-> in a command, 6Edit henceforth interprets it as the THRU keyword.

Related Topics:

Substitution

## TIME Command

Syntax:

```
TIME
```

Parameters:

None

Description:

TIME displays the current time and date. The format of the display is:

```
mmm dd 'yy hh:mm    (month)(day)(year)(hours)(minutes)
```

Example:

```
TIME
```

displays the current time and date.

Related Commands:

DATE

## WINDOW Command

Syntax:

Form 1, for creating a new editing window:

```
                  [      {PO[SITIONS]} ]
NE[W]  WI[NDOW] [ expr {LI[NES]     } ]
                  [      {PE[RCENT]   } ]
```

Form 2, for changing an existing editing window:

```
                         [      {PO[SITIONS]} ]
WI[NDOW] [window_num] [ expr {LI[NES]     } ]
                         [      {PE[RCENT]   } ]
```

Form 3, for switching to another editing window:

```
{ NEXT        }              [      {PO[SITIONS]} ]
{ PREV[IOUS] }  WI[NDOW]  [ expr {LI[NES]     } ]
                             [      {PE[RCENT]   } ]
```

Form 4, for deleting an editing window:

```
DE[LETE] WI[NDOW] [window_num]
```

Parameters:

expr    is a 6Edit string expression which evaluates to a number.

window_num    is a 6Edit string expression between 1 and the number of editing windows.

Description:

The WINDOW command can be used to create a new editing window, change or remove an existing editing window, or to switch to another editing window.

- o Creating a New Editing Window
  To create an editing window, form 1 of the WINDOW command is used:

  NEW WINDOW expr POSITIONS    will create a new window which is "expr" positions wide and the height of the editing window portion of the 6Edit screen. The new window will be created at the right of the rightmost editing window. Not implemented for A03 version.

  NEW WINDOW expr LINES    will create a new window which is "expr" lines high and the width of the 6Edit screen. The new window will be created at the bottom of the bottom editing window.

  NEW WINDOW expr PERCENT    will create a new window which is "expr" percent of the size of the 6Edit screen, along the dimension of the window creation mode. The new window will be created at the bottom or right of the bottom or rightmost editing window.

  If expr is given but neither POSITIONS, LINES, nor PERCENT is given, then PERCENT is assumed.

  Examples:

  NE WI

  will create a new editing window in the current creation mode; if there are no editing windows yet or just one, a new vertical window will be created at the bottom of the current editing window. $WI_PERCENT will determine the window size.

  NE WI; FI xxx

  will create the new editing window, and will begin editing file "xxx" in the new window.

- o Changing or Removing an Editing Window
  To change the size of the current editing window, form 2 of the WINDOW command is used.

  To remove the current editing window, use:

  DELETE WINDOW

  The window will be deleted, removing it from the screen. The space it occupied will be given to the editing window above it or to its left.

If expr (given as either a number of positions, a number of lines, or as a percentage of the screen size) specifies the entire editing window portion of the 6Edit screen (or anything larger), then 6Edit assumes you want to remove all editing windows except the current one. Thus:

```
    WINDOW 100 PERCENT
```

will make the current editing window into the only one on the screen, temporarily removing all others.

o Switching Between Editing Windows
There are two ways to switch to a different editing window: using the WINDOW command, or by typing an activation character.

In the command window, you can type form 3 of the WINDOW command:

```
    { NEXT       }
    { PREV[IOUS] }  WINDOW
```

This will move the cursor to the current file pointer location in the window to the "next" or "previous" editing window. NEXT means the window to the right or below the current window; PREVIOUS means the window to the left or above the current window. This movement "wraps" at the edges of the editing window portion of the 6Edit screen; for example, typing "PREVIOUS WINDOW" when the cursor is in the left-most editing window will move the cursor to the right-most editing window.

Also in the command window, you can type form 2 of the WINDOW command specifying a window number but no size option:

```
     WINDOW 2
```

This will move the cursor directly to the indicated editing window.

If 6Edit is being run on-line, the <CNTL-N> activation character can be used to move the cursor to the next editing window. "Wrapping" around the screen at the edges occurs as described above.

Even if 6Edit is reading from the command window when <CNTL-N> is typed, the cursor will be moved to the next editing window.

Related Topics:

Multiple Editing Windows
EQUALS Command
KEYIN Command

# Section 4

# String Expressions

## String Expressions

A string expression can specify either a character or a numeric value, depending on how it is used. Certain commands and string operators anticipate character operands. For them, an expression which evaluates to a numeric value is converted to a character string containing a decimal representation of the numeric value. If a command or string operator expects a numeric operand and you supply a string expression which evaluates to a character value, then 6Edit assumes that the string contains a decimal representation of a number, and tries to convert the string to a numeric value. If the string does not contain a reasonable decimal representation of a number, 6Edit rejects the expression and the command in which it is used. The maximum length of a string expression is 511 characters.

String expressions consist of functions, operators, constants, and variables. In 6Edit (and IBEX) string expressions, constants are either decimal numbers or quoted strings. Variables are symbols whose values have been defined by 6Edit EQUALS (and IBEX LET) commands, or by the processor itself (such as the 6Edit predefined names and IBEX system variables).

Most of the 6Edit's string operators and functions are also implemented by IBEX. The following tables list 6Edit's string operators and string functions. (For a complete description of the string functions, refer to Appendix B, Predefined String Functions.)

Table 4-1. String Functions

| | | | |
|---|---|---|---|
| $ABS | $FID_ORG | $LOC | $SUBSTR |
| $ACCT | $FID_RECS | $MAX | $SWITCH |
| $CMDVAR | $FID_TYPE | $MIN | $SYSID |
| $CNTL | $FID_UGRANS | $MOD | $TERM_FEP |
| $DATE | $FLAG | $MODE | $TERM_LINE |
| $DAY | $HSET | $NAME | $TERM_PROFILE |
| $DIR | $INDEX | $PRIV_ACTIVE | $TERM_SPEED |
| $EOF | $INPUT | $PRIV_AUTH | $TIME |
| $FID_ASN | $KEY | $REM | $UPC |
| $FID_EXIST | $KEYIN | $RERUN | $VERIFY |
| $FID_GRANS | $LASTBATCH | $SEARCH | $VERSION |
| $FID_NGAVAL | $LENGTH | $SITE | $WOO |

Table 4-2. String Operators

| | | | |
|---|---|---|---|
| & | .AND. | <= | .LE. |
| <? | .CONTAINS. | < | .LT. |
| ¦¦ !! | .CONCAT. | =? | .MATCHES. |
| / | .DIVIDED. | - | .MINUS. |
| = | .EQ. | ~= | .NE. |
| >= | .GE. | ~ | .NOT. |
| > | .GT. | \| | .OR. |
| ?= | .IMB. | + | .PLUS. |
| ?> | .IN. | * | .TIMES. |

Note that string operators can be typed as symbols, or words. If words are used, however, they must be enclosed by periods.

The special characters used as string operators are not reserved for this use. You can use the SYNONYM command (see Section 3, 6Edit Commands) to assign a special character to some other 6Edit keyword or command.

For example, the dash (-) is often interpreted to mean "through." To use the dash in this way, you would define it as a synonym for the 6Edit THRU keyword:

```
    SYNONYM - THRU
```

After defining the dash to mean THRU, you cannot use it to mean subtract. Instead, use .MINUS. to perform subtraction functions, or define another synonym:

```
    SYNONYM _ .MINUS.
```

After defining these two synonyms, you can use a dash (-) to mean THRU, and an underscore (_) to mean subtract.

Example:

Using the above synonyms,

DI 5-10

means display records 5 through 10.

SIZE EQ 5_10

means the name SIZE now has the value -5.

## Logical Operators

Syntax:

```
number {&|.AND.} number
number {||.OR.} number
{~|.NOT.} number
```

Parameters:

number    is a string expression. It evaluates to a logical value, and must consist of one or more numerals.

Description:

These operators compute Boolean values from their operands. All operands are assumed to represent logical values, that is, either a true value or a false value. All operands must be numeric; 6Edit interprets their values as:

- o True, if number is non-zero (either greater than zero or less than zero).
- o False, if number is exactly zero.

The .AND. operator forms the logical conjunction of its two operands. The result is either one or zero, representing true or false respectively. The result is one if both operands are true. The result will be zero if either or both operands are false.

The .OR. operator forms the logical union of its two operands. The result is either one or zero, representing true or false respectively. The result is zero if both operands are false. The result is one if either or both operands are true.

The .NOT. operator forms the logical inverse of its operand. The result is either one or zero, representing true or false respectively. The result is zero if the operand is true. The result is one if the operand is false.

## Relational Operators

Syntax:

```
string-1 {=|.EQ.} string-2
string-1 {~=|.NE.} string-2
string-1 {<|.LT.} string-2
string-1 {<=|.LE.} string-2
string-1 {>|.GT.} string-2
string-1 {>=|.GE.} string-2
```

Parameters:

string-1    is a string expression. It evaluates to the operand to be compared.

string-2    is a string expression. It represents the value to which string-1 is compared.

Description:

These operators compare their two operands.

If both operands are numeric, then a numeric comparison is performed (10 is equal to 010, 01 is less than 010). If either operand is not numeric, a character-by-character string comparison is performed. If one string is shorter than the other, 6Edit pads it with blanks.

The result computed by each of these operators is a numeric operand: either 1, indicating that the relation is true, or 0, indicating that the relation is false.

The result of the .EQ. operator is true if the two strings are equal, false otherwise.

The result of the .NE. operator is true if the two strings are unequal, false otherwise.

The result of the .LT. operator is true if string-1 is less than string-2, false otherwise.

The result of the .LE. operator is true if string-1 is less than or equal to string-2, false otherwise.

The result of the .GT. operator is true if string-1 is greater than string-2, false otherwise.

The result of the .GE. operator is true if string-1 is greater than or equal to string-2, false otherwise.

## Wildcard Operators

Syntax:

```
string-1 {?=|.IMB.} string-2
string-1 {=?|.MATCHES.} string-2
string-1 {?>|.IN.} string-2
string-1 {<?|.CONTAINS.} string-2
```

Parameters:

string-1    is a string expression. It evaluates to the operand to be compared.

string-2    is a string expression. It represents the value to which string-1 is compared.

Description:

These operators compare their two operands, taking the wildcard function into account.

Wildcard operators resemble relational operators, but allow one string to contain one or more "wildcard" characters, the question mark (?). A wildcard character (?) appearing in one string matches a sequence of any number of characters (including 0 characters) in the other string. For example:

If the wildcarded string is

    'Product is ? stock'

then it equals both of the following strings:

    'Product is in stock'

    'Product is out of stock'

Note that the wildcard character (?) used by these operators appears differently than the $CONTROL-Question Pattern function. (See Section 5, Block Expressions in 6Edit.) In the latter instance, the $CONTROL character is used, whereas for wildcard operators, the question mark is used alone.

Like the relational operators, the result computed by each of these wildcard operators is a numeric operand: either 1, indicating that the relation is true, or 0, indicating that the relation is false. Unlike the relational operators, string-1 and string-2 are always treated as sequences of characters: a character-by-character string comparison is always performed.

The .IMB. and .MATCHES. operators look for the complete match of a wildcarded string. The .IN. and .CONTAINS. operators look for a contained wildcarded string.

The result of the .IMB. ("Is Matched By") operator, accounting for wildcard characters, is true if the two strings are equal, false otherwise. Question marks appearing in string-1 represent wildcard characters; however, question marks appearing in string-2 are given no special interpretation, they are taken at face value.

The result of the .MATCHES. operator, accounting for wildcard characters, is true if the two strings are equal, false otherwise. Question marks appearing in string-2 represent wildcard characters; however, question marks appearing in string-1 are given no special interpretation, they are taken at face value.

The result of the .IN. operator, accounting for wildcard characters, is true if string-1 is a substring of string-2, false otherwise. Question marks appearing in string-1 represent wildcard characters; however, question marks appearing in string-2 are given no special interpretation, they are taken at face value.

The result of the .CONTAINS. operator, accounting for wildcard characters, is true if string-1 contains string-2 as a substring, false otherwise. Question marks appearing in string-2 represent wildcard characters; however, question marks appearing in string-1 are given no special interpretation, they are taken at face value.

## Arithmetic Operators

Syntax:

```
[number-1] {+|.PLUS.} number-2
[number-1] {-|.MINUS.} number-2
number-1 {*|.TIMES.} number-2
number-1 {/|.DIVIDED.} number-2
```

Parameters:

| | |
|---|---|
| number-1 and number-2 | represent the operands to be combined. |

Description:

These operators perform arithmetic functions with their operands. All operands must be numeric.

The .PLUS. operator adds its two operands. If number-1 is omitted, the result is number-2 itself.

The .MINUS. operator subtracts number-2 from number-1, or, if number-1 is omitted, from zero.

The .TIMES. operator multiplies its two operands.

The .DIVIDED. operator divides number-1 by number-2.

## CONCATENATE Operator

Syntax:

```
string-1 {|| or !! or .CONCAT.} string-2
```

Parameters:

| | |
|---|---|
| string | identifies the character or characters to be joined. |

Description:

The .CONCAT. operator joins its two string operands into one string.

# Section 5

# Block Expressions in 6Edit

This section discusses the types of block expressions used in 6Edit, and explains them in context.

## Using Block Expressions

6Edit editing commands work with "blocks" of data. A block can be an entire file, or any portion of a file.

A "block expression" specifies a block in a file. As with other types of expressions, a block expression consists of block operands, which are combined by block operators to yield a result. The result of the evaluation of a block expression is itself a block operand.

Block operands differ from operands in string expressions. They not only have values associated with them, they have locations as well. The "location" associated with a block operand is the location of a block within a file; the value of a block operand is the contents of the file at that location.

Evaluation of block expressions also differs from evaluation of string expressions. The evaluation of block operands results in not only a value and a location, but a signal as well. This signal indicates the success or failure of the evaluation of the block operand. Continued evaluation of the block expression depends on the success or failure of the evaluation of each block operand in the block expression.

### Specifying User-Defined Block Operands

In a block expression, you can use the basic block operands which are predefined by 6Edit, or you can define a block yourself and use it as a block operand. For example, you can define a group of three contiguous records as a block, then use this block as a block operand in a block expression.

In fact, the result of any block expression is a "user-defined block."

By combining the basic block operands and other user-defined block operands with "block options" and "block operators," you can define a user-defined block to be any continguous set of data characters anywhere in any file. This is done generally as follows.

After indicating the file in which the data characters reside, you use block operators and operands to move a file pointer through the file. The file pointer points to a single character in a record. Usually, you delimit a block by:

1. Moving the file pointer to the first character of the block (the starting point);

2. Typing the THRU keyword;

3. Moving the file pointer to the character following the last character of the block (the ending point).

Note that blocks need not start or end on record boundaries.

Basically, a block expression specifies three things:

- The "enclosing block." This is the file (or other block) in which the block being specified resides. If omitted, the edit block is used as the enclosing block. In commands, you can specify an enclosing block which is different than the edit block by explicitly naming the file (or other absolute block operand; see below) to use as the enclosing block for this block expression.

  The enclosing block limits the movement of the file pointer and hence the bounds of the block being specified by the block expression. The block being specified must lie completely within the bounds of the enclosing block.

- The starting point of the block. If omitted and an enclosing block was specified explicitly, then the starting point of the block defaults to the start of the enclosing block; if omitted and no enclosing block was explicitly specified, then the starting point of the block defaults to the current location of the file pointer.

  When moving the file pointer to the desired starting point, you are not allowed to move it outside the bounds of the enclosing block. Attempts to do so will be rejected.

- The ending point of the block. If omitted, the end of the last block operand given in the starting point specification is assumed; if the starting point was itself omitted, then the ending point of the block defaults to the end of the current selected block.

When moving the file pointer to the desired ending point, you are not allowed to move the file pointer outside the bounds of the enclosing block. Attempts to do so will be rejected.

Block expressions allow you to specify these components in many different ways. For example, after specifying a starting point and an ending point for the block being defined, you can change either or both boundaries (ADJUST clause). This can be useful when you are specifying the block interactively, and you want to change the boundaries of the block being defined as you examine the data in the file.

Also, you can specify a "repeated block." A repeated block is a block which is repeated throughout a file. It must not contain any absolute block operands. For example, you can define three records of a file to be a block; you can also repeat that block expression, thus defining each three-record group in the file to be a block. When a command includes a block expression which specifies a repeated block, then the command is repeated, once for each instance of the repeated block. In the example just given, the command which included the repeated block expression would be repeated, each time operating on a different three-record group in the file.

## Concepts

This subsection describes the concepts necessary for a complete understanding of block expressions.

### Block Operands

Operands in block expressions are usually blocks themselves. You build up the specification of a block by using the basic block operands predefined by 6Edit:

- A file.
- A string expression whose value is treated as a file.
- A previously-selected and named block.
- The previous or next entry in the edit block stack.
- A record in a file.
- A data character in a record.
- A string of data characters in a file which match a user-specified pattern string.

Generally, you can combine any of the above basic blocks to specify your own block, no matter what the "shape" or location of the block. For instance, you can specify a block to be three records in a file, or to be two records in a file and the first twenty characters of the next record in the file.

There are two classes of block operands: absolute and relative. They are discussed in the following paragraphs.

## Absolute Block Operands

An "absolute" block operand is one which has a specific location. The absolute block operands are:

```
FILE fid
STRING string_expression
PREVIOUS
NEXT
location_name
record_key
```

When evaluating an absolute block operand, the location of the file pointer after evaluating the operand has absolutely nothing to do with the location of the file pointer before evaluating the operand. For instance, record 62 of file "ABC" is an absolute block operand; there is only one location specified by that block.

- FILE specifies a file to be used as a block operand. A block expression can contain only one FILE operand, which must be the first block operand in the expression.

- STRING specifies a string expression whose value is to be treated as a file, and used as a block operand. Usually the string expression value is treated as record 0 of a one-record file; however, by including the $CONTROL-Record function in the string expression, a multi-record "file" may be specified.

  Like FILE, STRING may be used only once in a block expression, and only as the first block operand in the expression.

- PREVIOUS designates the previous edit block as a block operand. There may be only one PREVIOUS operand in a block expression, and it must be the first block operand in the expression.

- NEXT designates the next edit block in the edit block stack as a block operand. There may be only one NEXT operand in a block expression, and it must be the first block operand in the expression.

- The "location_name" specifies a previously-specified and named block to be used as an absolute block operand. This allows you to define your own absolute block operands and give a name to them (with the LOCATION command); thereafter, that name can be used in other block expressions as an absolute block operand.

The location_names may be used more than once in block expressions; however, all location_names used in a block expression must specify blocks residing in the same file. If a location_name is not the first operand in a block expression, then all location_names in the expression must specify blocks residing in the same file as the current edit block.

- A "record_key" identifies a specific record within the current enclosing block which is to be used as a block operand.

## Relative Block Operands

A "relative" block operand is a block which may be located anywhere in any file. For instance, a record as a block (with no record key) may be located anywhere in a file. A file is typically made up of many such records.

The relative block operands are RECORD, POSITION, and pattern-string.

Relative block operands are used to tell 6Edit in a relative way how to move the file pointer. The location of the file pointer after evaluating the operand is relative to the location of the file pointer before evaluating the operand. For example, assume that the file pointer is at position 21 of record 62 in file "ABC"; you can tell 6Edit to move forward five RECORD. In this use, RECORD is a relative block operand; its actual location is relative to the current location of the file pointer. No specific record key was given, only "direction" and "skip" block options ("forward" and "five" respectively).

All the relative block operands specify a block located at or around the current file pointer location. By preceding the operand with the SKIP option, you can move the file pointer forward or backward in units of the block operand; for example, SKIP moves by records with the RECORD operand, but by positions within a record when used with the POSITION operand, and by pattern-matches when used with the pattern-string operand.

- RECORD specifies the record to which the file pointer points as the block operand.

- POSITION specifies as the block operand the one data character to which the file pointer currently points. It may also be used to specify a position within the record to which the file pointer currently points.

- "Pattern-string" is a string expression; its value is a pattern to search for within the enclosing block. If a match is found, the matching data in the enclosing block is the block operand specified by "pattern-string."

## Block Options

You can modify block operands with "block options."

If a block operand is given without a particular option, 6Edit uses as the default the current value of the predefined name for that option. There is a specific predefined name for each option. The value assigned to these names, if any, is usually the keyword for the desired option (see the EQUALS command). If the predefined name for an option has no value and the option was not specified with a block operand, 6Edit uses a standard default.

There are two kinds of block options: "evaluation options" and "processing options."

### Evaluation Options

The evaluation options tell 6Edit how to evaluate the block operand with which they are specified, that is, how to move the file pointer through the file looking for an instance of the block operand.

In a command line, the evaluation options are typed before the block operand to which they apply.

| "Point" option | "Direction" option | "Skip" option |
|---|---|---|
| ALL OF | FORWARD | SKIP |
| BEGINNING OF | BACKWARD | |
| END OF | | |

- o The point option specifies to what point in the block operand the file pointer is to be moved. ALL OF moves the file pointer to either the start or the end of the block operand, as appropriate, and selects the entire block operand. BEGINNING OF always moves the file pointer to the beginning of the block operand. END OF always moves the file pointer to the end of the block operand. Both of the latter select only the end point of the operand. The default is $POINT.

- o The direction option tells 6Edit in which direction the file pointer is to be moved from its current location, forward or backward. The default is $DIRECTION.

o The skip option specifies which instance of a relative block operand is to be located. For example, if you specify a skip option (with its skip count) with a pattern-string block operand, then 6Edit does not treat the first match of the pattern string as the block operand, but instead the "nth" match (where "n" is the skip count). The default is $SKIP.

## Processing Options

The processing options give 6Edit additional instructions on how to process a block after evaluation.

The processing options apply to the result of the evaluation of the entire block expression, not to just one operand in the expression. Therefore, in a command lir--, the processing options are typed before or after the entire block expression.

| OPTION: | "Protect" | "Exist" | "Structure" | "Key increment" |
|---|---|---|---|---|
| | PROTECT | NEW | EDITKEY | BY |
| | DONT PROTECT | OLD | STRINGKEY | |
| | | ANY | SEQUENTIAL | |

o The protect option controls modification of the contents of the block during editing. PROTECT prohibits any modification of the data in the block during the processing of the command which includes the block expression. DONT PROTECT explicitly allows modification. The default is $PROTECT.

o The exist option performs a test on the file you are about to process. After locating the file specified by a block expression, 6Edit examines the exist option (if any) included in the block expression. NEW tells 6Edit that this block evaluation succeeds only if the file does not exist, hence the file is created. OLD tells 6Edit that the block evaluation succeeds only if the file does exist. ANY creates the file if it does not exist, or uses it if it does exist. The default is $EXIST.

o The structure option tells 6Edit how to treat the file in terms of record keys and insertion techniques.

o The key increment option tells 6Edit how to generate new record keys when inserting records into a file. It is used on blocks which will be the destination of a copy or move operation. 6Edit inserts records into destination blocks; it needs to generate new record keys for these inserted records. BY tells 6Edit how to generate new record keys based on the existing record keys of records in the block. The default is $BY.

## Evaluation of Block Operands

The result of the evaluation of a block operand has three components:

- A new "location" for the file pointer.
- A "value," which is always the data in the file at the new location of the file pointer.
- A "signal" -- "success" or "failure" -- indicating whether or not 6Edit succeeded in evaluating the block operand.

## Signals

A block operand yields a "signal." The signal indicates the success or failure of the evaluation of the block operand.

The evaluation of a block operand (as modified by any block options specified with it) fails in these four cases:

- An absolute operand specifies a block which is located completely outside the boundaries of the current enclosing block. For example, if the enclosing block is records 50 through 100 of a file, a record key operand specifying record 200 will fail when evaluated.
- A pattern-string operand fails when evaluated if 6Edit cannot find a match for the pattern string inside the boundaries of the current enclosing block.
- A repeated block has already been evaluated the specified number of times.
- The starting range for evaluation of the next instance of a repeated block is outside the boundaries of the enclosing block.

Failure in evaluating a block operand is not considered an error. It is usually a perfectly normal occurrence, and controls evaluation of the block expression. For example, to replace all appearances of a string in a file with another string, you specify a repeated block to COPY OVER all matches of a pattern string. This repeated block causes the command which includes it (the COPY OVER command) to be repeated for each match of the pattern string. When 6Edit cannot find any more pattern string matches, evaluation of the pattern-string block operand fails, thus terminating processing of the command. This failure is intended, and is not an error.

## Block Expression Components

This subsection describes in detail the components of a block expression.

### General Form of Block Expressions

General Form:

```
[enclosing_block] [REPEAT] SELECT [movement]
    {THRU|ADJUST} [movement] [{THRU|ADJUST} [movement]]...
        [[REPEAT] SELECT [movement]
            {THRU|ADJUST} [movement] [{THRU|ADJUST} [movement]]...]...
```

Parameters:

enclosing_block    is any block expression. An "enclosing_block" may be omitted; if it is, the current edit block is used as the enclosing block for the expression.

REPEAT SELECT    serves two purposes: it may be used to enclose block expressions, and/or to specify a repeated block.

movement    is a "movement expression": a list of block operands, possibly modified by evaluation options. A "movement expression" moves the file pointer to a new location.

THRU    tells 6Edit that the next movement (movement expression) moves the file pointer to the end of the block being specified.

ADJUST    tells 6Edit that the next movement (movement expression) moves the file pointer to the opposite end of the block being specified.

Description:

As described earlier, block expressions specify three things: the enclosing block, the beginning point of the block, and the ending point of the block.

The first SELECT clause is needed only when "enclosing_block" is given, or when the SELECT clause specifies a repeated block.

Each "SELECT ... {THRU|ADJUST} ... {THRU|ADJUST} ..." group specifies the beginning and ending points of a block; each such group except the last specifies the enclosing block for the next SELECT group.

Note that all components of a block expression are optional, but at least one component must be given.

## SELECT Clause

Syntax:

```
[REPEAT] SE[LECT] block_expression
```

Parameters:

block_expression     must not include absolute block operands. Only relative block operands may be included in "block_expression."

Description:

The SELECT clause is used for either or both of two purposes: to specify repeated blocks, and/or to enclose blocks.

If REPEAT is given, then block_expression is a repeated block (see below).

### Repeated Blocks

To specify a repeated block, precede the block expression with a REPEAT SELECT clause.

Including a repeated block expression in a command causes the entire command to be repeated until the end of the enclosing block is reached. On each repetition of the command, the block expression evaluates to successive blocks.

6Edit continuously evaluates repeated block expressions. On the first evaluation, the file pointer starts at its current location. The block_expression should contain movement expressions which move the file pointer through the enclosing block, first to the beginning, and then to the end of the block being specified.

On successive evaluations, the file pointer starts at the end of the block specified by this block_expression on the previous evaluation. The block_expression then moves the file pointer (relative to the end of the previously-found block) first to the beginning, and then to the end of the next block for block_expression.

Example:

Assume you want to replace whatever appears between the words "the Piercy " and " we sold", with the word "novel". That is, the following strings in the file will be changed to "the Piercy novel we sold":

```
    the Piercy book we sold
    the Piercy picture we sold
    the Piercy novella on urban renewal we sold
```

However, the string "the Atwood book we sold" does not change.

To do this, specify a repeated block, searching for the given patterns to find the start and the end of the block to be replaced:

```
COPY STRING 'novel' OVER REPEAT SELECT EO 'the Piercy ' THRU BO ' we sold'
```

contains a REPEAT SELECT clause. This causes the COPY OVER command to be repeated as long as the block expression following SELECT can find a block in the file. The block expression is:

```
EO 'the Piercy ' THRU BO ' we sold'
```

specifying two patterns to be searched for. The block found by this block expression begins at the end of (EO) the next string in the file matching the first pattern, and ends at the beginning of (BO) the next string in the file matching the second pattern. When these two patterns are found, all data between them is replaced with the string 'novel'.

After each replacement, 6Edit repeats the pattern search (because REPEAT was specified). Starting from the end of the found block (in this example, the beginning of " we sold"), 6Edit searches again for the first pattern ("the Piercy "). If it finds a match, 6Edit searches for the second pattern, and if it finds a match for that pattern, another replacement is made. Then, 6Edit repeats the evaluation of the block expression, searching for the first pattern again.

This repetition continues until the end of the file (or other enclosing block) is reached in one of the pattern searches.

## Enclosed Blocks

An "enclosed block" is one which is entirely contained within another block. The outer block is called the "enclosing block"; the enclosed block is the block being specified, which is located entirely inside the enclosing block.

To specify an enclosed block, follow the specification of the enclosing block with the SELECT clause, followed by the specification of the enclosed block. (If the edit block is the enclosing block, then the SELECT clause is not necessary, unless a repeated block is desired.)

Enclosed blocks are useful when you want to limit the movement of the file pointer to some block smaller than the edit block. The edit block always limits all file pointer movement.

Example:

Assume you want to insert a string of characters "xxx" at the beginning of every record between records 10.0 and 50.0, inclusive. To do this, you type the following commands.

```
COPY STRING 'xxx'
AFTER 10 THRU 50 REPEAT SELECT RECORD SELECT BO RECORD
```

The first command, COPY STRING 'xxx', sets up the source block for the copy operation.

The AFTER command contains a complex block expression, including both enclosed blocks and repeated blocks:

```
10 THRU 50
```

First, the enclosing block is specified: the block comprising records 10.0 through 50.0 inclusive in the current edit block.

```
REPEAT SELECT RECORD
```

is a repeated block expression, specifying a block for every record between 10.0 through 50.0. A repeated block specification evaluates to several blocks; these blocks are enclosed within the enclosing block. In this case, the enclosing block is 10.0 through 50.0; the enclosed blocks will be each whole record which appears in the enclosing block.

SELECT BO RECORD

specifies an enclosed block: an empty block at the start of each record. Note that each block specified in the previous step becomes the enclosing block for this step.

## Movement Expressions

Syntax:

[block_options] block_operand ...

Description:

A "movement expression" is a sequence of block operands, optionally modified by block options. These tell 6Edit how to move the file pointer through the current enclosing block.

In its simplest form, a movement expression is a list of block operands. In front of each block operand, you can give block options to be applied to that operand.

Example:

20.3 BACKWARD SKIP 30 RECORDS POSITION 18

Three block operands compose this movement expression: the record key "20.3", "RECORDS", and "POSITION 18". The "RECORDS" operand is modified by two block options: "BACKWARD" and "SKIP 30".

This expression causes 6Edit to move the file pointer to record 20.3, then back up 30 records in the file, and finally move the file pointer to position 18 of that record.

## THRU Clause

Syntax:

{TH[RU]|TH[ROUGH]} [movement_expression]

Parameters:

| | |
|---|---|
| movement_expression | is a movement expression. |

Description:

THRU tells 6Edit that "movement_expression" specifies the ending location of the block being specified. Whenever THRU is seen, the current file pointer location becomes the beginning of the block being specified.

If "movement_expression" is omitted, the block being specified becomes an empty block (containing zero data characters) located at the current file pointer location. If "movement_expression" is included, it should move the file pointer to the desired ending location for the block being specified.

Example:

```
DE 20.1
```

deletes the entire record 20.1.

```
DE 20.1 THRU 25
```

deletes all records between and including 20.1 and 25.

```
5 THRU
```

specifies a block beginning and ending at the start of record 5.0. This is an empty block.

```
THRU 6
```

specifies a block beginning at the current file pointer location, and ending at the start of record 6.0.

## ADJUST Clause

Syntax:

```
AD[JUST] [movement_expression]
```

Parameters:

| | |
|---|---|
| movement_expression | is a movement expression. |

Description:

ADJUST allows you to change either boundary location of the block being specified. It tells 6Edit that "movement_expression" specifies the location of the boundary opposite to the one currently being specified.

The following example illustrates a command sequence, using the ADJUST clause. The vertical format suggests successive returns to the command window after each function.

```
Records in file INVENTORY:

  40
  41
  42
  .
  .
  .
  .
  48
  49

Commands in command window:

  41            moves the cursor to the beginning of record 41.

  TH            begins block selection.

  48            selected block is 41 TH 48; cursor is at the beginning of
                record 49.

  ADJUST        moves the cursor back to the beginning of record 41;
                it does not change the selected block.

  BA SK 1 RE    moves the cursor to beginning of record 40.  The selected
                block is now beginning of 40 thru end of 48.

  ADJUST        moves the cursor back to the end of the selected block:
                beginning of 49.

  DE THAT       deletes beginning of 40 thru end of 48.
```

Figure 5-1. Using the ADJUST Clause

ADJUST is most useful after a THRU clause. When 6Edit sees the THRU clause, its movement expression moves the file pointer to the end of the block being specified. When the first ADJUST following THRU is given, 6Edit remembers the current file pointer location as the ending location of the block being specified, and moves the file pointer to the beginning location of the block. The movement expression following this first ADJUST then changes the beginning location of the block, by moving the file pointer (from its initial location at the beginning of the block) to a new beginning location.

The second ADJUST after THRU acts similarly. 6Edit remembers the current file pointer location as the beginning location of the block, and moves the file pointer to the ending location of the block; the movement expression following the ADJUST changes the ending location of the block.

Each successive ADJUST repeats this process. Odd-numbered ADJUSTs after THRU (i.e., the first, third, fifth, etc.) change the beginning location of the block being specified; even-numbered ADJUSTs after a THRU change the ending location of the block being specified.

At any time in a block expression, you can use THRU. This clause renders the current file pointer location the beginning location of the block being specified; the movement expression following THRU always modifies the ending location of the block.

## Movement Expression Components

The following paragraphs describe in detail the block operands and options used in movement expressions.

### ALL OF Option

Syntax:

{AL[L] OF|AO} block_operand

Parameters:

block_operand    may be any absolute or relative block operand, possibly modified by other block options (only one of the three "point" options -- ALL OF, BEGINNING OF, and END OF -- may be used with a block_operand).

Description:

ALL OF tells 6Edit to move the file pointer to either the beginning or the end of "block_operand," depending on the context:

o In the first movement expression of a SELECT clause or in an ADJUST clause, this construct will move the file pointer to the beginning of block_operand.

o In the last operand of a THRU clause, this construct will move the file pointer to the end of block_operand.

Example:

```
DELETE AO 5 THRU AO 6
```

deletes the records between record 5.0 and 6.0, including all of records 5.0 and 6.0.

## BACKWARD Option

Syntax:

```
BA[CKWARD] relative_block
```

Parameters:

| | |
|---|---|
| relative_block | must be a relative block operand. |

Description:

BACKWARD tells 6Edit in which direction in the file to move the file pointer when searching for a relative block operand.

Relative_block must be a record, position, or pattern-string operand; 6Edit moves the file pointer backward accordingly.

Examples:

```
BACKWARD SKIP 8 RECORDS
```

moves the file pointer backward in the file, skipping over 8 records.

```
BACKWARD SKIP 28 POSITIONS
```

moves the file pointer backward in the file, skipping over 28 positions (characters in records).

```
BACKWARD 'abc'
```

moves the file pointer backward in the file, searching for the string "abc" in the file.

## BEGINNING OF Option

Syntax:

{BE[GINNING] OF|BO} block_operand

Parameters:

block_operand    may be any absolute or relative block operand, possibly modified by other block options (only one of the three "point" options -- ALL OF, BEGINNING OF, and END OF -- may be used with a block_operand).

Description:

BEGINNING OF tells 6Edit to move the file pointer to the beginning of block_operand. For instance, if block_operand was a block comprising three records, BEGINNING OF tells 6Edit to move the file pointer to the first character of the first of the three records.

Example:

DELETE BO 5 THRU BO 6

deletes the records between record 5.0 and 6.0, including record 5.0. However, record 6.0 will not be affected.

## BY Option

Syntax:

block_expression  BY  edit_key

Parameters:

edit_key    is a CP-6 edit key, that is, a string of 1 to 8 digits; if a decimal point is included in the string, there may be 0 to 3 digits to the right of the decimal point.

Description:

BY specifies a value to be used when 6Edit generates new record keys for a keyed file.

BY may be included in block expressions which specify destination block operands only; this only includes the operands of the AFTER and OVER commands.

If block_expression resides in an edit-keyed file and 6Edit must insert new records in that file, 6Edit must generate new record keys for the new records. 6Edit does this by adding the value of edit_key to the record key of the last existing record in front of the location in the file into which 6Edit is inserting records. Therefore, for edit-keyed files, edit_key must evaluate to a numeric operand whose value is between 0 and 99999.999, inclusive. If the generated key is greater than or equal to an existing key in the file, then 6Edit discontinues the operation, and displays an error message.

Example:

```
CO FILE ACCTS_PAYABLE 1 TH 50 AF FILE ACCTS_RECEIVABLE 1024 BY .1
```

If the BY option is not specified with a block operand, the current value of $BY is used as the record key increment.

```
CO FILE *1 OVER NEW FILE *2 BY 1
```

If the BY option is used for a block expression that also creates a new edit block as in the example above, that key increment value overrides the value in the $BY predefined name for the rest of the life of that edit block.

## END OF Option

Syntax:

```
{EN[D] OF|EO} block_operand
```

Parameters:

block_operand may be any absolute or relative block operand, possibly modified by other block options (only one of the three "point" options -- ALL OF, BEGINNING OF, and END OF -- may be used with a block_operand).

Description:

END OF tells 6Edit to move the file pointer to the end of block_operand. For instance, if block_operand was a block comprising three records, END OF tells 6Edit to move the file pointer to the first character of the record following the three-record block.

## FILE Block Operand

Syntax:

```
[ { NE[W] }  { ST[RING]KEY  } ]
[ { OL[D] }  { SE[QUENTIAL] } ] FI[LE] fid
[ { AN[Y] }  { ED[IT]KEY    } ]
```

Parameters:

fid    is a CP-6 file identifier and must refer to a file.

Description:

The FILE operand specifies a certain file to be the enclosing block for the rest of the block expression.

NEW tells 6Edit that fid must specify a file that does not exist. 6Edit will create the file as an edit-keyed file. $FILETYPE specifies the value to use for the type attribute of the new file.

OLD tells 6Edit that fid must specify a file which currently exists.

ANY tells 6Edit to create the specified file if it does not already exist.

STRINGKEY tells 6Edit that the file to be created or modified must have keys, and that the keys are presumed to be textual names. Insertion of new records is permitted only if the key of the new record is explicitly specified. Record keys are specified using the KEY option.

SEQUENTIAL tells 6Edit that the file to be created or modified must be unkeyed. Records may be added or deleted only at the end of the file. Record keys are sequential record numbers starting at 1.

EDITKEY tells 6Edit that the file to be created or modified must have standard (3-byte) edit keys. Records may be inserted or deleted anywhere. Record keys must evaluate to 3-place decimal numbers up to 99999.999.

The FILE block operand may not be modified by the following block options:

    SKIP
    FORWARD
    BACKWARD

The default "point" option for the FILE block operand is ALL OF, regardless of the current value of $POINT. To begin editing a file at its end, use two commands: EDIT FILE fid; EO CURR .

Example:

```
COPY REPEAT SELECT SKIP 2 'The %? we sold' AFTER ANY FILE SALES
```

creates the file SALES if it does not already exist, then searches forward in the current edit block for matches of the pattern "The %? we sold", and copies every other such match to the end of the SALES file starting with the second match. After searching to the end of the current edit block, 6Edit displays the SALES file in the editing window.

## FORWARD Option

Syntax:

```
FO[RWARD] relative_block
```

Parameters:

| | |
|---|---|
| relative_block | must be a relative block operand. |

Description:

FORWARD tells 6Edit in which direction in the file to move the file pointer when searching for a relative block operand.

Relative_block must be a record, position, or pattern-string operand; 6Edit moves the file pointer forward in the file (toward the end of the file) accordingly.

## Names as Block Operands

Syntax:

location_name

Parameters:

location_name    is a name. It may be either a predefined name ($LEFTOVER, $NOT_COPIED, HERE, THAT, or CURRENT), or a name you created whose value has previously been set with the LOCATION command.

Description:

A location_name can be used as an absolute block operand. This operand specifies a block which you specified and named earlier (or a block predefined by 6Edit itself).

If location_name is the first operand of a block expression, then the file to which the name refers becomes the enclosing block for the rest of the block expression. If location_name is not the first operand of a block expression, then it must refer to the same file as the enclosing block for the block expression.

If the first block operand in a block expression is not one of the following block operands, then the edit block is used as the enclosing block for the rest of the block expression:

```
PREVIOUS
NEXT
FILE
STRING
location_name
```

The location_name operand may not be modified by the following block options:

```
SKIP
FORWARD
BACKWARD
```

The default "point" option for the names as block operands is ALL OF.

## Special location_names

Several predefined names are reserved as special location_names. They can be used in block expressions just like any other location_name, however, their value is determined by 6Edit (you cannot explicitly assign a value to these names). As for all other names, they may be entered in either upper-, mixed-, or lower-case.

CURRENT

The CURRENT predefined name always specifies the current edit block. The edit block is set by the EDIT command, the NEXT and PREVIOUS block operands, and implicitly whenever the enclosing block for a block expression is in a different file than the current edit block. The name CURRENT can be abbreviated to CURR.

THAT

The THAT predefined name always specifies the current selected block. The selected block is the block of data operated upon by the most recent editing command.

HERE

The HERE predefined name always specifies an empty block. This block is located at the current file pointer location.

$LEFTOVER

The $LEFTOVER predefined name contains some portion of a single record of the source data that does not fit into the destination block on a copy operation. Once you have corrected the problem that caused the incomplete copy, you may use $LEFTOVER as the source for another copy operation.

$NOT_COPIED

The $NOT_COPIED predefined name contains the location of some portion of the source data that does not fit into the destination block on a copy operation. Once you have corrected the problem that caused the incomplete copy, you may use $NOT_COPIED (which is a location name but does not hold the actual data) to perform another copy operation.

Examples:

```
BO CURRENT
```

moves the file pointer to the beginning of the current edit block (usually to the beginning of the file being edited).

```
COPY THAT AFTER EO CURRENT
```

contains two special location_names as block operands: "THAT" and "CURRENT". Assuming the user has just selected a block, this COPY command copies that selected block to the end of the current edit block (usually to the end of the file being edited).

```
COPY L AFTER HERE
```

contains two names as block operands: "L" and "HERE". "L" is assumed to be a name created by the user with the LOCATION command, so that it names the location of a block (either in the current file or in any other file). This COPY command copies the block at location "L" to the current file pointer (cursor) position.

## Pattern-String Block Operand

Syntax:

```
{ PA[TTERN] string_expression | 'string' }
```

Parameters:

string_expression    is a string expression which may be empty, (i.e., have no characters in its value). It specifies the pattern to search for in the file.

string    is a quoted string of characters which may be empty (i.e., ''). It specifies the pattern to search for in the file. If a quote character is to be included in string as part of the pattern to search for, then it must appear twice (e.g. 'don''t' searches for the word "don't").

Description:

The pattern-string block operand finds a match for a pattern string (string_expression) in the enclosing block. The file pointer is moved to the matching data.

The search for a match is bounded by the enclosing block for the block expression. If a match is not found within the enclosing block, then 6Edit will not move the file pointer from its starting location.

The search for a match of the pattern string ignores record boundaries in the enclosing block up to the limit specified by the $MATCH_LIMIT predefined name.

The $CASE predefined name tells 6Edit how to treat alphabetic characters while searching for a match of the pattern string. If the current value of $CASE is ON (or any number >= 1), then file data characters are compared directly with characters from the pattern string. If the current value of $CASE is OFF (or any number <= 0), then lower-case alphabetic characters in both the file data characters and the pattern string are treated as though they were upper-case.

The only difference between the two forms of this block operand (use of string expression vs. 'string') is that the string_expression form allows you to combine strings using the concatenate operator, and/or use EQUALS-names to specify the pattern to search for. The 'string' form is easier to type, however, and can be used when the facilities of string expressions (see Section 4, String Expressions) are not needed.

The default "point", "direction", and "skip" options for the pattern-string operand are $POINT, $DIRECTION, and $SKIP, respectively. SKIP 0 tests for a match at the current cursor location. SKIP 1 moves forward/backward one position before beginning the search.

## Syntax of the Pattern String

The string_expression designates the pattern string. It tells 6Edit how to identify a "match" in the enclosing block.

Most characters in the pattern string specify values which must appear in identical form and identical order in the enclosing block in order to yield a "match." For instance, if you say:

```
     abc
```

then the only match possible is a string of three characters in the enclosing block, whose values are the lowercase ASCII characters a, b, and c, in that order. An exception to this "exact match" rule is made if the $CASE predefined name is OFF when the pattern search is made. (See Appendix A, Predefined Names.)

The pattern string is specified by a string expression. This allows non-displayable character values to be included in the string (using the $CONTROL-Value function).

Several of the $CONTROL functions have special meaning when they appear in the pattern string. Also, some additional $CONTROL functions are supported only when they appear in pattern strings; these are called "pattern functions."

## $CONTROL-Beginning of Match Pattern Function

Function Identifier: B

This pattern function should appear only once in a pattern string. It specifies what point in the matching file data is to be considered the beginning of the match. If $CONTROL-Beginning of Match is not included in a pattern string, the first character of the matching file data is considered the beginning of the match.

For example, assuming the $CONTROL character is percent (%):

```
COPY STRING 'memo' OVER REPEAT SELECT PATTERN 'my %Breport'
```

searches for the string 'my report' in the file. However, the beginning of the match is the "r" of "report": the block found and replaced by this command will be the word "report", but only if it appears after the word "my ".

## $CONTROL-End of Match Pattern Function

Function Identifier: E

This pattern function should appear only once in a pattern string. It specifies what point in the matching file data is to be considered the end of the match. If $CONTROL-End of Match is not included in a pattern string, the character immediately following the matching file data is considered the end of the match.

For example, assuming the $CONTROL character is percent (%):

```
CO STRING 'memo' OVER REPEAT SELECT PATTERN 'report%Es from this office'
```

searches for the string 'reports from this office' in the file. However, the end of the match is the "s" of "reports": the block replaced by this command will be the word "report", but only if it appears before the string "s from this office".

## $CONTROL-Question Pattern Function

Function Identifier: ? (question mark).

No parameters may be included in the $CONTROL-Question function.

The $CONTROL-Question function specifies that a sequence of any number of characters in the enclosing block (including 0) is considered a match.

For example, consider the following pattern string, assuming the $CONTROL character is the percent symbol (%).

```
    abcde%?fghijkl
```

This pattern string matches the following strings in the enclosing block:

```
    abcde1234fghijkl
    abcde'tfghijkl
    abcdefghijkl
    abcde  fghijkl
```

Multiple instances of the $CONTROL-Question function are permitted. For example,

```
     COPY STRING 'from%W(2)any%W(1)to' OVER 'to%?any%?from'
```

substitutes the COPY string for the OVER string for each string that matches the pattern of the OVER string, retaining the portions of the pattern string represent by %?s. Up to 10 instances of %W are permitted in a COPY or MOVE string; any number of %?s are permitted in the pattern string.

## $CONTROL-Value Function in the Pattern String

Function Identifier: V

If the $CONTROL-Value function appears in the pattern string, it may include one or two parameters. Assuming the $CONTROL character is the percent symbol (%):

- o '%V()' - If no parameters are included, a character in the enclosing block whose ASCII code is zero is considered a match. This is identical to %V(0).

- o '%V(m)' - If one parameter is included, it must be a decimal number; a character in the enclosing block whose ASCII code equals the number is considered a match.

- '%V(m,)' - If one parameter is included, it must be a decimal number. If a comma follows it, a character in the enclosing block whose ASCII code equals or is greater than the number is considered a match.

- '%V(m,n)' - If two parameters are included, they must be decimal numbers. The first specifies a minimum value, the second a maximum value. A character in the enclosing block whose ASCII code is between the two values, or equal to either, is considered a match.

- '%V(,n)' - If the second parameter is included but not the first, it must be a decimal number, and it must be preceded by a comma. A character in the enclosing block whose ASCII code is less than or equal to the number is considered a match. For example:

  ```
  DELETE REPEAT SELECT '%V(,31)'
  ```

  Searches forward in the file for ASCII control characters, and deletes any it finds. The ASCII control characters have decimal codes in the range 0 to 31. The pattern '%V(,31)' matches any single character whose ASCII decimal code is in that range.

## $CONTROL-Wildcard Pattern Function

Function Identifier: W

The $CONTROL-Wildcard function is similar to the $CONTROL-Question function (%?), except that it allows restrictions on the length of the string. Assuming the $CONTROL character is the percent symbol (%), the wildcard function is used in the following form:

```
%W(min,max)
```

The min and max parameters are the minimum and maximum lengths, respectively, of the wildcard string and are always numbers which represent character positions. Both parameters or a single parameter may be included.

%W is most logically used between two specific strings. For example:

```
DISPLAY 'from%W(4,10)any'
```

finds the next occurrence of "from" followed by "any" at least 4 positions later, but not more than 10 positions later in the enclosing block.

```
DISPLAY 'from%W(4,)any'
```

finds the next occurrence of "from" followed by "any" at least four positions later in the enclosing block, with no restriction on the maximum number of positions within the block.

```
DISPLAY 'from%W(,10)any'
```

finds the next occurrence of "from" followed by "any" not more than 10 positions later in the enclosing block. There is no restriction here on the minimum number of positions after which "any" should follow "from".

## POSITION Block Operand

Syntax:

```
PO[SITION[S]] [number                 ]
              [NU[MBER] string_expression]
```

Parameters:

number    is a decimal number between 1 and 2048, inclusive.

string_expression    is a string expression. It must evaluate to a number between 1 and 2048, inclusive.

Description:

This block operand actually has two forms, with different effects.

Skipping Over Existing Positions

The first form of the position operand moves the file pointer over existing positions in records:

```
    PO[SITION[S]]
```

moves the file pointer forward or backward in the current Edit Block some number of positions. The SKIP block option is used to specify the number of positions to move; if no SKIP option is used, 6Edit will move the file pointer by the number of positions in the $SKIP predefined name.

This form of the position operand only specifies existing positions in records; it will not extend a record. This means the following:

o If the file pointer is at the beginning of a record when a BACKWARD POSITION expression is given, it moves to the end-of-record boundary of the previous record in the enclosing block.

- o If the file pointer is in the last position of a record when a FORWARD POSITION expression is given, it moves to the end-of-record boundary of that record. From the end-of-record boundary of a record, this form of the position block operand moves the file pointer either (for BACKWARD) to the last position of that record, or (for FORWARD) to the first position of the next record.

## Moving To a Specific Position In a Record

The second form of the position operand includes a position number:

```
    PO[SITION]  { number | NU[MBER] string_expression }
```

moves the file pointer to the specified position in the current record. If the current record is too short, it is lengthened (the value of the $PAD predefined name is appended to the record repeatedly until the record is long enough).

The position to move to can be specified either as a number, or as a string expression. Position numbers range from 1 to 2048. To move the file pointer to the first position of a record, use "POSITION 1".

The second form of the position operand may not be modified by the following block options:

```
    SKIP
    FORWARD
    BACKWARD
```

The default "point" option for the second form of the position operand is $POINT.

Examples:

```
SKIP 5 POSITIONS
```

moves the file pointer forward five positions. If this moves the file pointer beyond the end of the current record, then the file pointer moves to the beginning of the next record, and continues movement from there.

```
BACKWARD SKIP 3 POSITIONS
```

moves the file pointer backward five positions. If the file pointer starts out less than five positions from the beginning of the record, it moves to the end of the previous record and continues movement from there.

```
POSITION 28
```

moves the file pointer to position 28 of the current record. If the current record is shorter than 28 characters long, it is extended to 28 characters, using the value of the $PAD predefined name (usually a space character) to extend the record.

```
POSITION NUMBER MYTAB .MINUS. 8
```

subtracts 8 from the current value of the user-created name MYTAB, and uses the result as the position number in the current record to move the file pointer to. If the current record is too short, it is extended.

## PROTECT Option

Syntax:

```
[DONT] PR[OTECT]  block_expression
```

Parameters:

|  |  |
|---|---|
| block_expression | must begin with a FILE block operand. |

Description:

When not preceded by DONT, PROTECT tells 6Edit that its block_expression is to be protected from any changes. The block_expression may not be altered or deleted, and no new records may be added.

DONT PROTECT tells 6Edit to allow modification of the block.

When specifying the edit block in the EDIT command, giving the PROTECT option to the block expression will essentially provide read-only access to the file as long as that file is the current edit block.

If an attempt is made to alter a PROTECTed block, 6Edit will reject the attempt and issue a beep at the terminal to inform the user of the attempted violation.

## RECORD Block Operand

Syntax:

```
RE[CORD[S]]
```

Description:

RECORD is a relative block operand. It specifies a block whose starting point is POSITION 1 of a record and whose ending point is POSITION 1 of the next record (i.e., the end-of-record boundary is included in the block).

When searching for this operand, the search for the starting point of the RECORD aims in the direction specified by any FORWARD or BACKWARD options for this operand; however, once the starting point is found, RECORD always specifies the record from that starting point forward to the end-of-record boundary, including that end-of-record boundary.

The default "point", "direction", and "skip" options for the record operand are $POINT, $DIRECTION, and $SKIP, respectively. SKIP 0 specifies the entire record that the cursor is positioned in. SKIP 1 specifies the next/previous (forward/backward) record.

Examples:

```
SKIP 20 RECORDS
```

moves the file pointer forward in the file, skipping over 20 records.

```
DELETE RECORD
```

deletes the entire record in which the file pointer currently lies.

```
COPY RECORD AFTER L
```

copies the record in which the file pointer currently lies to the location named "L", where "L" is assumed to be a name created by the user with the LOCATION command.

## Record Keys as Block Operands

Syntax:

{ numeric_key | KEY string_expression }

Parameters:

| | |
|---|---|
| numeric_key | is either: |

- o A string of 1 to 9 digits; a decimal point (period) may appear in the string such that there are 0-3 digits to the right of the decimal point and not more than 5 to the left.
- o A string of 1 to 9 digits with no decimal point for sequential files.

| | |
|---|---|
| string_expression | specifies the key to be used for a string-keyed file. |

Description:

A record key may be used as an absolute block operand. 6Edit moves the file pointer to the specified record in the enclosing block.

If an edit-keyed file is being used, numeric_key must be a number between 0 and 99999.999; a decimal point (period) may appear in the string, with 0 to 3 digits to the right of the decimal point. The numeric_key is interpreted as the actual record key of the record which is the block operand represented by numeric_key. Note that this record key need not actually exist in the file; if the specified record does not exist, then numeric_key represents an empty block at the specified record key. In any case, the record key must be within the range of record keys of the current enclosing block.

If a non-keyed (consecutive) file is being used, numeric_key must be a string of 1 to 9 digits with no decimal point. It is interpreted as the record number within the file of the record which is the block operand represented by numeric_key. The record number must be within the range of record numbers of the current enclosing block.

The record key operand may not be modified by the following block options:

    SKIP
    FORWARD
    BACKWARD

The default "point" option for the record key operand is $POINT.

Examples:

```
48
```

moves the file pointer to the record with key "48", when using an edit-keyed file. In a consecutive or unit-record file, specifying a numeric key moves the file pointer to the 48th record in the file.

```
COPY 52.1 AFTER HERE
```

copies the record with key "52.1" to the current file pointer (cursor) location. This command can only be used in an edit-keyed file, because the record key contains a decimal point.

```
COPY ST 'NEW RECORD%R' AFTER KEY 'HELLO'
```

creates a new record at "HELLO" in a string-keyed file.

## SKIP Option

Syntax:

```
SK[IP] number relative_block
```

Parameters:

number    is a string expression. It must evaluate to a number which is greater than or equal to zero.

relative_block    must be a relative block operand.

Description:

The SKIP option tells 6Edit to repeat the relative movement specified by relative_block some number of times. The number parameter specifies the "repeat count."

Remember, movement of the file pointer is limited to the bounds of the enclosing block. If you specify SKIP 10 RECORDS, but fewer than ten records remain in the enclosing block, the file pointer moves only to the end of the enclosing block.

If the repeat count evaluates to zero, then 6Edit assumes that the current location of the file pointer lies within the bounds of the destination block. For example, if the file pointer is currently at position 10 of a record, then END OF SKIP 0 RECORDS moves it to the end of that same record.

Example:

```
SKIP 10 RECORDS
```

6Edit moves the file pointer 10 records.

## STRING Block Operand

Syntax:

```
ST[RING] string_expression
```

Parameters:

string_expression    is any string expression. It may include $CONTROL-Record, $CONTROL-Value, $CONTROL-Wildcard, and $CONTROL-Question functions.

Description:

Any string specified as a string expression can be included in a block expression as a block operand.

The string value of the string expression is treated as a file; it becomes the enclosing block for the rest of the block expression.

The STRING operand must stand alone; you may not select a block within the value of the string expression, using the usual block operands. The string is usually treated just like a one-record consecutive file; however, by including the $CONTROL-Record function in the string expression, you can create a multi-record file.

Example:

```
COPY STRING 'novels and literary criticism' OVER REPEAT SELECT 'books'
```

searches forward in the file for any matches of the pattern "books", and replaces them with the string "novels and literary criticism".

```
COPY STRING '%W(2) %?' OVER '%? %?'
```

switches the current word in the file with the next one.

When used in a string block operand, the $CONTROL-Wildcard function must have exactly one parameter whose value is greater than zero. When the operand is eventually used by an AFTER or OVER command, the $CONTROL-Wildcard function is replaced by the string that matched a $CONTROL-Question or $CONTROL-Wildcard function in a pattern string for the command. The parameter specifies which particular $CONTROL-Question or $CONTROL-Wildcard function in the pattern is to be substituted, counting from the beginning of the pattern string at one. For example, a $CONTROL-Wildcard function with a parameter value of 3 would be replaced by the string that matched the second occurrence of "%?" in the pattern string "begin%?mid1%W(5,10)mid2%?end". If the parameter exceeds the number of wildcard strings in the pattern, the function is ignored.

The $CONTROL-Question function, which permits no parameters, can also be used in a string block operand, and is exactly equivalent to a $CONTROL-Wildcard function with a parameter value of one.

A maximum of ten combined $CONTROL-Wildcard and $CONTROL-Question functions may be used in any string block operand.

# Appendix A

# Predefined Names

This appendix lists all the names predefined in 6Edit, and describes their values.

## Using Predefined Names

Predefined names serve several purposes:

o They provide a shorthand notation for certain commonly-used values. Some of these values are constants, and some are changed by 6Edit internally.

o They allow you to give 6Edit certain operating values, and allow you to request that 6Edit perform certain functions. This method of telling 6Edit what to do is used only for those functions which do not change often. Thus, 6Edit equips you to control many small details.

You refer to these names by using 6Edit's substitution facility. You use most predefined names just like user-defined names.

Predefined names differ from user-defined names in two respects:

o You cannot assign values to some of the predefined names in the usual way, that is, with the assignment commands. Of those predefined names which cannot be assigned a value with the assignment commands, some have constant values, and 6Edit assigns new values to others internally.

o Of the predefined names to which you can assign a value with the assignment commands, most can only be assigned values by specific assignment commands. This is because different assignment commands assign different types of values to names. The legal assignment command for each predefined name is listed in the names' descriptions below. If you try to use the wrong command to assign a value to a particular name, 6Edit will reject the command.

Table A-1 classifies the names predefined in 6Edit.

Table A-1. Predefined Name Classifications

| Category | Names | | |
|---|---|---|---|
| Constant Values Provided by 6Edit | | | |
| | ANY | BO | OLD |
| | AO | DONT_PROTECT | ON |
| | BACKWARD | EO | PROTECT |
| | BIN10 | FORWARD | STRING |
| | BINHLF | NEW | |
| | BIN521 | OFF | |
| Values Set by 6Edit | | | |
| | $CONTEXT | $RECORDS_INSERTED | $FILERECORDS |
| | $MATCHES | $SITES | * $FILETYPE |
| | $RECORDS_DELETED | $FILEORG | |
| | * $FILETYPE can also be used to set the file type of files being edited. | | |
| Location Names Set By 6Edit | | | |
| | $LEFTOVER | THAT | |
| | $NOT_COPIED | HERE | |
| | CURRENT | | |
| Terminal Control | | | |
| | $AUTOTAB | $HSCROLL_MARGIN | $SCROLL |
| | $END_MARK | $INSERT | $TEXTEDIT |
| | $EZ_APPEND | $INTRO_1 | $VSCROLL |
| | $HSALL | $INTRO_2 | $VSCROLL_MARGIN |
| | $HSCROLL | $RECORDWRAP | $WORDWRAP |

Table A-1. Predefined Name Classifications (cont)

| Category | Names | | |
|---|---|---|---|
| Session Control | | | |
| | $INITIALIZE | $RESTORE | |
| Window Control | | | |
| | $COMMAND | $MIN_COMMAND | $WI_BORDER |
| | $MAX_COMMAND | $SCREEN | $WI_PERCENT |
| Record Control | | | |
| | $BY | $MIN_RECORD | $STRIP_BLANKS |
| | $KEY_GENERATION | $PAD | |
| | $MAX_RECORD | $REKEY | |
| Block Expression Defaults | | | |
| | $DIRECTION | $POINT | $SKIP |
| | $EXIST | $PROTECT | |
| Pattern Searching | | | |
| | $CASE | | |
| | $MATCH_LIMIT | | |
| Lexical Characters | | | |
| | $COMMENT | | |
| | $CONTROL | | |

## Predefined Names

The following table describes the 6Edit predefined names.

Table A-2. Predefined Names

| Name | Assignment Command and Description |
|---|---|
| $AUTOTAB | EQUALS a number between 0 and 254, inclusive. The number assigned to this name designates the "autotab position." A value greater than 1 implements automatic tabbing. Henceforth, the system always begins new records at the autotab position. To discontinue automatic tabbing, set $AUTOTAB to 0 or 1. Initially, $AUTOTAB is set to 0.<br><br>Note that you can also use "Escape Tab" to control automatic tabbing. Move the cursor to the desired autotab position, and type <ESC> <TAB>. To discontinue tabbing, move the cursor to the beginning of the record, and type <ESC> <TAB>. If $TEXTEDIT is set to ON, then any attempt to move into a record (old or new) in a position to the left of the autotab causes an autotab. |
| $BY | EQUALS an edit key from 0 to 65.534. The number assigned to this name is the default key increment for new records. The default is 10.0. $BY is set and displayed as an edit key. An edit key is a valid component of a string expression, but cannot be combined with other component values. |
| $CASE | EQUALS ON or OFF. This name tells 6Edit how to handle alphabetic case in pattern comparison. If $CASE equals ON, then data bytes must equal pattern bytes to be considered a match. If $CASE equals OFF, then data bytes must either equal pattern bytes, or if alphabetic, equal the pattern byte when converted to the opposite case, to be considered a match. Initially, $CASE is set ON. |

Table A-2. Predefined Names (cont)

| Name | Assignment Command and Description |
|---|---|
| $COMMAND | EQUALS ON or OFF.<br>This name controls which window 6Edit reads. If $COMMAND EQUALS OFF (the initial setting), 6Edit reads from the editing window. If $COMMAND EQUALS ON, 6Edit reads from the command window.<br><br>If you want to type several commands in succession, setting $COMMAND to ON ahead of time eliminates repeatedly telling 6Edit to move the cursor to the command window. When you finish typing the series of commands, set $COMMAND to OFF. 6Edit then resumes its normal operation. |
| $COMMENT | EQUALS one or two characters.<br>The character assigned to this name is used to mark the beginning of commentary in command lines. When this character appears in a command line but outside of a literal string, 6Edit ignores characters up to the next appearance of the $COMMENT value, or up to the end of the command line. Characters between the $COMMENT value are assumed to be commentary (arbitrary text meaningful only to you, not to 6Edit). Initially, the value of $COMMENT is the quotation mark character (").<br><br>See Section 1, Overview of 6Edit, for restrictions on the character assigned to $COMMENT and a description of the $COMMENT character's function. |
| $CONTEXT | (You cannot assign a value directly.)<br>This variable is a read-only value that returns the fid of the file used as the 6Edit context file. |

Table A-2. Predefined Names (cont)

| Name | Assignment Command and Description |
|---|---|
| $CONTROL | EQUALS one or two characters.<br>Use the character(s) assigned to this name for certain control functions. If you wish to include the character itself in command lines or string literals, you must enter it twice in succession. You can effectively disable all $CONTROL functions by assigning an empty string to $CONTROL. Initially, the value of $CONTROL is the percent character (%).<br><br>See Section 1, Overview of 6Edit, for restrictions on the character assigned to $CONTROL and for a description of the $CONTROL character's function. |
| $DIRECTION | EQUALS FORWARD or BACKWARD.<br>This name specifies the default direction in the file in which pattern searching and the SKIP option in block expressions are to proceed. The initial value is FORWARD. |
| $END_MARK | EQUALS a character string.<br>The string assigned to this name defines a line of text to be displayed at end-of-file in the editing window. The initial value is a blank string (no display). The maximum length of the string is 80 characters. Any characters can be used; 0 characters or all blanks turns off an existing display. For example:<br><br>`$END_MARK EQ 'File Ends Here'`<br><br>defines "File Ends Here" as the end-of-file marker.<br><br>`$END_MARK EQ ''`<br><br>resets the marker to blank. |

Table A-2. Predefined Names (cont)

| Name | Assignment Command and Description |
|---|---|
| $EXIST | EQUALS OLD, NEW, or ANY. This name controls the default existence specification for files. $EXIST can be set to suit your particular usage of 6Edit. For example, if most of your usage of 6Edit is to build new files, setting $EXIST to NEW allows you to open a file with EDIT FILE filename instead of EDIT NEW FILE filename. Conversely, if most of the files you edit already exist, you could set $EXIST to OLD (which is the default value). If you work with an equal number of existing and new files, setting $EXIST to ANY eliminates the need for either the OLD or NEW keyword with EDIT FILE. |
| $EZ_APPEND | EQUALS ON or OFF. This name controls what is needed to extend (insert a blank line at the end of) a file in the editing window. When $EZ_APPEND is set to ON, any attempt (such as a carriage return) to position the cursor to column 1 of the line below the last record in the file will extend the file. When this feature is OFF, the last record in the file must not be empty for the file to be extended. The default is OFF. |
| $FILEORG | (You cannot assign a value directly.) This variable is a read-only value that returns the file organization of the file currently being edited (corresponding to the CURR location variable). |
| $FILERECORDS | (You cannot assign a value directly.) This variable is a read-only value that returns the number of records in the file currently being edited (corresponding to the CURR location variable). |

Table A-2. Predefined Names (cont)

| Name | Assignment Command and Description |
|---|---|
| $FILETYPE | EQUALS a 2-character string. The characters assigned to this name are used to change the file type for the current file, if it is changeable, and to set the default for any NEW FILEs created thereafter. (The current file is the file being edited [corresponding to the CURR location variable].) This variable may also be used to return the file type of the file currently being edited. Note that the default for NEW FILEs may not be the same as the value displayed by SHOW $FILETYPE, since that command always displays the current file's type. DELETE WINDOW can be used to obtain access to the default value or to set the default without changing the current file. |
| $HSALL | EQUALS ON or OFF. This name controls how lines are to be scrolled when horizontal scrolling is in effect. When $HSALL is ON, all the lines of a screen editing window are scrolled when a horizontal shift occurs. A value of OFF specifies to scroll only the record containing the cursor. The default is OFF. |
| $HSCROLL | EQUALS a number between 0 and half the screen width, inclusive. The number assigned to this name specifies how many columns to scroll the editing window when trying to move the cursor into a horizontal scrolling margin. Initially, $HSCROLL is 0.<br><br>If $HSCROLL is greater than 0, that is, if horizontal scrolling is enabled, 6Edit scrolls the the editing window horizontally so that the character on which the cursor lies is at the center of the editing window. |

| Table A-2. Predefined Names (cont) |
|---|
| **Name** Assignment Command and Description |
| $HSCROLL_MARGIN<br><br>EQUALS a number between 0 and half the screen width, inclusive. The number assigned to this name specifies the number of columns in the horizontal scrolling margins. A designated number of columns represents these margins at the right and left of the editing window.<br><br>Initially, $HSCROLL_MARGIN is set to 0; there are no horizontal scrolling margins, and horizontal scrolling is disabled. Setting $HSCROLL_MARGIN to a particular number of columns forces t he window to scroll horizontally when the cursor reaches the designated column, for example the fourth column from the left or right edge of the screen when $HSCROLL_MARGIN is set to 4. |
| $INITIALIZE<br><br>EQUALS a string expression consisting of 6Edit commands separated by the $CONTROL-record function.<br>The commands assigned to this name are interpreted immediately following restoration of the context file at startup, thus allowing "active" commands such as SHOW or IBEX commands to be included in saved context files. Note that any $CONTROL-functions in the commands other than those that separate the commands must use the $CONTROL-literal function to express the $CONTROL identifier (e.g., the command SHOW STRING '%V(33)A' would be specified for use in $INITIALIZE as 'SH STRING ''%%V(33)A'''.<br><br>At startup, until the first editing command is given, 6Edit remains in serial mode ($SCREEN EQ 0), using the window set in M$6E (the default is ME, which uses UC01). So, until the first editing command that needs to build an editing window, any operations that affect the building of windows will continue to have an effect when the windows are finally built. For example, $INITIALIZE can be used to move the IBEX window to the bottom of the screen by setting it to<br>'!LDEV UC98,POSITION=BOTTOM,WL=1%R!ADJUST M$6E UC01'<br>The second command (!ADJUST M$6E UC01) tells 6Edit to use all the rest of the screen for editing, instead of its default, which is "all-but-the-top-line". |

Table A-2. Predefined Names (cont)

| Name | Assignment Command and Description |
|---|---|
| $INSERT | EQUALS ON or OFF.<br>This name controls the insertion mode. Initially, the insertion mode is disabled by its initial setting ($INSERT EQUALS OFF). In this case, the characters typed replace the characters beneath the cursor, unless the cursor is at the end of the "insertion window" established with the Escape > or Escape J functions. When the insertion mode is enabled ($INSERT EQUALS ON), characters typed are always inserted at the cursor location; they never replace characters at the cursor.<br><br>Note that the insertion mode is also enabled with the Escape ' function, and disabled with either the Escape M or Escape O function. |
| $INTRO_1 | EQUALS one or two tokens. The first may be a $KEY function. The second, if present, must be a character.<br>The characters assigned to this name represent the primary introducer for KEYIN names (IMP functions). Initially, 6Edit uses the escape character <ESC> as the primary introducer. |
| $INTRO_2 | EQUALS one or two tokens. The first may be a $KEY function. The second, if present, must be a character.<br>The characters assigned to this name represent the secondary introducer for KEYIN names (IMP functions). Initially, 6Edit assumes no secondary introducer. If your KEYIN names use a secondary introducer, you must tell 6Edit (by setting $INTRO_2) before you use the KEYIN command. |

Table A-2. Predefined Names (cont)

| Name | Assignment Command and Description |
|---|---|
| $KEY_GENERATION | EQUALS BIN10, BINHLF, BIN521, or STRING. This name specifies how the FEP is to generate keys for new records inserted by means other than 6Edit's COPY or MOVE commands. If $KEY_GENERATION is set to BIN10 (the default), the key increment is divided by 10 repeatedly to find a usable increment. For the 6Edit default key increment of 10, inserted records would be separated by one of 10.000, 1.000, .100, .010, or .001 whichever works first. If $KEY_GENERATION is set to BINHLF, the key increment is halved repeatedly. For the key increment of 10, the inserted records would be separated by 10.000, 5.000, 2.500, 1.250, .625, etc. If $KEY_GENERATION is set to BIN521, the key increment is divided by 5, 2.5, and 2, and is then repeated. For the key increment of 10, the inserted records would be separated by 10.000, 5.000, 2.000, 1.000, .500, .200, .100, etc. If $KEY_GENERATION is set to STRING, insertion of new records is not permitted. STRING overrides any specification for string-keyed files.<br><br>In general, the BIN10 default may permit inserting a larger number of records in the same spot than would BINHLF or BIN521. Those values may be useful for inserting one or two records at a time and will tend to have a more even distribution of key numbers, but with more fractional keys. |
| $LEFTOVER | (You cannot assign value directly.) Whenever the source data for a copy operation does not fit into the destination block and $REKEY does not permit automatic rekeying, some portion of a single record of the source data may be saved as the value of the $LEFTOVER name. (6Edit does this internally when such an error occurs.) 6Edit tells you that it has stored some of the source data in $LEFTOVER. Once you have corrected the original problem, you may use $LEFTOVER as the source for another copy operation.<br><br>When such an error occurs, either $LEFTOVER or $NOT_COPIED, or both, may be set. If $LEFTOVER is set, the data contained therein no longer remains in the file. If $NOT_COPIED is set, it simply refers to the location of the data that remains in the file. |

Table A-2. Predefined Names (cont)

| Name | Assignment Command and Description |
|---|---|
| $MATCH_LIMIT | |
| | EQUALS a number.<br>The value assigned to this name specifies the maximum number of records which the match of a pattern expression can span. Initially, the value of $MATCH_LIMIT is 1; the entire pattern expression must match data residing in, at most, one record. If, for example, you were to set $MATCH_LIMIT to three, then all pattern expressions may match up to a three record span anywhere in the enclosing block.<br><br>$MATCH_LIMIT limits any $CONTROL pattern-match function except $CONTROL-Record; $MATCH_LIMIT applies independently to each pattern substring between any $CONTROL-Record functions.<br><br>When $MATCH_LIMIT is greater than 1, adjacent strings may match the pattern even if the strings span a record boundary. For instance, if the pattern is "reports from this office" and the file contains these strings split between two records:<br><br>................reports from<br>this office.................<br><br>then the occurrence in the file is considered a match, even though the pattern string contains a blank between "from" and "the", while the occurrence in the file contains only a record boundary there.<br><br>For example, consider the following pattern string:<br><br>'abc%?def'<br><br>If, when this is evaluated:<br><br>$MATCH_LIMIT = 1 - Both "abc" and "def" must be found in the same record for a match to be found.<br><br>$MATCH_LIMIT = 5 - "abc" must match in one record or two adjacent ones, and "def" must match in one record or two adjacent ones; however, there may be up to three records between the "a" record and the "f" record, so that the entire match spans five records. |

Table A-2. Predefined Names (cont)

| Name | Assignment Command and Description |
|---|---|
| $MATCHES | (You cannot assign value directly.) After each pattern search, 6Edit assigns a value to $MATCHES. This value represents the number of matches found for patterns in this evaluation of the block expression. Note that if a block expression contains more than one pattern operand, the value assigned to $MATCHES will be the sum of the matches found for both patterns. |
| $MAX_COMMAND | EQUALS a number. This name tells 6Edit how large to expand the command window when it displays messages (i.e. error or HELP messages). The number assigned to this name is the number of lines on the terminal screen of the largest command window. The minimum value is 1; the maximum value is the height (in lines) of your terminal screen. Initially, $MAX_COMMAND is set to approximately half the height of the terminal screen. |
| $MAX_RECORD | EQUALS a number between 0 and 2048, inclusive. The value of this name is the maximum length, in bytes, of records written to files in 6Edit. If you change a record, and its length is greater than $MAX_RECORD, 6Edit truncates bytes from the end of the record so that it is $MAX_RECORD bytes long. Initially, $MAX_RECORD is set to 2048. |
| $MIN_COMMAND | EQUALS a number. This name tells 6Edit how small to diminish the size of the command window when you type commands, or when you are editing in the editing window. The number assigned to this name is the number of lines in the smallest command window. The minimum value is 0; the maximum value is the height (in lines) of your terminal screen. Initially, $MIN_COMMAND is set to 2. |

Table A-2. Predefined Names (cont)

| Name | Assignment Command and Description |
|---|---|
| $MIN_RECORD | Equals a number between 0 and 2048, inclusive. The value of this name is the minimum length, in bytes, of records written to files by 6Edit. If you change a record, and its length is less than $MIN_RECORD, 6Edit appends the value of the $PAD name to the record repeatedly, until it is $MIN_RECORD bytes long. Initially, $MIN_RECORD is set to 1. In this case, all records will be at least 1 byte long. |
| $NOT_COPIED | (You cannot assign value directly.) When the source data for a copy operation does not fit into the destination block, the location of some portion of the source data may be saved as the $NOT_COPIED name. (6Edit does this internally when such an error occurs.) $NOT_COPIED is a location name; it does not hold the actual data, but rather remembers the location of that part of the source block that was not copied to the destination due to error. 6Edit tells you when it has saved the locations of some of the source data in $NOT_COPIED. Once you correct the original problem, you may use $NOT_COPIED as the source for another copy operation.<br><br>When such an error occurs, either $LEFTOVER or $NOT_COPIED, or both, may be set. If $LEFTOVER is set, the data contained therein no longer remains in the file. If $NOT_COPIED is set, it simply refers to the location of the data that remains in the file. |
| $PAD | EQUALS a character string. The value of this name is used to "pad" records which are shorter than the minimum specified by $MIN_RECORD. The minimum length of the value of $PAD is 1; the maximum length is 100. Initially, $PAD is set to a single space character. |

Table A-2. Predefined Names (cont)

| Name | Assignment Command and Description |
|---|---|
| $POINT | EQUALS AO, BO, or EO<br>This name specifies the default file position option for block expressions. AO (ALL OF, the default) causes the entire block to be selected. For example, a SELECT clause may move the cursor to the beginning of the block; END OF THAT would move it to the other end. BO (BEGINNING OF) causes only the beginning of the block to be selected; EO (END OF) causes only the end of the block to be selected.<br>Note: $POINT must not be set to EO for commands such as SHOW which select records by key, since EO RE is actually the beginning of the record following the desired one. |
| $PROTECT | EQUALS PROTECT or DONT_PROTECT<br>This name controls the default protect status for files. $PROTECT can be set to suit your particular usage of 6Edit. For example, if you usually just read files without updating them (and want them protected), setting $PROTECT to PROTECT allows you to open a file with EDIT FILE filename instead of EDIT PROTECT FILE filename. Conversely, if you usually update files rather than just read them, setting $PROTECT to DONT_PROTECT (the default) allows you to open a file with EDIT FILE filename instead of EDIT DONT PROTECT FILE filename. |
| $RECORDS_DELETED | (You cannot assign value directly.)<br>After each DELETE, OVER, or MOVE command, 6Edit assigns a value to $RECORDS_DELETED. This value represents the number of records deleted from the file by the command. Note that only entire records are counted; if only a portion of a record is deleted, it will not be included in the $RECORDS_DELETED value. |

Table A-2. Predefined Names (cont)

| Name | Assignment Command and Description |
|---|---|
| $RECORDS_INSERTED | (You cannot assign value directly.)<br>Following each AFTER or OVER command, 6Edit assigns a value to $RECORDS_INSERTED. This value represents the number of new records inserted into the file by the command. |
| $RECORDWRAP | EQUALS ON or OFF.<br>This name controls the record wrap mode. Initially, the record wrap mode is disabled ($RECORDWRAP EQUALS OFF). In this case, any attempt to move the cursor beyond the ends of a record is ignored. When the record wrap mode is enabled ($RECORDWRAP EQUALS ON), the actions of following input editing functions are modified when the cursor is in the editing window.<br><br>&lt;ESC&gt; &lt;D&gt; or &lt;CNTL-H&gt;<br><br>(BS)<br>If the cursor is at the first position of a record, it will move to just beyond the last position of the previous record.<br><br>&lt;ESC&gt; &lt;C&gt; or &lt;CNTL-R&gt;<br><br>(DC2)<br>If the cursor is beyond the end of a record, it will move to the first position of the next record.<br><br>&lt;ESC&gt; &lt;V&gt;<br><br>The search for the character typed after the &lt;V&gt; is not limited to the current record; however, it is limited to the records currently displayed in the editing window.<br><br>&lt;CNTL-W&gt;<br><br>(ETB)<br>Moves to the next or previous word in the record or, if necessary, in the next or previous record. |

Table A-2. Predefined Names (cont)

| Name | Assignment Command and Description |
|---|---|
| $REKEY | EQUALS a number between 0 and 1000. The value of this name controls automatic rekeying of the file. If $REKEY is set to zero, 6Edit never automatically rekeys the file. If $REKEY is greater than zero, 6Edit tries to rekey as small a range up to its value in first the forward direction and then backwards as will permit the rekeyed records plus the inserted ones to be separated by the current increment value (from a BY clause in the command, or the $BY predefined name). If neither direction is successful, the entire file is rekeyed instead. (No attempt is made to find a combination of forward and backward records.) Sometimes, if the number of records involved is small, a smaller increment is used. The initial value is 100. |
| $RESTORE | EQUALS a character string. The string assigned to this name is an IBEX command. The IBEX command should not begin with an exclamation character (!). When 6Edit exits, the current value of $RESTORE is passed to IBEX to execute. Usually, this IBEX command causes your normal terminal environment to be restored. IBEX commands used frequently are the IMP command, to process an IMP source or object file, and the XEQ command, to perform several functions. If $RESTORE has no value assigned to it, (e.g., $RESTORE EQUALS ''), then no such action is taken when 6Edit exits. |
| $SCREEN | EQUALS ON, OFF, or an integer. This name controls the size of the editing screen. Setting $SCREEN to 0 (OFF) requests serial editing by removing from the screen all editing windows. At startup, 6Edit sets $SCREEN OFF if the user is operating in batch mode or if the user's terminal profile indicates that screen editing is impossible. For an online CRT user running with $SCREEN OFF, setting $SCREEN to ON (or 1) restores the terminal to screen editing with editing windows in the same proportions as before. The size of the IBEX window can be increased (from 1 line) by setting $SCREEN. |

Table A-2. Predefined Names (cont)

| Name | Assignment Command and Description |
|---|---|
| | See Section 2, Serial Editing, for more information. If screen editing is not possible, $SCREEN may only be set to 0. |
| $SCROLL | EQUALS ON or OFF.<br>This name controls scrolling in the command window. The default is ON. With scrolling on, consecutive entries in the command window are scrolled upward. With scrolling off, the asterisk (*) prompt remains at the top of the command window; consecutive entries are cleared when they reach the editing line. OFF is particularly useful on terminals where scrolling windows that do not extend to the bottom of the screen are awkward. |
| $SITES | (You cannot assign value directly.)<br>After each command, 6Edit assigns a value to $SITES. This value represents the number of blocks found during evaluation of the command. If the command did not include the REPEAT keyword, the $SITES will be set to either 0 or 1; only if REPEAT is used will $SITES be greater than 1. |
| $SKIP | EQUALS a number from 0 to 131071.<br>This name specifies the number of times to repeat some relative movement of the file pointer within the specified block. The value assigned to $SKIP is used if no SKIP option is present in the PATTERN, POSITION, or RECORD block operands. The default is zero. |

Table A-2. Predefined Names (cont)

| Name | Assignment Command and Description |
|---|---|
| $STRIP_BLANKS | EQUALS ON or OFF. This name controls the stripping of trailing blanks from records written by 6Edit. If you change a record and $STRIP_BLANKS is set to ON, any space characters at the end of the record (which are not followed by any non-space characters) are removed from the record. Initially, $STRIP_BLANKS is set ON. |
| $TEXTEDIT | EQUALS ON or OFF. This name controls the spacing between words when joining or splitting text lines. With $TEXTEDIT ON, a blank space is automatically added between words (the last word of the first line and the first word of the next line) when joining two lines. When splitting a line at the blank space before a word, the blank is automatically removed so that the new (next) line starts with the word, not the blank. The default is OFF. |
| $VSCROLL | EQUALS a number between 0 and the length of the screen, inclusive. The number assigned to this name specifies how many lines to scroll the editing window when trying to move the cursor into a vertical scrolling margin.<br><br>When you reset $VSCROLL to 0, 6Edit scrolls the editing window so that the record in which the cursor lies is at the center of the editing window. Initially, $VSCROLL is set to 0. |
| $VSCROLL_MARGIN | EQUALS a number between 0 and half the length of the screen, inclusive. The number assigned to this name specifies the number of lines in |

Table A-2. Predefined Names (cont)

| Name | Assignment Command and Description |
|---|---|
| | the vertical scrolling margins. A designated number of lines represents these margins at the top and bottom of the editing window. When you try to move the cursor into one of these margins, the window scrolls in the opposite direction; the cursor will not remain in the vertical scrolling margin.<br><br>Setting $VSCROLL_MARGIN to a particular number of lines forces the window to scroll when the cursor reaches the designated line, for example the sixth line from the top or the bottom when $VSCROLL_MARGIN is set to 6. Initially, $VSCROLL_MARGIN is set to approximately 25% of the height of the editing window. |
| $WI_BORDER | EQUALS any single character.<br>This name specifies the character used for the border that divides the command and editing windows. The hyphen (-) is the default. $WI_BORDER should be used in a context file, or set before any editing commands are used in a 6Edit session. If set to a null string (''), no border is used. This is useful with $MIN_COMMAND EQ 0, which then allows the entire screen to be used for editing. (If a PC is being used, this also lets the PC Terminal Facility save lines that scroll off the top in its history buffer.) |
| $WI_PERCENT | EQUALS a number from 1 to 99.<br>The value assigned to this name is used in creation of windows. Rather than specifying the size of the new window to create, this name specifies the percentage of space remaining in the old window. For example, to create three even-sized windows in a 21-line editing space using the LINES option, "NE WI;NE WI LI 14;NE WI LI 7" would be used (presuming that no editing windows existed previously). Using the PERCENT option, "NE WI;NE WI 67;NE WI 33" would be used; but using the defaults, "$WI_PERCENT EQ 33;NE WI;NE WI;NE WI" would suffice. The default for $WI_PERCENT is 50. |

Table A-2. Predefined Names (cont)

| Name | Assignment Command and Description |
|---|---|
| $WORDWRAP | EQUALS a number between 0 and 254, inclusive. The number assigned to this name specifies the right margin for the "wordwrap mode." A value greater than 11 implements the wordwrap mode. The wordwrap mode removes a word which crosses the right margin from its original record, and replaces it at the beginning of a newly-inserted record. To discontinue the wordwrap mode, set $WORDWRAP to any value between 0 and 11.<br><br>Note that you can also use the Escape ^ function to control the wordwrap mode. Move the cursor to the intended right margin, and type <ESC> <^>. To discontinue this wordwrap mode, repeat this sequence, positioning the cursor at the beginning of the record.<br><br>The wordwrap mode allows you to enter textual material without concern with line length. As you type the text, 6Edit only allows you to type as far as the right margin. When you cross that margin, the system automatically begins a new record. If a word crosses the right margin, it moves that word to the new record also. You may thus type without ever touching the Return key. Initially, the $WORDWRAP mode is disabled. |
| ANY | (You cannot assign value.) This is a predefined constant; the value is always 3. |
| AO | (You cannot assign value.) This is a predefined constant; the value is always 1. |
| BACKWARD | (You cannot assign value.) This is a predefined constant; the value is always 2. |

Table A-2. Predefined Names (cont)

| Name | Assignment Command and Description |
|---|---|
| BIN10 | (You cannot assign value.) This is a predefined constant; the value is always 4. |
| BINHLF | (You cannot assign value.) This is a predefined constant; the value is always 5. |
| BIN521 | (You cannot assign value.) This is a predefined constant; the value is always 6. |
| BO | (You cannot assign value.) This is a predefined constant; the value is always 2. |
| CURRENT | (You cannot assign value directly.) This name points to the current edit block. You set it using the EDIT command, and use it when block expressions specify no explicit enclosing block. You can use CURRENT to refer to the entire edit block for movement, data selection, etc. (See Section 5, Block Expressions in 6Edit.) 6Edit accepts the "CURR" abbreviation. |
| DONT_PROTECT | (You cannot assign value.) This is a predefined constant; the value is always 2. |

Table A-2. Predefined Names (cont)

| Name | Assignment Command and Description |
|---|---|
| EO | (You cannot assign value.) This is a predefined constant; the value is always 3. |
| FORWARD | (You cannot assign value.) This is a predefined constant; the value is always 1. |
| HERE | (You cannot assign value directly.) HERE points to the current location of 6Edit's file pointer. 6Edit updates the value of HERE automatically after performing each editing command. |
| NEW | (You cannot assign value.) This is a predefined constant; the value is always 1. |
| OFF | (You cannot assign value.) This is a predefined constant; the value is always 0. |
| OLD | (You cannot assign value.) This is a predefined constant; the value is always 2. |

Table A-2. Predefined Names (cont)

| Name | Assignment Command and Description |
|---|---|
| ON | (You cannot assign value.) This is a predefined constant; the value is always 1. |
| PROTECT | (You cannot assign value.) This is a predefined constant; the value is always 1. |
| STRING | (You cannot assign value.) This is a predefined constant; the value is always 7. |
| THAT | (You cannot assign value directly.) This name points to the current selected block, the block operated on by the last editing command. 6Edit updates the value of this name automatically after each editing command. If the last editing command deleted its operand, THAT points to the empty block where the deleted operand resided. This empty block can then be used as the destination of a new insertion replacing the deleted operand. |

# Appendix B

# Predefined String Functions

This appendix lists the string functions supported by 6Edit and their descriptions. (See Section 4, String Expressions, for more information on string expressions.)

Table B-1. String Functions

| Name | Description |
|---|---|
| $ABS(expression) | Returns the absolute value of the expression. The expression must be numeric. |
| $ACCT | Returns the account portion of the current logon. |
| $CMDVAR(command_variable_name) | Returns the value of a command variable defined in IBEX with !LET, which is compatible with the variables defined in 6Edit with EQUALS. $CMDVAR(command_variable_name) allows access to the values of variables defined in IBEX just as %command_variable_name is used in IBEX. |
| $CNTL(expression) | Returns the expression string as though the control key had been depressed. For example, $CNTL('[W') is equivalent to <ESC>\|\|<CNTL-W>. |

Table B-1. String Functions (cont)

| Name | Description |
|---|---|
| $DATE | Returns the date in the form YYMMDD. An optional parameter TYPE='type' may be supplied. The 'type' expression may be EXT, ANS, or LOCAL. For example:<br><br>!OUTPUT $DATE (TYPE='ANS')<br>841203<br><br>!OUTPUT $DATE (TYPE='EXT')<br>DEC 03 '84<br><br>!OUTPUT $DATE (TYPE='LOCAL')<br>12/03/84 |
| $DAY | Returns SUN, MON, TUE, WED, THU, FRI, or SAT. |
| $DIR | Returns the current directory account. |
| $EOF | Returns 1 if an end-of-file has been encountered on M$UC. Returns 0 otherwise. |
| $FID_ASN(expression) | Expression is evaluated to a string that is treated as a file name. $FID_ASN performs a call to M$FID, but does not try to OPEN the FID. If the M$FID call was successful, the assignment type (ASN) is returned. Examples are FILE, DEVICE, and COMGROUP. If the file does not exist or cannot be accessed, then an error message is displayed, and the command is aborted. |

Table B-1. String Functions (cont)

| Name | Description |
|---|---|
| $FID_EXIST(expression) | Expression is evaluated as a string that is treated as a file name. If an attempt by IBEX to open the file yields any result other than "file does not exist", then $FID_EXIST returns a 1. This is no guarantee that the user may access the file. Returns a 0 otherwise. If the file does not exist or cannot be accessed, then an error message is displayed, and the command is aborted. |
| $FID_GRANS(expression) | Expression is evaluated as a string that is treated as a file name. $FID_GRANS specifies the size of the file in total number of granules. If the file does not exist or cannot be accessed, then an error message is displayed, and the command is aborted. |
| $FID_NGAVAL(expression) | Expression is evaluated as a string that is treated as a file name. $FID_NGAVAL specifies the size of the file in number of unused granules in the file. If the file does not exist or cannot be accessed, then an error message is displayed, and the command is aborted. |
| $FID_ORG(expression) | Expression is evaluated as a string that is treated as a file name. CONSEC, KEYED, RANDOM, INDEXED, RELATIVE, IDS, etc. If the file does not exist or cannot be accessed, then an error message is displayed, and the command is aborted. |
| $FID_RECS(expression) | Expression is evaluated as a string that is treated as a file name. $FID_RECS specifies the size of the file in number of records. If the file does not exist or cannot be accessed, then an error message is displayed, and the command is aborted. |

Table B-1. String Functions (cont)

| Name | Description |
|---|---|
| $FID_TYPE(expression) | $FID_TYPE returns the two-character file type of the file. If the file does not exist or cannot be accessed, then an error message is displayed, and the command is aborted. |
| $FID_UGRANS(expression) | Expression is evaluated as a string that is treated as a file name. $FID_UGRANS specifies the size of the file in number of Ugranules. If the file does not exist or cannot be accessed, then an error message is displayed, and the command is aborted. |
| $FLAG(fname) | $FLAG returns YES if a particular IBEX flag is set; otherwise it returns NO. The fname may be any of the following:<br>PROTECT DRIBBLE AC/CEPT BRO/ADCAST<br>LIST ECHO AC/CEPT SEND<br>COMMENT NOTIFY AC/CEPT AN/NOUNCE |
| $HSET | Returns the home packset name. |
| $INDEX(expression 1,expression 2[,expression 3[,expression 4]]) | Expressions 1 and 2 are evaluated to strings of characters; expressions 3 and 4, if given, must be numeric. $INDEX returns the index into expression 1 where expression 2 was found. If expression 2 is not found, the length of expression 1 is returned. If both 3 and 4 are omitted, then all of expression 1 is searched for expression 2. If expression 3 is included, then it specifies the beginning position for the search within expression 1. If expression 4 is included, then it specifies the ending position for the search within expression 1. The first character in the string is position 0. |

Table B-1. String Functions (cont)

| Name | Description |
|---|---|
| $INPUT(expression) | Prompts through the command window (M$ME) with expression. Returns a reply from the user. If the user's reply includes leading or trailing blanks, these are stripped from the input. When the user desires the blanks not be stripped, the input string must be enclosed in quotes (which will be stripped). Doubled quotes within a quote string are translated to a single quote in the resultant string. |
| $KEY(keyname) | Specifies the keyin name (string_expression_1) in a KEYIN command; can be used in an EQUALS command to define a KEYIN introducer. $KEY can also be used anywhere else with a standard ASCII control character mnemonic or a hex value as the keyname parameter, and is simply an alternate way to express the indicated character.<br><br>See the KEYIN command in Section 3 for more information. |
| $KEYIN(expression) | Sends expression to the system consoles. $KEYIN suspends execution until it receives a reply from the system console. $KEYIN then takes on the value of the reply. If the user's reply includes leading or trailing blanks, these are stripped from the input. When the user desires the blanks not be stripped, the input string must be enclosed in quotes (which will be stripped). Doubled quotes within a quote string are translated to a single quote in the resultant string. |
| $LASTBATCH | Returns the sysid of the last job issued via the BATCH command. |

Table B-1. String Functions (cont)

| Name | Description |
|---|---|
| $LENGTH(expression) | Returns the number of characters in expression. The length of a string expression is one greater than the last character used in that expression. |
| $LOC(expression) | Returns the string expression converted to lower case. |
| $MAX(exp1,exp2,...expn) | Returns the value of the largest expression. All expressions must be numeric. |
| $MIN(exp1,exp2,...expn) | Returns the value of the smallest expression. All expressions must be numeric. |
| $MOD(exp1, exp2) | Returns the arithmetic modulus of exp1 and exp2. Both exp1 and exp2 must be numeric. |
| $MODE | Returns ONLINE, BATCH, GHOST, or TP, whatever mode in which the user currently operates. |
| $NAME | Returns the name portion of the logon. |

Table B-1. String Functions (cont)

| Name | Description |
|---|---|
| $PRIV_ACTIVE(pname) | Returns YES if a particular IBEX PRIV bit is turned on. The pname may be FMSEC, etc. |
| $PRIV_AUTH(pname) | Returns YES if a particular IBEX PRIV bit is authorized. The pname may be FMSEC, etc. |
| $REM(exp1, exp2) | Returns the remainder of exp1 and exp2. Both exp1 and exp2 must be numeric; exp2 may not be 0. |
| $RERUN | Returns 1 if this job is being rerun automatically by the system. Otherwise, it returns 0. |
| $SEARCH(expression 1,expression 2[,expression 3[,expression 4]]) | Expressions 1 and 2 are evaluated to strings of characters; expressions 3 and 4, if given, must be numeric. $SEARCH checks each character in expression 1 for membership in expression 2. The index of the first match is returned; if not found the length of expression 1 is returned. If expression 3 is included, then it specifies the beginning position for the search within expression 1. If expression 4 is included, then it specifies the ending position for the search within expression 1. The first character in the string is position 0. |

Table B-1. String Functions (cont)

| Name | Description |
|---|---|
| $SITE | Returns the textual site identification. |
| $SUBSTR(expression 1[,expression 2[,expression 3]]) | Expression 1 is evaluated to a string of characters; expressions 2 and 3, if given, must be numeric. A character string is extracted from expression 1 beginning at the nth position as specified by expression 2 with a length specified by expression 3. If expression 2 is not specified then the first character is assumed. If expression 3 is not specified then the length of expression 1 minus expression 2 is assumed. The first character in the string is position 0. |
| $SWITCH(expression) | Returns the value 0 or 1 of the specified SWITCH. The expression must evaluate to a numeric value between 0 and 35. Switches are set by the IBEX SWITCH command. |
| $SYSID | Returns the user's sysid. |
| $TERM_FEP | Returns node of the user's FEP. |
| $TERM_LINE | Returns the channel to which the user is connected. |

Table B-1. String Functions (cont)

| Name | Description |
|---|---|
| $TERM_PROFILE | Current terminal profile. |
| $TERM_SPEED | Returns communication speed in bits per second, e.g. 300, 1200, etc.<br><br>Note: The TERM functions will return an error message if mode is not ONLINE. |
| $TIME | Returns the current time in the form HHMM. An optional parameter TYPE='type' may be supplied. The 'type' expression may be EXT, ANS, or LOCAL. For example:<br><br>`!OUTPUT $TIME (TYPE='ANS')`<br>`1224`<br><br>`!OUTPUT $TIME (TYPE='EXT')`<br>`12:24`<br><br>`!OUTPUT $TIME (TYPE='LOCAL')`<br>`12:24` |
| $UPC(expression) | Returns the string expression converted to upper case. |
| $VERIFY(expression 1,expression 2[,expression 3[,expression 4]]) | Expressions 1 and 2 are evaluated to strings of characters; expressions 3 and 4, if given, must be numeric. Each character of expression 1 is checked for membership in expression 2. $VERIFY returns the first non-match; if not found, it returns the length |

Table B-1. String Functions (cont)

| Name | Description |
|---|---|
| | of the expression. If expression 3 is included, then it specifies the beginning position for the search within expression 1. If expression 4 is included, then it specifies the ending position for the search within expression 1. The first character in the string is position 0. |
| $VERSION | Returns the current version of the CP-6 system. |
| $WOO | Returns the current workstation of origin. |

# Appendix C

# Context Files in 6Edit

This appendix illustrates several context files for 6Edit. The function descriptions in Table C-2 remain constant regardless of the context file in use. Their key strokes, however, vary with each context file.

The files described here set up the 6Edit editor for their various terminal environments.

A context file defines:

o The keys and key sequences used on your terminal to invoke 6Edit's special input editing functions (such as "move cursor to command window"), as well as many common commands.

o Names for the ASCII control characters used to invoke 6Edit's special input editing functions. These are helpful when you use the KEYIN command interactively to define keys on your terminal.

o Synonyms for several commonly-used phrases in the 6Edit command language.

Table C-1 describes the standard names and their values. These names apply to all context files.

Table C-1. Standard Names in a Context File

| Command | Description |
|---|---|
| CMD EQ '%V(3)' ... | |
| | A series of commands similar to this one defines the activation characters which invoke 6Edit's special input editing functions. This allows you to use the name "CMD" in string expressions; 6Edit uses the correct binary value.<br><br>In addition, the value of the ASCII Escape and carriage return control characters is given a name, so that it can be used more easily in 6Edit expressions. |

Table C-1. Standard Names in a Context File (cont)

| Command | Description |
|---|---|
| | These names are most useful in KEYIN commands, which redefine the keys of your terminal keyboard. For example, to tell 6Edit that the sequence of keys "Escape Z" should cause the cursor to move to the end of the next record, use the following KEYIN command:<br><br>KEYIN ESCII'Z' IS ESCII'B'\|\|ESCII'N' |
| SYNONYM REKEY COPY BLOCK OVER BLOCK | REKEY can be used to rekey (renumber) a file. It can only be used when you are currently editing the entire file to be rekeyed. By default, REKEY uses 10 as the key increment. To rekey the file by a specific key increment, append the BY option to the REKEY synonym; for example:<br><br>REKEY BY 1<br><br>or<br><br>REKEY BY 2.5 |
| SYNONYM SHL DELETE THAT EACH BO RECORD THRU POSITION | SHL can be used to shift a block of records left some number of positions. Before using SHL, you must select a block of text. (See Section 1, Overview of 6Edit.) Note that the block you select must begin and end at the beginning of records; that is, the beginning of the block must be at the beginning of the first record to be shifted, and the end of the block must be at the beginning of the record following the last record to be shifted.<br><br>Once you have selected the block, go to the command window, and type:<br><br>SHL 4<br><br>This shifts the block of records left 4 positions. To shift a different number of positions, specify that number after the SHL synonym. |

Table C-1. Standard Names in a Context File (cont)

| Command | Description |
|---|---|
| SYNONYM EACH REPEAT SELECT RECORD SELECT | |
| | EACH can be used in a command to cause an editing operation to be repeated on each record in a block. For example, to insert a string at the beginning of each record in the selected block, type: `COPY ST 'xxx' AFTER THAT EACH BO RECORD` Or, to delete positions 10 through 13 from each record in the selected block, type: `DELETE THAT EACH PO 10 THRU PO 13` |

Figure C-1 illustrates the standard context file for 6Edit. The standard context file can be used with any terminal. The name of this file is:

    :6EDIT_CONTEXT.:LIBRARY

```
"----- Standard synonyms and values
SYNONYM REKEY IS COPY CURRENT OVER CURRENT
SYNONYM EACH IS REPEAT SELECT RECORD SELECT
SYNONYM SHL IS DELETE REPEAT SELECT RECORD SELECT BO RECORD THRU POSITION
ESC EQ '%V(27)';    CR EQ '%V(13)'
"----- Special input activation characters for 6Edit
CMD EQ '%V(3)'
"----- Cursor movement keyins
KEYIN '%V(21)' IS ESC||'A'                        "BACKWARD 1 <CNTL-U>
KEYIN '%V(22)' IS ESC||'B'                        "FORWARD 1 <CNTL-V>
KEYIN ESC||'W' IS CMD||'BACKWARD SKIP 8 BO RECORDS'||CR
                                                   "BACKWARD 8 <ESC> <W>
KEYIN ESC||'E' IS CMD||'SKIP 8 BO RECORDS'||CR   "FORWARD 8 <ESC> <E>
KEYIN ESC||'A' IS CMD||'BACKWARD SKIP 20 BO RECORDS'||CR
                                                    "BACKWARD 20 <ESC> <A>
KEYIN ESC||'L' IS CMD||'SKIP 20 BO RECORDS'||CR  "FORWARD 20 <ESC> <L>
KEYIN ESC||'P' IS CMD||'BACKWARD SKIP 1 ''''%V(8)%V(27)>'
                                                 "BACKWARD PATTERN <ESC> <P>
KEYIN '%V(16)' IS CMD||'SKIP 1 ''''%V(8)%V(27)>' "FORWARD PATTERN <CNTL-P>
KEYIN ESC||'B' IS CMD||'BO CURRENT'||CR          "<ESC> <B>
KEYIN ESC||'F' IS CMD||'EO CURRENT'||CR          "<ESC> <F>
KEYIN ESC||'.' IS CMD||'L'||CR                   "<ESC> <.>
"----- Block editing keyins
KEYIN '%V(7)'  IS CMD                             "<CNTL-G>
KEYIN '%V(20)' IS CMD||'THRU'||CR                "<CNTL-T>
KEYIN '%V(1)'  IS CMD||'ADJUST'||CR              "<CNTL-A>
KEYIN ESC||'1' IS CMD||'COPY THAT'||CR           "<ESC> <1>
KEYIN ESC||'!' IS CMD||'MOVE THAT'||CR           "<ESC> <!>
KEYIN ESC||'3' IS CMD||'COPY RECORD'||CR         "<ESC> <3>
KEYIN ESC||'#' IS CMD||'MOVE RECORD'||CR         "<ESC> <#>
KEYIN ESC||'4' IS CMD||'COPY L AFTER HERE'||CR   "<ESC> <4>
KEYIN ESC||'$' IS CMD||'MOVE L AFTER HERE'||CR   "<ESC> <$>
KEYIN ESC||'/' IS CMD||'AFTER HERE'||CR          "<ESC> </>
KEYIN ESC||'@' IS CMD||'OVER THAT'||CR           "<ESC> <@>
KEYIN ESC||'*' IS CMD||'DELETE THAT'||CR         "<ESC> <*>
KEYIN ESC||'_' IS CMD||'EDIT THAT'||CR           "<ESC> <_>
```

Figure C-1. Standard Context File (cont. next page)

```
KEYIN ESC||'[' IS CMD||'EDIT PREVIOUS'||CR       "<ESC> <[>
KEYIN ESC||']' IS CMD||'EDIT NEXT'||CR           "<ESC> <]>
KEYIN ESC||':' IS CMD||'LOCATION L IS THAT'||CR  "<ESC> <:>
KEYIN ESC||'=' IS CMD||'COPY STRING '''' OVER REPEAT SELECT '''''||;
  '%V(27)%V(13)%V(27)V''%V(18)%V(27)>'           "SUBSTITUTE <ESC> <=>
```

Figure C-1. Standard Context File

Figure C-2 illustrates the context file for the DEC VT 100 terminal. The name of this file is:

```
:6EDIT_CONTEXT_DECVT100.:LIBRARY
```

```
"----- Standard synonyms and values
SYNONYM REKEY IS COPY CURRENT OVER CURRENT
SYNONYM EACH IS REPEAT SELECT RECORD SELECT
SYNONYM SHL IS DELETE REPEAT SELECT RECORD SELECT BO RECORD THRU POSITION
ESC EQ <ESC>;    CR EQ <CR>
$END_MARK EQ '   ---end---'
"----- Parameters
$INTRO_1 EQ ESC
"----- Special input activation characters for 6Edit
CMD EQ '%V(3)'
"----- Cursor movement keyins
KEYIN <UPARROW> IS ESC||'A'                         "BACKWARD 1 <Up Arrow>
KEYIN <DNARROW> IS ESC||'B'                         "FORWARD 1 <Down Arrow>
KEYIN <RTARROW> IS ESC||'C'                         "RIGHT 1 <Right Arrow>
KEYIN <LTARROW> IS ESC||'D'                         "LEFT 1 <Left Arrow>
KEYIN <PF3> IS <ESC>||'8A'                          "BACKWARD 8 <PF3>
KEYIN <PF4> IS <ESC>||'8B'                          "FORWARD 8 <PF4>
KEYIN <PF1> IS <ESC>||'20A'                         "BACKWARD 20 <PF1>
KEYIN <PF2> IS <ESC>||'20B'                         "FORWARD 20 <PF2>
KEYIN <UF10> IS CMD||'BA SK 1 PA $INPUT(''Search Pattern:'')'||CR
                                                    "BACKWARD PATTERN <Pad ,>
KEYIN <ENTER> IS CMD||'SKIP 1 PA $INPUT(''Search Pattern:'')'||CR
                                                  "FORWARD PATTERN <Pad Enter>
KEYIN <UF5> IS CMD||'BO CURRENT'||CR                "<Pad 5>
KEYIN <UF6> IS CMD||'EO CURRENT'||CR                "<Pad 6>
KEYIN <CNTL-L> IS CMD||'L'||CR                      "<CNTL-L>
"----- Block editing keyins
KEYIN <CNTL-A> IS CMD                               "<CNTL-A>
KEYIN <UF0> IS CMD||'THRU'||CR                      "<Pad 0>
KEYIN <UF11> IS CMD||'ADJUST'||CR                   "<Pad ->
KEYIN <UF7> IS CMD||'COPY THAT'||CR                 "<Pad 7>
KEYIN ESC||'!' IS CMD||'MOVE THAT'||CR              "<ESC> <!>
KEYIN <UF4> IS CMD||'COPY RECORD'||CR               "<Pad 4>
KEYIN ESC||'#' IS CMD||'MOVE RECORD'||CR            "<ESC> <#>
KEYIN <UF9> IS CMD||'COPY L AFTER HERE'||CR         "<Pad 9>
KEYIN <ESC>||<@> IS CMD||'MOVE L AFTER HERE'||CR    "<ESC><@>
KEYIN <UF8> IS CMD||'AFTER HERE'||CR                "<Pad 8>
KEYIN <ESC>||<$> IS CMD||'OVER THAT'||CR            "<ESC><$>
KEYIN <ESC>||<%> IS CMD||'DELETE THAT'||CR          "<ESC><%>
KEYIN <UF2> IS CMD||'EDIT THAT'||CR                 "<Pad 2>
```

Figure C-2. DECVT100 Context File (cont. next page)

```
KEYIN <UF1> IS CMD||'EDIT PREVIOUS'||CR          "<Pad 1>
KEYIN <UF3> IS CMD||'EDIT NEXT'||CR              "<Pad 3>
KEYIN <UF12> IS CMD||'LOCATION L IS THAT'||CR    "<Pad .>
KEYIN <CNTL-T> IS CMD||;
 'CO ST $INPUT(''New string:'') OVER REPE SE PA $INPUT(''Old string:'')'||;
 CR                                              "SUBSTITUTE <CNTL-T>
KEYIN <LF> IS ESC||<LF>                          "SPLIT <LF>
```

Figure C-2. DECVT100 Context File

Figure C-3 illustrates an example context file for an IBM PC-compatible personal computer using the PC Terminal Facility (PCT) with the PCTX364 profile. The name of this file is:

    :6EDIT_CONTEXT_PCTX364.:LIBRARY

```
!SET M$LL UC,ORG=T    "Allows RESTORE command to work
"*****     Replace TOGKEYPAD with KEYPADON if on new release of PCT  *****
$INITIALIZE EQ 'SHOW STRING ''%V(27)_TOGKEYPAD%V(27)\''%R'||"Keypad mode";
  '!SET M$LL UC,ORG=T%R'||              "Make $RESTORE work properly";
  '!EJECT UC01%R'||                     "Blank screen at 1st entry to 6Edit";
  '!EJECT%R'                            "Reposition cursor to bottom
"*****
$MIN_COMMAND EQ 2
"*****     Replace TOGKEYPAD with KEYPADOFF if on new release of PCT  *****
$RESTORE EQ 'OUTPUT '''||<ESC>||'_TOGKEYPAD'||<ESC>||'\'''
"*****
$VSCROLL_MARGIN EQ 0
$VSCROLL EQ 1
"*****     Replace TOGKEYPAD with KEYPADON if on new release of PCT  *****
SHOW STRING <ESC>||'_TOGKEYPAD'||<ESC>||'\'
"*****
"----- Standard synonyms and values
SYNONYM EACH IS REPEAT SELECT RECORD SELECT
SYNONYM REKEY IS COPY CURRENT OVER CURRENT
SYNONYM SHL IS DELETE REPEAT SELECT RECORD SELECT BO RECORD THRU POSITION
ESC EQ <ESC>;    CR EQ <CR>
"----- Custom synonyms and values
$END_MARK EQ '   ---end---'
"----- Special input activation characters for 6Edit
CMD EQ '%V(3)'
"----- Cursor movement keyins
KEYIN <CNTL-G>  IS CMD                           "COMMAND <CNTL-G>
KEYIN <UPARROW> IS ESC||'A'                      "BACKWARD 1 <Up Arrow>
KEYIN <DNARROW> IS ESC||'B'                      "FORWARD 1 <Down Arrow>
KEYIN <RTARROW> IS <DC2>                         "RIGHT 1 <Right Arrow>
KEYIN <LTARROW> IS <BS>                          "LEFT 1 <Left Arrow>
KEYIN <SCROLL_SEG_UP> IS <ESC>||'20A'            "BACKWARD 20 <Page Up>
KEYIN <SCROLL_SEG_DOWN> IS <ESC>||'20B'          "FORWARD 20 <Page Down>
KEYIN <SU15> IS <ESC>||'8A'                      "BACKWARD 8 <CNTL-Page Up>
KEYIN <SU14> IS <ESC>||'8B'                     "FORWARD 8 <CNTL-Page Down>
KEYIN <HOME> IS CMD||'BO CURRENT'||CR            "<Home>
KEYIN <SU16> IS CMD||'BO THAT'||CR               "<CNTL-Home>
KEYIN <UF11> IS CMD||'EO CURRENT'||CR            "<End>
KEYIN <SU11> IS CMD||'EO THAT'||CR               "<CNTL-End>
```

Figure C-3. PCTX364 Context File (cont. next page)

```
KEYIN <FF>   IS CMD||'L'||CR                       "<CNTL-L>
"----- File positioning
KEYIN <DLE>  IS CMD||'EDIT PREVIOUS'||CR           "<CNTL-P>
KEYIN <SO>   IS CMD||'EDIT NEXT'||CR               "<CNTL-N>
"----- Pattern definition and searching
KEYIN <UF1>  IS CMD||'BA SK 1 PA $INPUT(''Search Pattern:'')'||CR
                                                  "BACKWARD PATTERN <CNTL-F1>
KEYIN <UF2>  IS CMD||'SKIP 1 PA $INPUT(''Search Pattern:'')'||CR
                                                  "FORWARD PATTERN <CNTL-F2>
KEYIN <F11>  IS CMD||'BA SK 1 PA '''''||CR        "BACKWARD PATTERN <ALT-F1>
KEYIN <F12>  IS CMD||'SKIP 1 PA '''''||CR         "FORWARD PATTERN <ALT-F2>
"----- Block editing keyins
KEYIN <LF>   IS ESC||<LF> IN EDITING WINDOW        "SPLIT <LF>
KEYIN <F9>   IS ESC||<FF> IN EDITING WINDOW        "DELETE RECORD <F9>
KEYIN <F1>   IS CMD||'COPY RECORD'||CR             "<F1>
KEYIN <F2>   IS CMD||'MOVE RECORD'||CR             "<F2>
KEYIN <F3>   IS CMD||'LOCATION L IS THAT'||CR      "<F3>
KEYIN <F4>   IS CMD||'COPY L AFTER HERE'||CR       "<F4>
KEYIN <F5>   IS CMD||'MOVE L AFTER HERE'||CR       "<F5>
KEYIN <SOH>  IS CMD||'AFTER HERE'||CR              "<CNTL-A>
KEYIN <STX>  IS CMD||'THRU'||CR                    "<CNTL-B>
KEYIN <ETX>  IS CMD||'COPY THAT'||CR               "<CNTL-C>
KEYIN <EOT>  IS CMD||'DELETE THAT'||CR             "<CNTL-D>
KEYIN <ENQ>  IS CMD||'EDIT THAT'||CR               "<CNTL-E>
KEYIN <VT>   IS CMD||'ADJUST'||CR                  "<CNTL-K>
KEYIN <SI>   IS CMD||'OVER HERE'||CR               "<CNTL-O>
KEYIN <CNTL-T> IS CMD||;
 'CO ST $INPUT(''New string:'') OVER REPE SE PA $INPUT(''Old string:'')'||;
 CR                                                "SUBSTITUTE <CNTL-T>
KEYIN <SYN>  IS CMD||'MOVE THAT'||CR               "<CNTL-V>
```

Figure C-3. PCTX364 Context File

Figure C-4 illustrates the context file for the Honeywell VIP7205 terminal. The name of this file is:

    :6EDIT_CONTEXT_VIP7205.:LIBRARY

```
"----- Standard synonyms and values
SYNONYM REKEY IS COPY CURRENT OVER CURRENT
SYNONYM EACH IS REPEAT SELECT RECORDS SELECT
SYNONYM SHL IS DELETE REPEAT SELECT RECORD SELECT BO RECORD THRU POSITION
ESC EQ '%V(27)';    CR EQ '%V(13)'
"----- Special input activation characters for 6Edit
CMD EQ '%V(3)'
"----- Cursor movement keyins
KEYIN ESC||'A' IS ESC||'A'                   "BACKWARD 1 <Up Arrow>
KEYIN ESC||'B' IS ESC||'B'                   "FORWARD 1 <Down Arrow>
KEYIN ESC||'5' IS CMD||'BACKWARD SKIP 8 BO RECORDS'||CR
                                             "BACKWARD 8 <SHIFT-F2>
KEYIN ESC||'2' IS CMD||'SKIP 8 BO RECORDS'||CR "FORWARD 8 <F2>
KEYIN ESC||'7' IS CMD||'BACKWARD SKIP 20 BO RECORDS'||CR
                                             "BACKWARD 20 <SHIFT-F3>
KEYIN ESC||'6' IS CMD||'SKIP 20 BO RECORDS'||CR "FORWARD 20 <F3>
KEYIN ESC||'1' IS CMD||'BACKWARD SKIP 1 ''''%V(8)%V(27)>'
                                             "BACKWARD PATTERN <SHIFT-F1>
KEYIN ESC||'0' IS CMD||'SKIP 1 ''''%V(8)%V(27)>' "FORWARD PATTERN <F1>
KEYIN ESC||'T' IS CMD||'BO CURRENT'||CR      "<ESC> <T>
KEYIN ESC||'F' IS CMD||'EO CURRENT'||CR      "<ESC> <F>
KEYIN ESC||'L' IS CMD||'L'||CR               "<ESC> <L>
"----- Block editing keyins
KEYIN ESC||''' IS CMD                        "<CLR>
KEYIN ESC||'i' IS CMD||'THRU'||CR            "<XMIT>
KEYIN '%V(1)'  IS CMD||'ADJUST'||CR          "<CTRL-A>
KEYIN ESC||':' IS CMD||'COPY THAT'||CR       "<F5>
KEYIN ESC||';' IS CMD||'MOVE THAT'||CR       "<SHIFT-F5>
"KEYIN IS CMD||'COPY RECORD'||CR"            "<CLR> CO RE <RETURN>
"KEYIN IS CMD||'MOVE RECORD'||CR"            "<CLR> MO RE <RETURN>
"KEYIN IS CMD||'COPY L AFTER HERE'||CR"      "<CLR> CO L AF HERE <RETURN>
"KEYIN IS CMD||'MOVE L AFTER HERE'||CR"      "<CLR> MO L AF HERE <RETURN>
KEYIN ESC||'<' IS CMD||'AFTER HERE'||CR      "<F6>
"KEYIN IS CMD||'OVER THAT'||CR"              "<CLR> OV THAT <RETURN>
KEYIN ESC||'=' IS CMD||'DELETE THAT'||CR     "<SHIFT-F6>
"KEYIN IS CMD||'EDIT THAT'||CR"              "<CLR> ED THAT <RETURN>
```

Figure C-4. VIP7205 Context File (cont. next page)

```
"KEYIN IS CMD||'EDIT PREVIOUS'||CR"          "<CLR> ED PREV <RETURN>
"KEYIN IS CMD||'EDIT NEXT'||CR"              "<CLR> ED NEXT <RETURN>
KEYIN ESC||'9' IS CMD||'LOCATION L IS THAT'||CR "<SHIFT-F4>
KEYIN ESC||'8' IS CMD||'COPY STRING '''' OVER REPEAT SELECT '''''||;
  '%V(27)%V(13)%V(27)V''%V(18)%V(27)>'         "SUBSTITUTE <F4>
```

Figure C-4. VIP7205 Context File

Figure C-5 illustrates the context file for Honeywell VIP7801 or VIP7802 terminals. The name of this file is:

:6EDIT_CONTEXT_VIP7801.:LIBRARY

```
"----- Standard synonyms and values
SYNONYM REKEY IS COPY CURRENT OVER CURRENT
SYNONYM EACH IS REPEAT SELECT RECORD SELECT
SYNONYM SHL IS DELETE REPEAT SELECT RECORD SELECT BO RECORD THRU POSITION
ESC EQ '%V(27)';  CR EQ '%V(13)';  $INTRO_2 EQ '%V(27)['  "VIP78xx's only
"----- Special input activation characters for 6Edit
CMD EQ '%V(3)'
"----- Cursor movement keyins
KEYIN ESC||'A' IS ESC||'A'                      "BACKWARD 1 <Up Arrow>
KEYIN ESC||'B' IS ESC||'B'                      "FORWARD 1 <Down Arrow>
KEYIN ESC||'[I' IS CMD||'BACKWARD SKIP 8 BO RECORDS'||CR
                                                "BACKWARD 8 <DEL/CHAR/INS>
KEYIN ESC||'[L' IS CMD||'SKIP 8 BO RECORDS'||CR "FORWARD 8 <DEL/LINE/INS>
KEYIN ESC||'[P' IS CMD||'BACKWARD SKIP 20 BO RECORDS'||CR
                                          "BACKWARD 20 <SHIFT-DEL/CHAR/INS>
KEYIN ESC||'[M' IS CMD||'SKIP 20 BO RECORDS'||CR
                                          "FORWARD 20 <SHIFT-DEL/LINE/INS>
KEYIN ESC||'1' IS CMD||'BACKWARD SKIP 1 ''''%V(8)%V(27)>'
                                              "BACKWARD PATTERN <SHIFT-F1>
KEYIN ESC||'0' IS CMD||'SKIP 1 ''''%V(8)%V(27)>' "FORWARD PATTERN <F1>
KEYIN ESC||'5' IS CMD||'BO CURRENT'||CR         "<SHIFT-F2>
KEYIN ESC||'2' IS CMD||'EO CURRENT'||CR         "<F2>
KEYIN ESC||'L' IS CMD||'L'||CR                  "<ESC> <L>
"----- Block editing keyins
KEYIN ESC||'e' IS CMD                           "<CLEAR/RESET>
KEYIN ESC||'i' IS CMD||'THRU'||CR               "<TRANSMIT>
KEYIN ESC||''' IS CMD||'ADJUST'||CR             "<SHIFT-CLEAR/RESET>
KEYIN ESC||':' IS CMD||'COPY THAT'||CR          "<F5>
KEYIN ESC||';' IS CMD||'MOVE THAT'||CR          "<SHIFT-F5>
"KEYIN IS CMD||'COPY RECORD'||CR"             "<CLEAR/RESET> CO RE <RETURN>
"KEYIN IS CMD||'MOVE RECORD'||CR"             "<CLEAR/RESET> MO RE <RETURN>
"KEYIN IS CMD||'COPY L AFTER HERE'||CR"
                                       "<CLEAR/RESET> CO L AF HERE <RETURN>
"KEYIN IS CMD||'MOVE L AFTER HERE'||CR"
                                       "<CLEAR/RESET> MO L AF HERE <RETURN>
KEYIN ESC||'<' IS CMD||'AFTER HERE'||CR    "<F6>
"KEYIN IS CMD||'OVER THAT'||CR"            "<CLEAR/RESET> OV THAT <RETURN>
KEYIN ESC||'=' IS CMD||'DELETE THAT'||CR   "<SHIFT-F6>
"KEYIN IS CMD||'EDIT THAT'||CR"            "<CLEAR/RESET> ED THAT <RETURN>
```

Figure C-5. VIP7801/VIP7802 Context File (cont. next page)

```
"KEYIN IS CMD||'EDIT PREVIOUS'||CR"       "<CLEAR/RESET> ED PREV <RETURN>
"KEYIN IS CMD||'EDIT NEXT'||CR"           "<CLEAR/RESET> ED NEXT <RETURN>
KEYIN ESC||'9' IS CMD||'LOCATION L IS THAT'||CR "<SHIFT-F4>
KEYIN ESC||'8' IS CMD||'COPY STRING '''' OVER REPEAT SELECT '''''||;
  '%V(27)%V(13)%V(27)V''%V(18)%V(27)>'          "SUBSTITUTE <F4>
```

Figure C-5. VIP7801/VIP7802 Context File

Figure C-6 illustrates the context file for the Zenith Z19 terminal. The name of this file is:

    :6EDIT_CONTEXT_ZENZ19.:LIBRARY

```
"----- Standard synonyms and values
SYNONYM REKEY IS COPY CURRENT OVER CURRENT
SYNONYM EACH IS REPEAT SELECT RECORD SELECT
SYNONYM SHL IS DELETE REPEAT SELECT RECORD SELECT BO RECORD THRU POSITION
ESC EQ '%V(27)';    CR EQ '%V(13)'
"----- Special input activation characters for 6Edit
CMD EQ '%V(3)'
"----- Cursor movement keyins
KEYIN ESC||'A' IS ESC||'A'                      "BACKWARD 1 <Up Arrow>
KEYIN ESC||'B' IS ESC||'B'                      "FORWARD 1 <Down Arrow>
KEYIN ESC||'!' IS CMD||'BACKWARD SKIP 8 BO RECORDS'||CR
                                                "BACKWARD 8 <ESC> <!>
KEYIN ESC||'1' IS CMD||'SKIP 8 BO RECORDS'||CR  "FORWARD 8 <ESC> <1>
"KEYIN IS CMD||'BACKWARD SKIP 20 BO RECORDS'||CR"
                                     "BACKWARD 20 <F1> BA SK 20 RE <RETURN>
KEYIN ESC||'2' IS CMD||'SKIP 20 BO RECORDS'||CR "FORWARD 20 <ESC> <2>
KEYIN ESC||'V' IS CMD||'BACKWARD SKIP 1 ''''%V(8)' "BACKWARD PATTERN <F4>
KEYIN ESC||'W' IS CMD||'SKIP 1 ''''%V(8)'       "FORWARD PATTERN <F5>
KEYIN ESC||'O' IS CMD||'BO CURRENT'||CR         "<ESC> <O>
KEYIN ESC||'F' IS CMD||'EO CURRENT'||CR         "<ESC> <F>
KEYIN ESC||'.' IS CMD||'L'||CR                  "<ESC> <.>
"----- Block editing keyins
KEYIN ESC||'S' IS CMD                           "<F1>
KEYIN ESC||'H' IS CMD||'THRU'||CR               "<Home>
KEYIN ESC||'T' IS CMD||'ADJUST'||CR             "<F2>
KEYIN ESC||'R' IS CMD||'COPY THAT'||CR          "<Gray>
KEYIN ESC||'P' IS CMD||'MOVE THAT'||CR          "<Blue>
KEYIN ESC||'3' IS CMD||'COPY RECORD'||CR        "<ESC> <3>
KEYIN ESC||'#' IS CMD||'MOVE RECORD'||CR        "<ESC> <#>
KEYIN ESC||'4' IS CMD||'COPY L AFTER HERE'||CR  "<ESC> <4>
KEYIN ESC||'$' IS CMD||'MOVE L AFTER HERE'||CR  "<ESC> <$>
KEYIN ESC||'Q' IS CMD||'AFTER HERE'||CR         "<Red>
"KEYIN IS CMD||'OVER THAT'||CR"                 "<F1> OV THAT <RETURN>
KEYIN ESC||'E' IS CMD||'DELETE THAT'||CR        "<SHIFT-ERASE>
KEYIN ESC||'_' IS CMD||'EDIT THAT'||CR          "<ESC> <_>
KEYIN ESC||'[' IS CMD||'EDIT PREVIOUS'||CR      "<ESC> <[>
KEYIN ESC||']' IS CMD||'EDIT NEXT'||CR          "<ESC> <]>
KEYIN ESC||':' IS CMD||'LOCATION L IS THAT'||CR "<ESC> <:>
KEYIN ESC||'U' IS CMD||'COPY STRING '''' OVER REPEAT SELECT '''''||;
  '%V(27)%V(13)%V(27)V''%V(18)%V(27)>'          "SUBSTITUTE <F3>
```

Figure C-6. Zenith Z19 Context File

Table C-2 describes the definitions for common functions of the context files listed in this appendix supported by 6Edit. These function descriptions apply to each context file; their key invocations differ, depending on the terminal used. Refer to the appropriate figure in this appendix for the key strokes corresponding to that context file.

Table C-2. Common Functions Defined By a Context File

| Function | Description |
|---|---|
| ADJUST | Move the cursor to the other end of the block currently being selected. If you type the ADJUST function repeatedly, the cursor bounces back and forth from one end of the current selected block to the other. While the cursor is at an end of the selected block, moving the cursor changes the specification of the block. |
| AFTER HERE | Insert the source block at the current cursor location. You must have set the source block previously with some COPY or MOVE command. |
| BACKWARD 8 RECORDS | Move the cursor backward (up) 8 records. |
| BACKWARD 20 RECORDS | Move the cursor backward 20 records. |
| Backward Pattern Search | Search backward for a match of a pattern. This resembles the Pattern Search key, searching backward in the file from the current cursor location. |

Table C-2. Common Functions Defined By a Context File (cont)

| Function | Description |
|---|---|
| BO CURRENT | Move the cursor to the beginning of the current edit block. |
| Command Window | Move the cursor to the command window. |
| COPY L AFTER HERE | Insert the block named L at the current cursor location. You should have given a block the name L previously, using the LOCATION command. |
| COPY RECORD | The record on which the cursor is sitting becomes the source block for the next AFTER/OVER command. |
| COPY THAT | The selected block becomes the source block for the next AFTER/OVER operation. Before typing the COPY key you should select a block of data. (See Section 1, Overview of 6Edit, subsection 'Specifying a Block.') |
| DELETE THAT | Delete the selected block. Before typing the DELETE key you should select the block of data to be deleted. |

Table C-2. Common Functions Defined By a Context File (cont)

| Function | Description |
|---|---|
| EDIT NEXT | EDIT NEXT advances you to the next edit block. This allows you to refer to the block specification following your current edit block. |
| EDIT PREVIOUS | EDIT PREVIOUS restores the old edit block from a previous EDIT command. This becomes the new edit block. EDIT THAT and EDIT PREVIOUS can be used together to make editing easier.<br><br>For example, to change all occurrences of "giraffe" to "horse" only in a specific group of records, use the following procedure:<br><br>o Select the group of records: move the cursor to the beginning of the first record in the group, type THRU, then move the cursor to the beginning of the record after the last record in the group.<br><br>o Type EDIT THAT.<br><br>o Type the Substitute key, (see below), fill in the new and old strings, and perform the operation.<br><br>o Type EDIT PREVIOUS. |
| EDIT THAT | The selected block becomes the new edit block. 6Edit remembers the old edit block so it can restore it later. The selected block must begin and end at the beginning of the records you want to include in the new edit block. This key is useful before using the Substitute key to limit the string substitutions to the records in the selected block. |

Table C-2. Common Functions Defined By a Context File (cont)

| Function | Description |
|---|---|
| EO CURRENT | Move the cursor to the end of the current edit block. |
| FORWARD 8 RECORDS | Move the cursor forward (down) 8 records. |
| FORWARD 20 RECORDS | Move the cursor forward 20 records (about a screen-full). |
| Move to L | Move the cursor to the block named L. You should have previously located a block, giving it the name L. This is done with Location L function. Having named a block L, you can return to it at any time (in the same editing session) using L. |
| LOCATION L IS THAT | Gives the name L to the location of the selected block. |
| MOVE L AFTER HERE | Insert the block named L at the current cursor location. This is the same as the "COPY L AFTER HERE" key except that after inserting the block at the cursor location the block named L is deleted. (Note that after this deletion, the name L still refers to the same location in the file.) You should have given a block the name L previously, using the LOCATION command. |

Table C-2. Common Functions Defined By a Context File (cont)

| Function | Description |
|---|---|
| MOVE RECORD | The record on which the cursor is sitting becomes the source block for the next AFTER/OVER command. Also, after performing the next AFTER/OVER command, 6Edit deletes this record. |
| MOVE THAT | The selected block becomes the source block for the next AFTER/OVER operation. This is the same as the COPY key, except that after the next AFTER/OVER operation the source block is deleted. |
| OVER THAT | Replace the selected block with the source block. Before typing the OVER key you should select the block of data which is to be replaced. |
| Pattern Search | Search forward for a match of a pattern. When you type the Pattern Search key, the cursor moves to the command window, and the one of following message displays appears.<br><br>o The message<br><br>SKIP 1 ''<br><br>appears and the cursor moves to the second apostrophe ('), ready for you to type a pattern string. After you have typed the pattern string, press the Return key.<br><br>o Or, the message<br><br>Pattern string:<br><br>appears; you type the pattern string and press the Return key. |

Table C-2. Common Functions Defined By a Context File (cont)

| Function | Description |
|---|---|
|  | 6Edit searches for the pattern, moving forward in the file from the current cursor location. If 6Edit finds a match, it places the cursor on the match and fills the screen with data from the area of the file surrounding the match; if 6Edit does not find a match, it does not move the cursor from its location when the Pattern Search key is typed.<br><br>If you want to search for the same pattern string that was used in the last pattern search, just type Return immediately after typing the Pattern Search key. This enters an empty pattern string, which tells 6Edit to use the last pattern string typed. |
| THRU | Type this key when the current cursor location is the beginning of a block of data you want to select. This block becomes the selected block. To complete the selection after typing the THRU key, move the cursor to the end of the block you want to select. At that time, you can either:<br><br>Type a key which performs some operation with the selected block, such as COPY, DELETE, AFTER, or LOCATION, or<br><br>move the cursor to the command window, and enter an editing command. In the command, you can use the predefined name THAT to refer to the selected block, the block of data which you just selected. |
| Substitute | Substitute a string for all pattern matches. When you type this key, the cursor moves to the command window, and one of the following displays occurs.<br><br>o The message<br><br>COPY STRING '' OVER REPEAT SELECT PATTERN '' |

Table C-2. Common Functions Defined By a Context File (cont)

| Function | Description |
|---|---|
| | appears with the cursor on the second apostrophe ('), ready for you to type a replacement string for all pattern matches. You can also type a pattern string between the second pair of apostrophes (''). <br><br> o Or, the message <br><br> New string: <br><br> appears; you then type the new string and press the Return key. Then 6Edit displays the message: <br><br> Old string: <br><br> to which you enter the old string and press the Return key. <br><br> This key can be used to replace all matches of a pattern string with a different string. <br><br> You do not have to type a pattern string if the last pattern you searched for is the one you want to use for the substitution operation. <br><br> The replacement is done over the remainder of the edit block, from the current cursor location to the end of the edit block. <br><br> If you want to limit the replacement to some block of data in the file, insert the specification of that block between the keywords OVER and REPEAT. <br><br> Example: <br><br> `COPY STRING 'xyz' OVER REPEAT SELECT PATTERN 'abc'` <br><br> replaces the remaining occurrences of the string "abc" with the replacement string "xyz". <br><br> `COPY STRING 'xyz' OVER THAT REPEAT SELECT PATTERN 'abc'` |

Table C-2. Common Functions Defined By a Context File (cont)

| Function | Description |
|---|---|
| | limits replacement to the current selected block. You can specify the selected block (prior to typing the Substitute key) by moving the cursor and using the THRU key or the ADJUST key. (See Section 1, Overview of 6Edit.)<br><br>COPY STRING 'xyz' OVER 10 THRU 20 REPEAT SELECT PATTERN 'abc'<br><br>searches between records 10.0 and 20.0; in that range, it replaces all appearances of "abc" with "xyz".<br><br>COPY STRING 'xyz' OVER SECTION_4 REPEAT SELECT PATTERN 'abc'<br><br>assumes that you have created a name "SECTION_4" with the LOCATION command; it limits replacement to the block of data which you located as "SECTION_4". |

# Appendix D

# Customizing the 6Edit User Interface

As you become more familiar with 6Edit, you might want to tailor a context file to better address your particular needs. This appendix outlines some fundamentals to consider before you begin.

## Reasons to Customize

Several reasons warrant designing an alternate context file for 6Edit. You might want to:

o Adapt 6Edit to different keyboards. The key layout would differ.

o Adapt 6Edit to different uses. Different functions would be more important than others, and thus easier to use regularly.

o Adapt 6Edit to different users. Users have different backgrounds, needs, and preferences.

You can design a new context file based on any or all of the above. Design several different context files first for different combinations of needs. For example, design a context file for terminal "x", for function "y", and for user "z". Then design the same file for user "q", addressing various factors until you find the right combination.

You should be familiar with each consideration before designing a new context file. Ask yourself questions about each to determine exactly what to include, and what to omit.

## Customizing: Know the Keyboard

o Does it have function keys?

If so, are they conveniently placed? How far must a hand move to press each one? Are the function keys touch-typed? Are they arranged within a keypad, all typed from a single hand position; or are they arranged in rows, typed by moving the hand position and redirecting the eyes? Must the shift key be engaged to invoke a function key? Must the keys be labeled? On CP-6, many keys which are not normally considered function keys can be treated as such, since their meanings can be redefined. For example, linefeed, home, and delete are used as already-labeled function keys.

o Is the Control key accessible?

  Is it easy to use? With which keys can it be used easily; with which keys must the hand reach?

o Is the Escape key accessible?

  Is it easy to use? Usually a two-key sequence is easier to use when the second key is pressed by the opposite hand.

o Are other special keys accessible?

  Are delete, backspace, return, tab, break, and arrow keys easy to use? Some keyboards place the delete key near letter keys; touch typists can easily reach it there. Other keyboards place the delete key away from the other keys; some easier-to-reach key may perform the "delete character" function instead.

## Customizing: Know the Functions Required

o Are the data stream-oriented, or structured?

  Document files are mostly stream-oriented, represented as a stream of characters to be formatted later. Computer programs are more structured; the data in each record are indented in a significant way. For computer programs, it is more important to be able to manipulate records. For documents, it is more important to manipulate sentences and paragraphs.

o Are the data fixed-form (i.e. tables), or free-form?

o Will users change existing file data, or add data to files? More functions are needed to be able to easily change existing data.

o Will there be much inter-file copying?

  Will users want to view or edit more than one file at a time? If so, inter-file functions become more important.

o How is each function generally viewed?

  Large-scale or small scale? Positioning or editing? What is the unit of data on which each function operates? Character, word, sentence, paragraph, chapter, document; or character, word, record, procedure, module?

## Customizing: Know the Users

o Do the users use only one type of terminal, or do they switch between terminal types regularly? Are most users touch typists? If so, hand position is an important consideration when designing a context file.

o To what system and editor are the users accustomed?

  To CP-6? To another editor? To another operating system? If users are accustomed to another system or editor, you may want to note how they invoke common functions on terminals with which they are familiar. Then, make 6Edit respond similarly when users type old keys and sequences out of habit. This will save learning and re-learning time, and reduce errors.

o How often will the users use 6Edit?

  Frequently? Then performance is important. Casually? Then learning, remembering, and using the processor are important.

o How knowledgeable are the users?

  Do they know computers, jargon, and concepts such as files, records, and record keys? Do they know CP-6 and its capabilities? Do they know editors in general? Are they accustomed to making file transformations in a computer-based, editor-based way, or in a typewriter way? Do they know a specific editor? Is their editing knowledge biased?

These parameters indicate how users feel about editing tasks. If they have never used a computer-based editor before, you might want to emphasize typewriter analogies in the context file. If they know a different system, you might pinpoint similarities between CP-6 and 6Edit concepts, and similar concepts in the old system.

## Invoking Functions

There are three basic ways to invoke functions in 6Edit.

o Using the KEYIN command, you can bind a function to a key or key sequence. (For more information regarding the KEYIN command, see Section 3, 6Edit Commands.)

  You must account not only for the type of key in question, but for the number of keystrokes necessary for each key, the hand position, and whether the key is labeled on the terminal. For example:

| Type of Key/Sequence | number of keystrokes | hand position | labeled? |
|---|---|---|---|
| Function keys | 1 | lost | usually |
| Shifted function keys | 1.5 | lost | usually |
| Alternate-function keys | 1.5 | lost | usually |
| Control-function keys | 1.5 | lost | usually |
| Control key | 1.5 | kept | yes |
| Escape sequence | 2 | may be kept | yes |
| Escape shift sequence | 2.5 | kept | no |

When invoking functions, note the following points:

o Defining a key sequence to be immediate-type in one window, but typeahead-type in another window may give unpredictable results. Either key sequence definition could be used by the FEP, depending on where 6Edit is in its processing of the last terminal input.

o Typing an immediate-type function before 6Edit has IMPed the key sequence to a typeahead-type function, invokes an immediate-type function.

For additional information about these points, see the KEYIN command.

o Using the SYNONYM command, you can define a synonym for a command keyword, part of a command, or several commands. The user remembers the synonym more easily than a command. It is also easier to type. (For more information regarding the SYNONYM command, see Section 3, 6Edit Commands.)

- A simple command is the most flexible means of invoking a function. It is usually more difficult to remember and type, since there are more keystrokes involved.

## Invoking Functions on a Personal Computer

A PC used with the PC Terminal Facility (PCT) and the default PCT key definitions sends two-character sequences for individual function keys (Fn) F1 through F10, shifted function keys (SFn) F1 through F10, alternate function keys (Alt-Fn) F1 through F4, and the cursor control (including Home) keys; and sends three-character sequences for the Ins and Backtab (shifted Tab) keys. These can be defined in 6Edit using the KEYIN command (optionally with the $KEY predefined string function).

The character sequences sent by PCT are interpreted by the FEP based on KEYIN, IMP, and built-in definitions and on the terminal profile, with the results returned to 6Edit. (See Section 3, 6Edit Commands, for more information on KEYIN. See the CP-6 PC Terminal Facility Reference Manual, HA15, for more information on PCT.)

To use the full sets of one-and-a-half-stroke alternate and control function keys (Alt-Fn and Cntl-Fn) with 6Edit KEYIN definitions, the PCT.CMD file must exist on your PC and must have the alternate and control keys defined in the following form:

```
DEF A-Fn "$[[(n-1)$?$ (for alternate function keys)

DEF C-Fn "$[[(n-1)~$ (for control function keys)
```

where n is the number of the function key. For example, Alt-F5 would be defined in PCT.CMD as DEF A-F5 "$[[4$?$. Note that while you can define the keys differently in PCT, the following discussion is based on these definitions and will work only with them, because they correspond to the definitions contained in the CP-6 profiles for PCT (PCTV7800 and PCTX364).

The alternate and control function keys defined in the PCT.CMD file as above cause the PC to send four-character sequences, as follows:

```
ESC [ n DEL (for alternate function keys)

ESC [ n ~ (for control function keys)
```

where n is one less than the number on the function key.

These four-character sequences exceed the three-character limit on the key name (string_expression_1) parameter of the KEYIN command. But the following two variations of the $KEY function permit you to use them:

```
    $KEY(SUn) (for alternate function keys)

    $KEY(UFn) (for control function keys)
```

where n is the number of the function key.

For example:

```
KEYIN $KEY(SU5) IS '%V(27)5R%V(23)'
```

defines the Alt-F5 key sequence to move the cursor left by one word.

```
KEYIN $KEY(UF5) IS '%V(27)5S%V(23)'
```

defines the Cntl-F5 key sequence to move the cursor right by one word.

For variations of the above examples for different key sequences (and additional explanations), see the KEYIN command in Section 3.

Note that introducer characters (set by the $INTRO_1 and $INTRO_2 predefined names) are not required with the SUn and UFn forms of $KEY.

## Associating Functions with Invocations

There are three ways to associate functions with keys:

### Established Association

This method takes advantage of an invocation or function association already familiar to users. For example, if the users have worked on CP-6 before, they know that the Escape N function moves the cursor to the end of the record. If they are familiar with computers, they know that in ASCII, the Control-H function means backspace. If they know a different editor, they may know how to invoke a function similar to a 6Edit function, using another editor's sequence.

## Mnemonic Association

This method employs a mnemonic relationship between the name of a function and the key or sequence used to invoke it. For example, the user types <CNTL-R> for "move right," or <ESC> <B> for "move to the beginning of file."

## Physical Association

This method uses a correlation between the physical relation of two keys or sequences on the keyboard, and the logical relation of two functions. There are three types of physical/logical relations:

- o Spectrum of scale relations:

  You can allocate certain parameters for certain key operations. For example:

  | | |
  |---|---|
  | Control key: | small scope |
  | Esc sequence or function key: | medium scope |
  | shifted Esc sequence key, or shifted function key: | large scope |

  Thus, use Escape 1 for "skip 1 line", and Escape shift 1 for "skip 1 page". Or, use function key 1 for "copy record", and shifted function key 1 for "copy block" or "copy file".

- o Direction on keyboard:

  The direction on the keyboard can imply the direction in the file. For example, type <CNTL-W> to "move left 1 word," and <CNTL-E> to "move right 1 word." Type <ESC> <W> to "move up 8 lines," and <ESC> <A> to "move down 8 lines."

- o Combine the above. For example:

  | | |
  |---|---|
  | <CNTL-W> | move left 1 word |
  | <CNTL-E> | move right 1 word |
  | <ESC> <W> | move up 8 lines |
  | <ESC> <E> | move down 8 lines |

## Hazards

Avoid conflicts with global or system-wide associations. Escape D and Control-X are useful in all CP-6 programs. They should not be redefined unless users will not need the standard CP-6 functions. XON and XOFF (Control-Q and Control-S) are needed for flow control on certain terminals. Do not redefine or use them at all on these terminals.

Avoid erroneous associations:

o Established associations: If users depend on Escape Q to signify status when not using 6Edit, you should not define <ESC> <Q> to mean "delete file."

o Mnemonic associations: Escape D or Control-D could be mnemonically associated with either "down" or "delete."

o Physical associations: When reaching for <CNTL-X> to delete a line, it is easy to accidentally hit <CNTL-Z> or <CNTL-C>. Do not place drastic functions near safe functions.

o Many terminals react to certain control characters in such a way that you cannot redefine the control character, and should not even type the control character on those terminals. Control characters to be aware of include:

  <CNTL-Q> and <CNTL-S>

  XON and XOFF; used for flow control on many terminals.

  <CNTL-P>

  Used by many communications networks; avoid it if your terminal is connected to the CP-6 system via a commercial communications service.

  <CNTL-E>

  ACK; many terminals respond to this key by sending miscellaneous characters to the system.

o When KEYINing a key sequence (either user-defined or built-in) which is defined to be immediate-phase outside of 6Edit, NEVER use an IN-clause ("IN COMMAND" or "IN EDITING"); this would create the dangerous situation where the key sequence is defined to be an immediate-phase function in one of 6Edit's windows, and a read-phase function in the other 6Edit window. Such a situation causes the FEP to act in ways which are unpredictable and confusing to users.

# Appendix E

# Input Editing Functions

This appendix lists all input editing functions available in 6Edit. It assumes that you have not redefined any of the keys on your terminal keyboard. These functions are executed by the Front End Processor (FEP). (For more information on Input Editing Functions, see Section 2, Using 6Edit.)

Table E-1. Input Editing Functions

| Name | Function |
|---|---|
| <ESC> <A> | When the cursor is in the 6Edit editing window: move the cursor up one record. When the cursor is in the 6Edit command window: toggle pagehalt. (Note, however, that 6Edit continuously toggles pagehalt in the command window automatically, overriding your selection.) In the editing window only: <ESC> <n> <A> moves the cursor up "n" records. The n parameter is any 1 or 2 decimal digits. |
| <ESC> <B> | In the 6Edit editing window: move cursor down one record. In the 6Edit command window: simulate Break function. In the editing window only: <ESC> <n> <B> moves the cursor down "n" records. The n parameter is any 1 or 2 decimal digits. |
| <ESC> <CNTL-B> | Simulates the Break function. Does not delete typeahead. |

Table E-1. Input Editing Functions (cont)

| Name | Function |
|---|---|
| <ESC> <C> | In the 6Edit editing window: move cursor right one position (same as <CNTL-R>). In the 6Edit command window: toggles relative tabbing mode. In the editing window only: <ESC> <n> <C> moves the cursor right "n" positions. The n parameter is any 1, 2, 3, or 4 decimal digits. |
| <ESC> <D> | In the 6Edit editing window: move cursor left one position (same as <BS>). In the 6Edit command window: recalls the last saved input record. In the editing window only: <ESC> <n> <D> moves the cursor left "n" positions. The n parameter is any 1, 2, 3, or 4 decimal digits. |
| <ESC> <CNTL-D> | Recall last input saved in command or IBEX window. |
| <ESC> <n> <E> | Position to the top, middle, or bottom of the window for n less than, equal to, or greater than one, respectively. |
| <ESC> <G> | Display FEP information. |
| <ESC> <H> | Halt output immediately. Type <CR> to resume output. |

Table E-1. Input Editing Functions (cont)

| Name | Function |
|---|---|
| <ESC> <n> <H> | Set HSSHIFT (the number of columns that an input image is to be shifted) to the value n. For this sequence to work, the $HSCROLL predefined name must be set non-zero. |
| <ESC> <I> | Simulate Tab.<br><br><ESC> <n> <I> moves the cursor to position "n" of the current record. <ESC> <-> <I> moves the cursor to the previous tab stop. The n parameter is any 1, 2, 3, or 4 decimal digits. |
| <ESC> <J> | Define or reset the insertion window. |
| <ESC> <K> | Delete record (or insertion window) to right of cursor. |
| <ESC> <CNTL-K> | Delete record (or insertion window) to left of cursor. |
| <ESC> <CNTL-L> | Delete record, leaving the cursor at the next record. |
| <ESC> <M> | Set replacement mode. Resets overstrike and insertion mode. |

Table E-1. Input Editing Functions (cont)

| Name | Function |
|---|---|
| <ESC> <n> <M> | Set HSMARGIN (the closest the cursor is to be allowed to approach the end of an input area) to the value n. For this sequence to work, the $HSCROLL predefined name must be set non-zero. |
| <ESC> <N> | Move cursor to the end of record (or insertion window). |
| <ESC> <O> | Set overstrike mode. Resets replacement and insertion modes. |
| <ESC> <P> | Recall record as it was when the cursor was last moved onto record. This function is only available in the 6Edit editing window. <ESC> <n> <P> copies the nth record below the current record over the current record.<br><br><ESC> <n> <-> <P> copies the nth record above the current record over the current record. The n parameter is any 1 or 2 decimal digits. |
| <ESC> <Q> | Display system status. |
| <ESC> <R> | Retype record; reset insertion window.<br><br><ESC> <n> <R> resets the parameter identified by "n" (turns off the parameter). The n parameter is any 1 or 2 decimal digits. The table below lists the values for n: |

Table E-1. Input Editing Functions (cont)

| Name | Function |
|---|---|
| | |

| n | Meaning |
|---|---|
| 1 | Autotab mode (see $AUTOTAB in Appendix A). Can only be used to disable autotabing. Setting this parameter (via <ESC> <1> <S>) or toggling the parameter on (via <ESC> <1> <T>) has the same effect as <ESC> <1> <R> -- the parameter is reset. |
| 2 | Insert mode (see $INSERT in Appendix A). |
| 3 | Message line for operator messages. |
| 4 | Wordwrap mode (see $WORDWRAP in Appendix A). Can only be used to disable wordwrap mode. Setting this parameter (via <ESC> <4> <S>) or toggling the parameter on (via <ESC> <4> <T>) has the same effect as <ESC> <4> <R> -- the parameter is reset. |
| 5 | Search to the right. Establishes the direction for word and character searches. |
| 6 | Truncate mode (see the IBEX TERMINAL command). |
| 7 | Actontrn (see the IBEX TERMINAL command). |
| 8 | Wordwrap toggle (see $WORDWRAP in Appendix A). Resetting this parameter disables wordwrap mode but remembers the column. Setting this parameter enables wordwrap mode at the remembered column. |
| 9 | Recordwrap mode (see $RECORDWRAP in Appendix A). |
| 10 | Textedit mode. Attempts to preserve spaces between words when splitting and joining records. |
| 11 | Easyappend. Allows records to be appended to the end of the file simply by moving the cursor down past the end-of-file position. |
| 12 | Changecase. Makes forward cursor movement change the case of the characters under it. |
| 13 | Parkcursor. Moves the cursor to the window's home position when activation occurs in screen-editing mode. |

The following values of "n" are only available in the command window:

| Name | Function |
|---|---|
| | |

| n | Meaning |
|---|---|
| 20 | Overstrike mode |
| 21 | Echo |
| 22 | Paritycheck |
| 23 | Outputdiscard |
| 24 | Uppercase |
| 25 | Apllcnrm |
| 26 | Retypovr |
| 27 | Editovr |
| 29 | Pagehalt |
| 30 | Printhalt |
| 31 | Relpage |
| 32 | Simper |
| 33 | Nooptimiz |
| 34 | Wrappage |
| 35 | Localech |
| 36 | Messagehalt |
| 37 | Autocursor |
| 38 | Truovrprt |
| 39 | Cursorread |
| 40 | Keeptypahd |

See the IBEX TERMINAL command for descriptions of these parameters.

The following values of "n" are available only in the command window, and when FCNTBL=CP5:

| n | Meaning |
|---|---|
| 50 | Tabsim |
| 51 | Tabrelative |
| 52 | Spaceinsert |
| 53 | Lowercase |
| 54 | Fullduxpapertape |
| 55 | Halfduxpapertap |

See the IBEX TERMINAL command for descriptions of these parameters.

Table E-1. Input Editing Functions (cont)

| Name | Function |
|---|---|
| <ESC> <CNTL-R> | Reset insertion window. |
| <ESC> <n> <S> | Sets the parameter identified by "n" (turns on the parameter). The n parameter is any 1 or 2 decimal digits. See <ESC> <R> for the table of values for n. |
| <ESC> <n> <T> | Toggles the parameter identified by "n". If the parameter was set, it will be reset; if it was reset, it will be set. The n parameter is any 1 or 2 decimal digits. See <ESC> <R> for the table of values for n. |
| <ESC> <U> | Toggle uppercase restriction. |
| <ESC> <V> | Search record (or insertion window) for character. |
| <ESC> <CNTL-W> | Delete word, beginning with the character under the cursor, and proceeding to the right, up to the beginning of the next word. |
| <ESC> <X> | Erase contents of record. |

Table E-1. Input Editing Functions (cont)

| Name | Function |
|---|---|
| <ESC> <Y> | Monitor attention. |
| <ESC> <'> | Set insertion mode. Resets overstrike and replacement modes. |
| <ESC> <*> | Save contents of the insertion window. |
| <ESC> <:> | Recall the data saved by the last <ESC> <*> function. |
| <ESC> <.> | When more than one session has been started on this terminal, <ESC> <.> moves the cursor to the next session reading from the terminal. |
| <ESC> <(> | Enter lowercase shift. |
| <ESC> <)> | Exit lowercase shift. |

| Table E-1. Input Editing Functions (cont) | |
|---|---|
| Name | Function |
| <ESC> <-> | Redisplay entire editing window. This function is only available in the 6Edit editing window. |
| <ESC> <>> and <<> | Set limits of insertion window. |
| <ESC> <^> | Set wordwrap column.<br><br><ESC> <n> <^> sets the wordwrap mode column to position "n". The n parameter is any 1, 2, 3, or 4 decimal digits. For example, <ESC> <8> <^> sets the wordwrap column to position 8. (See Section 2.) |
| <ESC> <BS> | If cursor is at beginning of record, join record to previous record. If cursor is at end of record, join next record to current record. This function is only available in the 6Edit editing window. |
| <ESC> <HT> | Set autotab column.<br><br><ESC> <n> <HT> sets the autotab column to position "n". The n parameter is any 1, 2, 3, or 4 decimal digits. (See predefined name $AUTOTAB in Appendix A.) |

Table E-1. Input Editing Functions (cont)

| Name | Function |
|---|---|
| <ESC> <CR> | Move cursor to beginning of record (or insertion window). |
| <ESC> <LF> | In the 6Edit editing window: split record, inserting the portion of the record to the right of the cursor as a new record to follow the current record. In the 6Edit command window: local linefeed.<br><br>In the editing window only: <ESC> <2> <LF> will fill or shorten the current record to the wordwrap column. If the record is shorter than the wordwrap column, words from the next record will be moved onto the current record. If the record is longer than the wordwrap column, it will be split near the wordwrap column. (See predefined name $WORDWRAP in Appendix A.)<br><br>In the editing window only: <ESC> <1> <LF> will perform the <ESC> <2> <LF> actions repeatedly until the cursor reaches the next paragraph, or the bottom of the editing window. |
| <ESC> <ESC> | Monitor attention (same as <ESC> <Y>). |
| <ESC> <DEL> | Delete the character to the left of the cursor; if at the beginning of record, do nothing.<br><br>If the recordwrap mode is set and the cursor is at the beginning of the record, acts like <ESC> <BS>. It joins the current record to the previous record. (See $RECORDWRAP in Appendix A). |

Table E-1. Input Editing Functions (cont)

| Name | Function |
|---|---|
| <CNTL-H> | (BS) Move cursor left one position; if at the beginning of record, action depends on the record wrap mode (see $RECORDWRAP in Appendix A). If the record wrap mode is disabled (the initial setting) or the cursor is in the command window, <CNTL-H> when at the beginning of a record does nothing. If the record wrap mode is enabled and the cursor is in the editing window, <CNTL-H> when at the beginning of a record moves the cursor to just beyond the end of the previous record. |
| <CNTL-I> | (TAB) Move cursor to the next tab stop. Extend record if necessary. |
| <CNTL-J> | (LF) In the 6Edit editing window: move cursor down one record (same as <ESC> <B>). In the 6Edit command window: activates a command entry. |
| <CNTL-M> | (CR) In the 6Edit editing window: move cursor to the beginning of the next record. If at the end of the file, append a zero-length record to the file. In the 6Edit command window: activates a command entry. |
| <CNTL-R> | (DC2) Move cursor right one position; if at the end of record, action depends on the record wrap mode (see $RECORDWRAP in Appendix A). If the record wrap mode is disabled (the initial setting) |

Table E-1. Input Editing Functions (cont)

| Name | Function |
|---|---|
| | or the cursor is in the command window, <CNTL-R> when at the end of a record appends a space character to the record. If the record wrap mode is enabled and the cursor is in the editing window, <CNTL-R> when at the end of a record moves the cursor to the beginning of the next record. |
| <CNTL-U> | (NAK) Move cursor right one character, changing the case of the character under the cursor. |
| <CNTL-X> | (CAN) Same as <ESC> <X>, but acts immediately. (Also deletes typeahead and queued output.) |
| <CNTL-W> | (ETB) Move cursor to the next or previous word, depending on current direction, set by the last <BS> or <CNTL-R>. |
| <CNTL-Y> | (EM) Monitor attention. (Also deletes typeahead and queued output.) |
| <DEL> | Delete character under cursor. If in the replacement mode and at end of record (or insertion window), delete character to left of cursor; if in the insertion mode and at end of record (or insertion window), <DEL> is ignored. |

Table E-1.  Input Editing Functions (cont)

| Name | Function |
|---|---|
| <BREAK> | Interrupt program. |

# Appendix F

# The ASCII Character Set

This appendix presents the ASCII character set.

Table F-1. The ASCII Character Set

| ASCII NAME | CHARACTER ENTRY | DECIMAL CODE |
|---|---|---|
| NULL | <CNTL-@> | 0 |
| SOH | <CNTL-A> | 1 |
| STX | <CNTL-B> | 2 |
| ETX | <CNTL-C> | 3 |
| ECT | <CNTL-D> | 4 |
| ENQ | <CNTL-E> | 5 |
| ACK | <CNTL-F> | 6 |
| BEL | <CNTL-G> | 7 |
| BS | <CNTL-H> | 8 |
| HT | <CNTL-I> | 9 |
| LF | <CNTL-J> | 10 |
| VT | <CNTL-K> | 11 |
| FF | <CNTL-L> | 12 |
| CR | <CNTL-M> | 13 |
| SO | <CNTL-N> | 14 |
| SI | <CNTL-O> | 15 |
| DLE | <CNTL-P> | 16 |
| DC1 | <CNTL-Q> | 17 |
| DC2 | <CNTL-R> | 18 |
| DC3 | <CNTL-S> | 19 |
| DC4 | <CNTL-T> | 20 |
| NAK | <CNTL-U> | 21 |
| SYN | <CNTL-V> | 22 |
| ETB | <CNTL-W> | 23 |

Table F-1. The ASCII Character Set (cont)

| ASCII NAME | CHARACTER ENTRY | DECIMAL CODE |
|---|---|---|
| CAN | <CNTL-X> | 24 |
| EM | <CNTL-Y> | 25 |
| SUB | <CNTL-Z> | 26 |
| ESC | <CNTL-[> | 27 |
| FS | <CNTL-\> | 28 |
| GS | <CNTL-]> | 29 |
| RS | <CNTL-^> | 30 |
| US | <CNTL-_> | 31 |
| SP | < > | 32 |
| ! | <!> | 33 |
| " | <"> | 34 |
| # | <#> | 35 |
| $ | <$> | 36 |
| % | <%> | 37 |
| & | <&> | 38 |
| ' | <'> | 39 |
| ( | <(> | 40 |
| ) | <)> | 41 |
| * | <*> | 42 |
| + | <+> | 43 |
| ' | <'> | 44 |
| - | <-> | 45 |
| . | <.> | 46 |
| / | </> | 47 |
| 0 | <0> | 48 |
| 1 | <1> | 49 |
| 2 | <2> | 50 |
| 3 | <3> | 51 |
| 4 | <4> | 52 |
| 5 | <5> | 53 |
| 6 | <6> | 54 |
| 7 | <7> | 55 |

Table F-1. The ASCII Character Set (cont)

| ASCII NAME | CHARACTER ENTRY | DECIMAL CODE |
|---|---|---|
| 8 | <8> | 56 |
| 9 | <9> | 57 |
| : | <:> | 58 |
| ; | <;> | 59 |
| < | <<> | 60 |
| = | <=> | 61 |
| > | <>> | 62 |
| ? | <?> | 63 |
| @ (at sign) | <@> | 64 |
| A | <A> | 65 |
| B | <B> | 66 |
| C | <C> | 67 |
| D | <D> | 68 |
| E | <E> | 69 |
| F | <F> | 70 |
| G | <G> | 71 |
| H | <H> | 72 |
| I | <I> | 73 |
| J | <J> | 74 |
| K | <K> | 75 |
| L | <L> | 76 |
| M | <M> | 77 |
| N | <N> | 78 |
| O | <O> | 79 |
| P | <P> | 80 |
| Q | <Q> | 81 |
| R | <R> | 82 |
| S | <S> | 83 |
| T | <T> | 84 |
| U | <U> | 85 |
| V | <V> | 86 |
| W | <W> | 87 |

Table F-1. The ASCII Character Set (cont)

| ASCII NAME | CHARACTER ENTRY | DECIMAL CODE |
|---|---|---|
| X | <X> | 88 |
| Y | <Y> | 89 |
| Z | <Z> | 90 |
| [ (left bracket) | <[> | 91 |
| \ (backslash) | <\> | 92 |
| ] (right bracket) | <]> | 93 |
| ^ (circumflex) | <^> | 94 |
| _ | <_> | 95 |
| ' | <'> | 96 |
| a | <a> | 97 |
| b | <b> | 98 |
| c | <c> | 99 |
| d | <d> | 100 |
| e | <e> | 101 |
| f | <f> | 102 |
| g | <g> | 103 |
| h | <h> | 104 |
| i | <i> | 105 |
| j | <j> | 106 |
| k | <k> | 107 |
| l | <l> | 108 |
| m | <m> | 109 |
| n | <n> | 110 |
| o | <o> | 111 |
| p | <p> | 112 |
| q | <q> | 113 |
| r | <r> | 114 |
| s | <s> | 115 |
| t | <t> | 116 |
| u | <u> | 117 |
| v | <v> | 118 |
| w | <w> | 119 |

Table F-1. The ASCII Character Set (cont)

| ASCII NAME | CHARACTER ENTRY | DECIMAL CODE |
|---|---|---|
| x | <x> | 120 |
| y | <y> | 121 |
| z | <z> | 122 |
| { (left brace) | <{> | 123 |
| \| (or bar) | <\|> | 124 |
| } (right brace) | <}> | 125 |
| ~ (tilde) | <~> | 126 |
| DEL | <DEL> | 127 |

# Index

Technical Publications Remarks Form

| | | | |
|---|---|---|---|
| TITLE | CP-6 6Edit Screen Editor Reference | ORDER NO. | CE70-02 |
| | | DATED | June 1990 |

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here. ☐

**PLEASE FILL IN COMPLETE ADDRESS BELOW.**

FROM: NAME ______________________ DATE ________

TITLE ______________________

COMPANY ______________________

ADDRESS ______________________

______________________

CUT ALONG LINE

.ASE FOLD AND TAPE–
ΓE: U.S. Postal Service will not deliver stapled forms

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

**BUSINESS REPLY MAIL**
FIRST CLASS PERMIT NO. 39531 WALTHAM, MA

POSTAGE WILL BE PAID BY ADDRESSEE

**Bull HN Information Systems Inc.**
**Attn: Publications MA02/305C**
**Technology Park**
**Billerica, MA 01821-9904**

Bull

**Bull HN Information Systems Inc.**

**Corporate Headquarters:** Technology Park, Billerica, MA 01821-4199
**Mexico:** Hamburgo No. 64, Col. Juarez Delegacion Cuauhtemoc, 06600 Mexico, D.F.
**U.K.:** Great West Rd., Brentford, Middlesex TW8 9DH, England **Italy:** 32 Via Pirelli, 20124 Milan
**Canada:** 155 Gordon Baker Rd., North York, Ontario, M2H 3P9 **New Zealand:** 14/16 Liverpool Street, Auckland 1
**Asia:** 4/F, Shui on Centre, 6-8 Harbour Rd., Wanchai, Hong Kong **Australia:** 124 Walker Street, North Sydney, N.S.W. 2060

Printed in U.S.A.