Utility Commands

P-STAT: Utility Commands

Second edition. January 2013

This publication corresponds to **P-STAT Version 3.01, January 2013**. This publication is designed to provide a comprehensive description of P-STAT's utility commands for controlling the environment and for the import and export of files in various formats.

Please direct any questions to:

P-STAT, Inc.
230 Lambertville-Hopewell Rd.
Hopewell, New Jersey 08525-2809
U.S.A.

Telephone: 609-466-9200

Fax: 609-466-1688

Internet: support@pstat.com

Web Page URL: http://www.pstat.com

All rights reserved. Except as permitted under the United States Copyright Act of 1976, no part of this publication may be reproduced or distributed in any form or by any means, electronic or mechanical, including photocopying, recording, or any information storage and retrieval system without the prior written permission of P-STAT, Inc.

P-STAT is a registered trademark of P-STAT, Inc. Windows is a registered trademark of MicroSoft Corp.
Copyright © 1972-2013 P-STAT, Inc. Printed in the US. Published by P-STAT, Inc.

CONTENTS

Utility Commands for File Management

THE SHOW COMMAND 1.1

THE SHOWBYTES COMMAND 1.4

THE REFORMAT COMMAND 1.6

THE PATCH COMMAND 1.7

THE EXAMINE COMMAND 1.8

Utility Commands For General Usage

BATCH AND INTERACTIVE RUNS 2.1

CONTROLLING THE PRINTOUT 2.2

- PRINTER.SETTINGS and DEFAULT.SETTINGS 2.2
- The Vertical Bar Character 2.3
- The Page Control Character 2.3
- Changing the Output Destination 2.4
- Setting the Number of Lines of Output 2.6
- Using General Identifiers 2.6

BUILDING A DICTIONARY 2.7

THE TRANSFER COMMAND 2.8

- The TRANSFER Command 2.8
- Errors in Transfer Files 2.9

PSTART 2.10

Utility Commands For Interactive Use

CONTROLLING THE INTERACTIVE ENVIRONMENT 3.1

- Starting an Interactive Run 3.1
- Controlling Output on the Terminal 3.2
- Using LINES to Control the Output 3.3
- Re-executing Commands 3.3

CONTROLLING THE EDIT FILE AND ITS CONTENTS 3.4

- System and External Edit Files 3.4
- Adding a Command to an Edit File 3.5

Batch Usage For Continuous Execution

Starting a Batch Run 4.1
The BATCH Command 4.2
Differences Between Batch and Interactive Computing 4.2
Handling Errors in Batch Mode 4.4
Printing the Input Stream 4.5

HELP IN BATCH RUNS 4.5

Interactive Usage and the Editor

INTERACTIVE MODE 5.1
Listing Identifiers for a Command 5.1
Prompt Messages 5.2

THE P-STAT EDITOR 5.3
The Edit File 5.3
Execution from the Edit File 5.4

BASIC STRATEGIES FOR EDITOR USE 5.4
Correcting a Command Error Using QUIT 5.4
Deleting a Command Error 5.5
Correcting an Error in the Command Text 5.5
Using Mixed Case Within the Editor 5.7

The Editor in Detail

EDIT MODE FOR CORRECTION 6.1
Structure of the Edit File 6.1
Entering the Editor 6.2
Moving Between the Command and Data Editor 6.3
Fixing an Error in a Command 6.5
Executing a Corrected Command 6.6
Single-Letter Instructions 6.7

MOVING ABOUT WITHIN THE EDIT FILE 6.7
Instructions that Move the Pointer 6.7
Other Instructions that May Move the Pointer 6.8
Locating a Specific Record 6.8
Checking for Correct Syntax 6.13
Inserting a Single Record in the Edit File 6.14

MORE ADVANCED EDITOR INSTRUCTIONS 6.15
Repeating Editor Instructions for Global Corrections 6.15
Saving and Restoring the Edit File 6.15

Saving and Restoring Subcommand and Data Records. 6.16
Moving and Copying Commands 6.18
Using the SQUEEZE Operator 6.18
Fixing a Transmission Error 6.19

The Editor For Input

INPUT MODE 7.1
The INPUT Instruction 7.1
Location of the Pointer 7.2
Editor Prompting 7.2
BUILDING A RUN 7.2
Checking For Syntax 7.4
Correcting the Edit File 7.4
ENTERING COMMAND AND DATA RECORDS 7.5
Moving to Different Levels of the Editor 7.6
Entering Data Input From the Data Editor. 7.7

FIGURES

The SHOW Command 1.1
Using SHOW to Display the Variables in a File 1.2
SHOW Command Identifiers. 1.2
SHOW with TAGS.ONLY and TEXT.ONLY 1.4
Partial Output From SHOWBYTES Command 1.5
REFORMAT Report 1.7
EXAMINE Report 1.9
EXAMINE Report: End of Case is Used. 1.10

Directing Output to Disk File “PrtFile” and then Back to the Terminal 2.5
Creating a Dictionary of Variables 2.7
PSTART File: Prompt for the Autosave Directory 2.10

Commands to Begin an Interactive Run in the PSTART File. 3.2
Adding a Revised Command to an Edit File 3.6

Fatal Errors in a Batch Run 4.3

Determining the Identifiers for a Command 5.2
Long and Short Prompt Messages 5.2
Full Screen Editor 5.6

Using the Editor To Correct a Command Error. 6.3
Using the Editor To Correct an Error in Data 6.10
Moving the Pointer 6.12
Syntax Check in the Editor 6.14
GET and PUT with Data Records 6.17
Fixing a Transmission Error 6.19

Building a Run Using the Editor 7.3
Editor input: Commands and Data 7.5
Help with Data Input in the Editor. 7.7
Correcting a Data Record 7.9

1
Utility Commands for File Management

There are four chapters containing utility commands. This chapter covers those commands that provide information about files as distinct from the commands which create or modify P-STAT system files. Those commands which include MODIFY, SORT, CONCAT and LOOKUP are documented in the manual "P-STAT: File Management"

The other chapters detailing the various utility commands are;

1. "Utility Commands For General Use" includes printer controls such as OUTPUT.WIDTH and PAGE.CHARACTER and commands such as TRANSFER, SLEEP and BYPASS which control the flow of the run
2. "Utility Commands For Interactive Use including an introduction to the P-STAT editor.
3. "Batch Usage for Continuous Execution" which describes commands needed to run large jobs in the background.

Figure 1.1 The SHOW Command

```
SHOW, IDEN $

The SHOW command has these identifiers.....

across                    head                      sort
break                     show
columns                   single
```

1.1 THE SHOW COMMAND

SHOW lists the variables in the file with their position in the file and their data type. If the file name is not provided the most recently referenced file is used. SHOW can be used as a dictionary of the variables in a file and has a number of identifiers to control the appearance of the display.

Figure 1.1 uses the general identifier "IDEN" to list all the identifiers that are specific to the SHOW command. IDEN, a general identifier that can be used with any command, is not included in the identifier list.

Figure 1.2 illustrates the use of SHOW with a file named Paceset. This file has both character and numeric data. It also contains some variables with simple variable names and others which contain tags and text. Character data is indicated with the letter C followed by the length assigned to the variable. Numeric data is indicated by an N. The N indicates numbers that are stored as double precision values assuring the accuracy of any computations that may be done. Because the command has no instructions about the form of the variable names full names are used.

Figure 1.2 Using SHOW to Display the Variables in a File

```
SHOW  Paceset $

Jan 24, 2012: positions, types and full names in file Paceset

   1 N    id
   2 N    Age::Age_of_respondent
   3 N    Sex::Gender_of_respondent
   4 N    Income
   5 N    Num.TV::Number_of_television_sets_in_your_home

   6 N    VCR::Owns_a_VCR_or_a_DVD_player
   7 N    CD::Owns_one_or_more_VCRs
   8 N    Port.Phone::Primary_telephone_is_portable
   9 N    Ans.Mach::Telephone_has_an_answering_machine
  10 N    Store.1::Where_was_the_purchase_made

  11 N    Store.2
  12 N    Store.3
  13 N    Store.4
  14 N    Income.Groups::Income_of_the_Respondent
  15 C6   Own::Home_Ownership
```

:

Figure 1.3 has 3 SHOW commands all using just the TAG format variable names. The first illustrates the SORT identifier which arranges the variables in alphabetic order. The second illustrates the ACROSS identifier which has the variables arranged across the rows rather than down the columns . The third illustrates the COLUMNS identifier. The fourth example in Figure 1.3 uses a combination of SORT, ACROSS, and COLUMNS.

When you have long variable names, the SHOW command cannot fit all of these options across the page. The only identifiers that are honored when you have variable names longer than 16 characters are SORT and NO BREAK .

Figure 1.3 SHOW Command Identifiers

```
SHOW paceset, TAGS,  SORT $

...............

Tue Mar 31  1998: positions, types and names in file paceset

  2 D   Age               14 D   Income.Groups      10 D   Store.1
  9 D   Ans.Mach           5 D   Num.TV             11 D   Store.2
  7 D   CD                15 C16 Own                12 D   Store.3
  1 D   ID                 8 D   Port.Phone         13 D   Store.4
  4 D   Income             3 D   Sex                 6 D   VCR
```

```
SHOW paceset, ACROSS, TAGS $

...............

Tue Mar 31  1998: positions, types and names in file paceset

  1 D   ID                  2 D   Age                 3 D   Sex
  4 D   Income              5 D   Num.TV              6 D   VCR
  7 D   CD                  8 D   Port.Phone          9 D   Ans.Mach
 10 D   Store.1            11 D   Store.2            12 D   Store.3
 13 D   Store.4            14 D   Income.Groups      15 C16 own


SHOW paceset, TAGS, COLUMNS 2 $

...............

Tue Mar 31  1998: positions, types and names in file paceset

  1 D   ID                 11 D   Store.2
  2 D   Age                12 D   Store.3
  3 D   Sex                13 D   Store.4
  4 D   Income             14 D   Income.Groups
  5 D   Num.TV             15 C16 own

  6 D   VCR
  7 D   CD
  8 D   Port.Phone
  9 D   Ans.Mach
 10 D   Store.1


SHOW paceset, ACROSS, SORT, COLUMNS 2, TAGS $

...............

Jan 24, 2012: positions, types and short: tags in file test

  2 N  Age                  9 N  Ans.Mach
  7 N  CD                   1 N  id
  4 N  Income              14 N  Income.Groups
  5 N  Num.TV              15 C6 Own
  8 N  Port.Phone           3 N  Sex

 10 N  Store.1             11 N  Store.2
 12 N  Store.3             13 N  Store.4
  6 N  VCR
```

:

Figure 1.4 SHOW with TAGS.ONLY and TEXT.ONLY

```
SHOW paceset, TAGS.ONLY $

Jan 27, 2012: positions, types and tags only in file test

   1 N  ---                   6 N  VCR               11 N  ---
   2 N  Age                   7 N  CD                12 N  ---
   3 N  Sex                   8 N  Port.Phone        13 N  ---
   4 N  ---                   9 N  Ans.Mach          14 N  Income.Groups
   5 N  Num.TV               10 N  Store.1           15 C6 Own



SHOW paceset, TEXT.ONLY

 Jan 27, 2012: positions, types and text only in file test

    1 N    ---
    2 N    Age_of_respondent
    3 N    Gender_of_respondent
    4 N    ---
    5 N    Number_of_television_sets_in_your_home

    6 N    Owns_a_VCR_or_a_DVD_player
    7 N    Owns_one_or_more_VCRs
    8 N    Primary_telephone_is_portable
    9 N    Telephone_has_an_answering_machine
   10 N    Where_was_the_purchase_made

   11 N    ---
   12 N    ---
   13 N    ---
   14 N    Income_of_the_Respondent
   15 C6   Home_Ownership
```

The next chapter contains an illustration which uses the SHOW command combined with several other utility commands to build a dictionary for a P-STAT system file.

1.2 THE SHOWBYTES COMMAND

SHOWBYTES displays the bytes of a file. Three values are shown for each byte: its position in the file, its ascii value (0 to 255), and its ascii representation (if it is a printable character). Non-printable characters like carriage return and line-feed are represented by CR, LF, and so forth.

A 24 by 80 screen can display up to 120 bytes at a time, and a 59 by 132 page can display up to 500 bytes. The default is to display all of the bytes in the file, which is fine for files of a few thousand bytes, but impractical for megabyte files. Therefore, the following identifiers can be used to select which part of the file should be displayed:

1. FIRST 22, start with byte 22.
2. FIRST -50, start with the 50th byte from the end. Thus, if the file has 1,000,000,050 bytes, the display would start with byte 1,000,000,001.
3. LAST 90, end with byte 90.
4. MAX 100, show no more than 100 bytes.

The FIND identifier can be used to find specific bytes in the file and display the surrounding area. For example:

```
SHOWBYTES 'somefile', FIND LF, FIRST 25001 $
```

causes the command to search for a linefeed at or beyond byte 25,001 of the file. If one is found, a display occurs showing the context of that linefeed. Each subsequent linefeed causes another display.

Figure 1.5 Partial Output From SHOWBYTES Command

```
SHOWBYTES  'Test.lab',  COLUMNS 3,  FIND LF,  PREVIOUS 0 $


 81  10 lf   101  32      121  32
 82  32      102  50 2    122  32
 83  83 S    103  32      123  32
 84 101 e    104  41 )    124  32
 85 120 x    105  32      125  32
 86  32      106  39 ’    126  32
 87  40 (    107  70 F    127  32
 88  32      108 101 e    128  32
 89  49 1    109 109 m    129  32
 90  32      110  97 a    130  32

 91  41 )    111 108 l    131  32
 92  32      112 101 e    132  32
 93  39 ’    113  39 ’    133  32
 94  77 M    114  32      134  32
```

Find can define a search string of up to 80 bytes. Usually, however, one or two bytes will be sought. The bytes after FIND can be specified by ascii value, by name, and by quotes. For example:

```
FIND cr lf ‘ab’ 99,
```

looks for any 5-byte sequence of carriage return, linefeed, lowercase ’a’, lowercase ’b’ and lowercase ’c’.

```
FIND 13 10 97 98 99,
```

would do the same thing.

The default is to start each display 10 bytes before the matched byte or bytes. Using PREVIOUS 0 in the command causes the display to start with the matched area. PREVIOUS 50 shows more of the preceding bytes, and so on. Figure 1.5 illustrates SHOWBYTES output. The first linefeed is found at byte 81. This is printed at the top of the printout because PREVIOUS 0 is specified.

There are a number of identifiers which determine how the bytes will be displayed. A screen will usually have 6 columns of byte descriptions, separated by a one-position gap. Each column contains the three values described above. The screen will have 10 lines of byte information, a blank line, and 10 more lines.

A page will usually have 5 chunks of 10 lines by 10 columns. The identifiers can be used to modify the default layouts are:

1. CHUNK 5, sets the number of lines before a break. The default is 10, it can be 1 to 20. The number of lines (not counting breaks) on a screen or page will be an integer multiple of the CHUNK size.
2. GAP 3, sets the number of spaces between columns of printout (where each column has 3 values). The default is 1, it can be 1 to 10.
3. NO BREAK, the default is to print a blank line after every CHUNK of lines. Using NO BREAK causes the blank line to be omitted
4. NO HEAD, should each page or screen begin with a heading, which consists of one line with the filename and the file length, followed by a blank line.

   The default is yes for pages, and no for screens. Using NO HEAD causes the heading to be omitted, which provides 2 additional lines for printout. Using HEAD causes it to be included.
5. COLUMNS 4,requests that many columns of byte information on the screen or page. The default is as many as can fit. COLUMNS can only request a smaller amount.
6. ZERO, should the initial byte in the file be labelled byte 0 or byte 1, and so forth? The default is 1. If ZERO is used, the byte locations are zero-based.

1.3 THE REFORMAT COMMAND

REFORMAT is a utility command that reads an external file, removes, changes, and/or inserts characters as indicated, and writes the resulting external file.

```
REFORMAT 'xxx1',  OUT 'xxx2',  REMOVE NL,  INSERT CRLF $
```

When two characters are used, they should be separated by a space except for CRLF and “cc”. The characters can be represented as follows:

1. CRLF this is the same as using CR and LF
2. CR carriage return (13)
3. LF line feed (10)
4. NL new line (10)
5. EOF end-of-file (26)
6. 10 any integer from 0 to 255.
7. 'c' any single character
8. 'cc' any two characters

There are 7 forms of the command, as shown below. In these examples, KK (kk) means one or two characters can be given, and K (k) means only one character can be given. Integers must be used where integer constants appear.

1. Remove the one or two characters wherever they occur. For example, REMOVE EOF to strip the end-of-file character from (hopefully) the end of the file.
   ```
   REFORMAT ’x’, OUT ’y’, REMOVE kk $
   ```
2. Replace the one or two REMOVE characters wherever they occur with the one or two INSERT characters. For example, REMOVE NL, INSERT CRLF. Or, REMOVE NL, INSERT ’ ’ could be used to change newline characters into blanks in Unix TEXTWRITER output.
   ```
   REFORMAT ’x’, OUT ’y’, REMOVE kk, INSERT kk $
   ```
3. Insert the one or two characters after every 12 input characters. For example, suppose a file has 32000 bytes, 1000 cases with 32 characters each, but no NL or CRLF. Using AFTER 32, INSERT CRLF would make the file usable in line-oriented commands like BUILD.

```
REFORMAT 'x', OUT 'y', REMOVE kk, INSERT kk $
```

4. Insert the one or two characters after every 12th occurrence of the AFTER character. Occurrences within quoted strings, however, are ignored. For example, each case has 12 vertical-bar delimited values but no newline: use AFTER 12 '|', INSERT NL.

```
REFORMAT 'x', OUT 'y', AFTER  12  k, INSERT kk $
```

5. Like the previous example, except that occurrences within quoted strings are included in the count.

```
REFORMAT 'x', OUT 'y', AFTER  12  k, INSERT kk,  ANYWHERE $
```

6. Copy the file from the first indicated byte through the last. If FIRST is omitted, 1 is assumed. If LAST is omitted, the final byte of the file is assumed. FIRST and/or LAST cannot be used with the other forms.

```
REFORMAT 'x', OUT 'y', FIRST  4,  LAST 12980 $
```

7. Copy the file, changing the specified bytes to the given values. Here, byte 1 in the output file becomes a CR (13), and byte 733 an X (88). The other bytes of the input file are copied to the output file without change.

```
REFORMAT 'x', OUT 'y', BYTES 1 733, VALUES cr 'X' $
```

Up to 20 bytes locations and values can be given. If only one value is given, all the specified byte locations are changed to that value; otherwise there should be a supplied value for each designated byte.

Figure 1.6 REFORMAT Report

```
REFORMAT 'Unix.text',  OUT 'PC.txt', REMOVE NL,  INSERT CRLF $


--------REFORMAT command completed--------
Input  file Unix.text has 37,586 bytes.
Output file PC.txt has 38,913 bytes.
An ascii 10 was to be replaced by ascii 13 and 10
wherever it occurred.
1,327 such replacements were done.
```

1.4 THE PATCH COMMAND

The PATCH command, which is similar in spirit to the REFORMAT command, reads an external ASCII file and writes a new external ASCII file. In the process, bytes can be deleted and/or added at a location specified in the command.

```
PATCH file1,  OUT file2,  AFTER 1234,             ADD 'abc' $
PATCH file1,  OUT file2,  AT START,    REMOVE 7, ADD 'xxx' $
```

The tab-delimited file written by EXCEL is an example of what is, from P-STAT's point of view, an external file. PATCH might be used when TABFILE.IN has rejected an input file at a certain byte location for some reason, and then SHOWBYTES, displaying the bytes at that location, has revealed what is amiss.

1. PATCH — is followed by the name of an existing external ASCII file to be read. The name may have a path. It usually will need to be quoted. There is no limit on size This file is not changed by the PATCH command.
2. OUT — is followed by the name for the new external file being created in this command.

There must be a specific location in the file where the ADD and REMOVE actions are to be done. Either AT or AFTER can be used to set the change location. When both REMOVE and ADD are used, the REMOVE action is done first and then the ADD.

1. AT START — This sets the change location so that it precedes the initial byte of the file. If three bytes are added to the file, they become the initial three bytes of the output file.
2. AT END — This sets the change location so that it follows the last byte of the file. If three bytes are added to the file, they become the final three bytes of the output file.
3. AFTER 123 — This sets the change location to a spot between two bytes within the file. AFTER 123 causes the the ADD and REMOVE changes to begin between bytes 123 and 124. If three bytes are added to the file, they would be inserted between those two bytes.
4. REMOVE 23 — This removes the 23 bytes that follow the change location. AFTER 90, REMOVE 5 would remove bytes 91 through 95.
5. REMOVE -3 — This removes the 3 bytes that precede the change location. AFTER 90, REMOVE -5 would remove bytes 86 through 90.
6. AT END, REMOVE -3 removes the final three bytes in the file.
7. REMOVE ALL — removes all of the bytes beyond the change location.
8. AFTER 622, REMOVE ALL would make byte 622 the final byte in the file. If ADD is also being done, the new bytes would then follow byte 622.

The characters to be added can be specified in three ways: strings, names of characters, and the integer value of ascii characters. These can be intermixed.

```
GIVEN:               abcdefgh
PATCH INSTRUCTIONS   AFTER 4, REMOVE 2, ADD 'XYZ'
PRODUCES             abcdXYZgh
```

Using "'xxx'" would cause 'xxx' to be added. Using '"xxx"' would cause "xxx" to be added. The supported names of characters include

```
HTAB  ( 9) horizontal tab     BLANK (32) blank    RPREN (41) the )
LF    (10) line feed          QUOTE (34) the "    COMMA (44) the ,
CR    (13) carriage return    APOST (39) the '    DASH  (45) the -
EOF   (26) end of file        LPREN (40) the (    DOT   (46) the .
```

1.5 THE EXAMINE COMMAND

EXAMINE 'external.file', EOC CR LF, OUT work1 $

EXAMINE reads an external input file and counts how many times each type of ascii byte occurs. It writes a report and, if requested, a P-STAT output file which has the number of times each of the 256 possible ascii characters was found.

An end-of-case (EOC) field like CR LF can be supplied. If so, the report is extended to show the number of cases in the file, the smallest and largest record lengths, and the number of characters, if any, after the final EOC.

The report summarizes what was found. It also gives the number of CR-LF combinations, and displays the first ten and last ten bytes in the input file. If, however, there are 20 or fewer bytes in the file, all are displayed.

Consider this unlikely 8-byte file (named test77).

```
byte 1: an uppercase A (ascii 65)
```

```
byte 2: CR, which means a carriage return (ascii 13)
byte 3: LF, which means a line feed       (ascii 10)
byte 4: an ascii 211
byte 5: an ascii 022
byte 6: LF
byte 7: an uppercase B (ascii 66)
byte 8: CR
```

Doing EXAMINE on this file produces the report shown below in Figure 1.7

Figure 1.7 EXAMINE Report

The Command

```
EXAMINE 'test77' $
```

The Report

```
-------------------EXAMINE completed---------------------
| Input file test77 has 8 bytes.                        |
|                                                       |
|     1 CR-LF was found.                                |
|     1 CR without a following LF was found.            |
|     1 LF without a preceding CR was found.            |
|     1 byte less than 32 (besides CR or LF) was found. |
|     1 byte greater than 127 was found.                |
|                                                       |
| The input file contains these bytes:                  |
|       A       CR      LF      211     022             |
|       LF      B       CR                              |
|                                                       |
| Time: less than 0.1 second.                           |
---------------------------------------------------------
```

An end-of-case field like CR LF (or on unix, just LF) can be supplied. The field can be as much as 100 bytes. The EXAMINE command will count the number of such fields, and assume that each concludes a case in the file. The longest and shortest such cases will be reported. Figure 1.8 shows the report when end-of-case (EOC) is used.

The OUT identifier can be used to request an output file, which is a P-STAT system file with 4 variables and at most 256 rows. A row is written for each different ascii character that was found inthe input file. The variables are:

1. VALUE the ascii value of the character (0 to 255).
2. CHARACTERthe character, like A or CR or COMMA.
3. COUNT how many times that character was found in the file.
4. FIRST.USEDthe location in the file where it was first found.

Using the 8-byte file “test77” as input, EXAMINE with the OUT identifier produces a P-STAT system file with the following contents:

```
                                first
 value  character  count    used

    10  LF             2       3
    13  CR             2       2
    22                 1       5
    65  A              1       1
    66  B              1       7
   211                 1       4
```

Figure 1.8 EXAMINE Report: End of Case is Used

```
---------------------EXAMINE completed--------------------
| Input file /usr1/newsource has 21,492,263 bytes.        |
|                                                         |
| 732,373 LFs without a preceding CR were found.          |
|                                                         |
| The input file begins with these 10 bytes:              |
|       =       A       P       O       COMMA             |
|       BLANK   p       r       o       g                 |
|                                                         |
| The input file ends with these 10 bytes:                |
|       BLANK   BLANK   BLANK   BLANK   BLANK             |
|       BLANK   e       n       d       LF                |
|                                                         |
| One end-of-case character was supplied: LF              |
|                                                         |
| 732,373 end-of-case fields were found.                  |
|         There were no extra bytes after the final EOC.  |
|                                                         |
|         These lengths do not include the EOC byte.      |
|       0 was the smallest record length.                 |
|      80 was the largest  record length.                 |
|                                                         |
| 169,682 cases had the smallest record length.           |
|  10,752 cases had the largest  record length.           |
-----------------------------------------------------------
```

SUMMARY

EXAMINE

```
EXAMINE 'SomeFile' $
```

EXAMINE reads an external input file and counts how many times each type of ascii byte occurs. It writes a report and, if requested, a P-STAT output file which has the number of times each of the 256 possible ascii characters was found.

Required:

EXAMINE 'fn'

supplies the name in quotes of an external file which is to be examined.

Optional Identifiers:

OUT fn

supplies the name for a P-STAT system file containing the counts of each character found.

EOC CR / LF / CRLF

An end-of-case (EOC) field like CR LF can be supplied. If so, the report is extended to show the number of cases in the file, the smallest and largest record lengths, and the number of characters, if any, after the final EOC.

PATCH

```
PATCH  External.file,  OUT  changed.file,  after 2336,
       REMOVE 3,  add 'fixed' $
```

PATCH is similar to REFORMAT. It is often used to fix a problem that was located by the SHOW-BYTES command.

Required:

PATCH 'fn'

supplies the name in quotes containing the text to be patched.

OUT 'fn'

supplies the nae in quotes for the resulting output file.

Optional Identifiers:

ADD Name / String / Integer

Characters to be added can be specified in three ways: strings, names of characters, and the integer value of ascii characters.

ADD blank ADD 'abc' ADD 32

AFTER **nn**

Supplies a byte location for the change to be made

AT **START / END**

Sets a location for the change to be made.

OUT **'fn'**

Supplies the name in quotes for the output file

REMOVE **nn**

The number of bytes to be removed.

REFORMAT

```
REFORMAT 'xxx1',  OUT 'xxx2',  REMOVE NL,  INSERT CRLF $
```

This is a utility command that reads an external file, removes, changes, and/or inserts characters as indicated, and writes the resulting external file.

Required:

REFORMAT **'fn'**

supplies the name in quotes of an external file which is to be changed in some way.

OUT **'fn'**

supplies the name in quotes of an external file which is to contain the output.

Optional Identifiers:

AFTER **nn**

specifies a pattern counter. Do this after every nth occurrence.

ANYWHERE

Occurences anywhere (even withing quotes) are included.

BYTES **nn nn**

is paired with values. For each byte location a value is given that is to replace what is found at that location. Up to 20 BYTE/VALUE pairs can be supplied.

FIRST **nn**

supplies the location of the first byte in the input file which is to be copied to the output file. If LAST is not supplied, the final byte of the file is assumed.

INSERT **cc**

Insert 1 or two of the specified characters. The characters can be any of CR (carriage return), LF (line feed), CRLF, NL (new lien), EOF (end-of.file), any integer, or any 1 or 2 characters in quotes.

LAST **nn**

REMOVE cc

Remove 1 or 2 characters. See INSERT for the characters that can be used.

VALUES cc cc

is paired with BYTES. Values supplied the character which is to replace whatever is found at that location. Up to 20 BYTE/VALUE pairs can be supplied.

SHOW

```
SHOW  Staff  $
```

SHOW lists the variables in a file, with their positions and their data types. The format, when the variable names are no larger than 16 characters usually has 3 columns on an 80-character screen, and 5 columns on a 132 column printout. With longer labels the display is always a single column. Use the TAGS general identifier if you have long labels and need other than single column output.

Required Identifiers:

SHOW fn

supplies the file name. If no file name is given, the most recently referenced file is used.

Optional Idenfiers:

SORT

the variables are arranged not by their position in the file. They are sorted alphabetically by the variable name.

ACROSS

the order in which the variables are displayed is across the page rather than down the columns.

COLUMNS nn

specifies the number of columns to be used. 2-5 is the maximum depending on the output width and the SPREAD/TIGHT setting. COLUMNS does not work with long variable labels.

NO BREAK

suppresses the normal break in the printout after every 5 lines. NO BREAK can be used with long variable labels

NO HEAD

suppresses the heading at the top of each printed page.

SINGLE

is the equivalent of COLUMNS 1.

SHOWBYTES

```
SHOWBYTES 'external.fil'
```

SHOWBYTES displays the bytes of a file. Three values are shown for each byte: its position in the file, its ascii value (0 to 255), and its ascii representation (if it is a printable character). Non-printable characters like carriage return and linefeed are represented by CR, LF, and so forth.

Required:

SHOWBYTES 'fn'

supplies the name in quotes of an external file which is to be changed in some way.

Optional Identifiers:

FIRST nn

Provides the start byte. If nn is negative, it is the number of bytes before the end. If LAST is not specified, the final byte in the file is assumed.

LAST nn

provides the final byte. If FIRST is not specified, byte 1 is assumed.

MAX nn

specifies the maximum number of bytes to be displayed.

FIND cs

defines a search string of 1 to 80 bytes. It can be composed of quoted strings, numbers or the special combinations CR (carriage return), LF (line feed), NL (new line), EOF (end-of-file).

PREVIOUS nn

specifies how many bytes before the selected FIND byte should be included. The default is 10.

CHUNK nn

specifies the number of lines to be printed before a break The default is 10.

GAP nn

specifies the number of spaces (1-10) between columns where each column has three values. The defaults is 1.

BREAK and NO BREAK

the default is to break between each chunk of lines. NO BREAK turns this off.

HEAD and NO HEAD

controls the printing of a heading giving the file name and the file size.

COLUMNS nn

specifies the number of columns per page. The default is the number that fits. The number cannot be larger than this amount

ZERO

specifies that the byte locations should be zero based. The default is to start at 1.

2
Utility Commands For General Usage

Utility commands set various parameters of a P-STAT session, such as page width, lines per page, input files and output destinations. This chapter discusses general utility commands that may be used in either interactive or batch modes. The next chapter discusses utility commands specifically for interactive use, and the chapter after that discusses those utility commands specifically for batch usage.

A number of the utility commands are self-explanatory. These commands are not discussed in the body of the chapter and appear only in the summary section at the end of the chapter.

2.1 BATCH AND INTERACTIVE RUNS

The most important utility commands are the two that tell P-STAT whether a job is interactive or batch. Some operating systems have batch as the assumed mode of operation while most assume the interactive mode. The BATCH command is used to commence batch execution and the INTERACTIVE command is issued to commence interactive execution.

On many systems, P-STAT can automatically determine whether a run is interactive or batch at the start of the run. However, there are some computers on which it is not possible to tell the difference between the two and there are times when you start one way and then decide to change. One indication that the system is not automatically distinguishing between batch and interactive usage is when P-STAT begins with a banner announcing its existence but does *not* issue a prompt for a command. At this point, the command INTERACTIVE must be issued to commence interactive execution:

```
P-STAT, version 3.01 (Aug 1, 2012)
WHOPPER 2 (6000 variable) size with storage options 222.
Copyright (c) 1972 to 2012, P-STAT Inc.
Use  HELP NEWS$  for general news about this version.

P-STAT starting... 16:10:15  Tue Mar 24  1998

INTERACTIVE $

P-STAT is now interactive.
Use   PROMPT SHORT$   for short prompts.

Enter a command:
```

The basic difference between a batch and and interactive run is the user's presence. In an interactive session, the user is able to correct errors and make decisions about options and the flow of the run. In a batch session, all decisions must be made before execution begins and errors may not be corrected during execution.

2.2 CONTROLLING THE PRINTOUT

Another important difference between batch and interactive computing is in the destination of printed output. In a batch run, all output usually goes to a printer or disk file. In an interactive session, output is usually displayed at the terminal. In P-STAT, the output destination may be changed as desired during the course of either batch or interactive runs. It is sometimes convenient to direct the printed output to a destination other than the primary output destination — for example, to a disk file. This disk file of output can then be printed, edited, or included directly in a report. When the results of an interactive session produce many pages of tables or listings, it may be better to direct that output directly to a high speed printer.

The terminal and the printer often have different physical capabilities. While most printers can print at least 132 characters per line there are some terminals that can handle only 80 characters. In order to take advantage of the capabilities of different print destinations, it is necessary to define print settings for the various output destinations.

Output is reformatted automatically when different settings for different output destinations have been defined. For example, a table that is printed first at the terminal with only an 80 character width may have an output width of 132 characters or more in an output file sent to a line printer.

2.3 PRINTER.SETTINGS and DEFAULT.SETTINGS

P-STAT begins a session writing its output to a terminal if interactive or to a printfile if batch. In addition, other output destinations or “printers” can be defined and used. Each of these printers has certain settings associated with it. These are:

1. OUTPUT.WIDTH — the maximum number of characters in a line Usually 80 on an interactive terminal and 132 on a printer file. The maximum size for a printline is 5,000 characters. This maximum is far greater than the output width used by most commands. OW is a synonym for OUTPUT.WIDTH.
2. LINES — The maximum number of lines in a page, set to 59.
3. ECHO — should the command itself be written to a printer? The default is no on a terminal, yes on the batch printfile, and no on other printfiles.
4. VBAR — the vertical bar character to be used. The default is the standard vertical bar. If the output destination has no vertical bar character the letter I is often a substitute
5. PAGE.CHARACTER — the carriage control character to be used for a page change. The default is a control L on most ASCII computers and “1” on the others. A blank can be used for printout that is to be stored on disk

All five keywords in the preceding list can be used in three different contexts:

1. as a command to change the setting of the current printer: OW 77 $. See the individual command descriptions for full details.
2. as an identifier in PRINTER.SETTINGS. This defines or changes the settings of a particular printer.
3. as an identifier in DEFAULT.SETTINGS. This changes the default settings of any printer whose initial reference occurs later in the session.

LINES and OUTPUT.WIDTH can also be used as general identifiers in any command. To print a single file with LINES set to 35 and OUTPUT.WIDTH set to 72 without changing the default printer settings:

```
LIST Myfile,  LINES 35,  OUTPUT.WIDTH 72 $
```

The PRINTER.SETTINGS command changes the settings associated with a printer. If a printer name does not follow the command name, the changes apply to the current printer. For example, the following command sets the current output destination to the default interactive settings:

```
PRINTER.SETTINGS,  OW 80,  LINES 59,
                   NO ECHO,  VBAR,  PAGE.CHARACTER $
```

VBAR and PAGE.CHARACTER, since they have no arguments, use the current default settings.

A new printer named “Results” is opened, and its settings are defined or changed by the following command:

```
PRINTER.SETTINGS 'Results',  PAGE.CHARACTER ' ' $
```

This command opens file “Results”, and notes that a blank will be used for a page control character. This has the effect of turning off a page change. The “Results” file is not yet active.

```
PR  'Results' $
```

activates Results as the current print destination. Use of:

```
LIST  X,  PR  'Results' $
```

activates it for the LIST command only.

The DEFAULT.SETTINGS command has the same identifiers as PRINTER.SETTINGS, but does not have any specific file name after the command name. For example:

```
DEFAULT.SETTINGS,  PAGE.CHARACTER  '1'  $
```

causes all printers that have not yet been referenced to have a “1” as the page control character. PAGE.CHARACTER is discussed later in this chapter. DEFAULT.SETTINGS can be used in your PSTART file to set the print settings to the best defaults for your environment.

The number of different destinations that can be assigned with the PRINTER.SETTINGS command depends on the size of P-STAT that is being used. In the Jumbo version, which is the one usually distributed, up to 20 different destinations can be defined. A print destination, unlike other files which are only open while they are in use, remains open from the time it is defined until it is explicitly closed or until the P-STAT session ends.

2.4 The Vertical Bar Character

Some printers or terminals may display P-STAT’s default vertical bar with a character that is unsatisfactory in some way.

```
VBAR  'I'  $
```

for example changes the vertical bar to the letter “I” for the current printer.

The character to be used must be enclosed in single quotes. The following command defines the colon as the vertical bar character:

```
VBAR ':'  $
```

The colon often produces better looking output than the “I”. Another way to change the vertical bar character is to use VBAR as an identifier in the PRINTER.SETTINGS command:

```
PRINTER.SETTINGS,  OUTPUT.WIDTH 80,  NO ECHO,
  NO UNDERLINE,    VBAR ':'  $
```

2.5 The Page Control Character

P-STAT uses either the character “1” or, in some Unix and Dos/Windows environments, a character representing a form feed as the first character in the first line of each print page. This character can be changed with the PAGE.CHARACTER command. The single argument to the PAGE.CHARACTER command is used as the page change character on the current printer until the next PAGE.CHARACTER command is issued. A PAGE.CHARACTER command with no argument returns the value to its default setting.

The argument can be either a decimal number or a single character in quotes.

```
PAGE.CHARACTER 12  $
```

This will use the character which is the equivalent of a decimal 12 (a form feed) as the page change control character.

```
PAGE.CHARACTER   ' ' $
```

If the page change character is set to a blank, there will be no page changes indicated in the printout. This is especially useful if you want several TEXT.WRITER commands to be string together without the final page change that each TEXT.WRITER command produces. The page control character can also be defined in the PRINTER.SETTINGS or DEFAULT.SETTINGS commands.

2.6 Changing the Output Destination

Usually the command procedure or executive routine that calls P-STAT defines the physical device (terminal, printer or disk file) that is the primary output destination. It is typically the terminal for interactive sessions. On most systems, this linkage is set up when P-STAT is installed so that the user does not have to supply this information. If the user wants to change this output destination to a device other than the primary output destination, the PR command is used:

```
PR  'PrtFile'  $
```

PR is an abbreviation of printer and tells P-STAT to change the output destination to a file named “PrtFile”. When the user wishes to change the print destination back to the primary output destination, the command:

```
PR $
```

is issued. If the PRINTER.SETTINGS command has not been used and output is directed to a destination other than the primary output destination, the PRINTER.SETTINGS setting has an output width of 132 and there is no vertical bar, or command echo.

In Figure 2.1, a correlation is done on four cases of the file with the output printed at the terminal. A correlation of the entire file is then done and a description file is produced. The PR command is used to change the output destination to a file named “PrtFile”. STUB is used in the LIST command so that the row label generated by the CORRELATE command prints on the left. The two LIST commands now send their printed output to the file PrtFile rather than to the terminal. Finally the output destination is reset to the terminal by:

```
PR  $
```

If the output destination is to be changed only for a single command, the PR command may be used as a general identifier:

```
LIST  Cor,  PLACES 2,  STUB,  PR  'PrtFile'  $
```

Note that the ECHO and NO ECHO identifiers in the PRINTER.SETTINGS command have no effect when the print destination is changed for a single command. If you wish your commands echoed in the printout use PR as a command to change the destination even if it is only for a single command.

Once the output has been directed to a disk file, the file may be printed. On most UNIX and DOS computers this command, issued from *within P-STAT*,

```
PRINT  'PrtFile' $
```

closes and prints the specified file on the primary printer. On other computers the file must be printed *after exiting* from P-STAT to the operating system. Once outside P-STAT, the individual operating system command for printing text files is used. This command differs from system to system. On some computers using UNIX, the command is:

```
lpr  PrtFile
```

On machines using DOS, the command is:

```
PRINT  Prtfile
```

Figure 2.1 Directing Output to Disk File "PrtFile" and then Back to the Terminal

```
PRINTER.SETTINGS  'PrtFile',   OUTPUT.WIDTH 132,   VBAR,
         UNDERLINE,   NO ECHO  $


CORRELATE  MyFile  [ CASES 1 TO 4 ],  OUT  Cor  $

 Correlate completed.
 4 cases were read.
 There was no missing data.


 LIST,  PLACES 2,  STUB  $

                                                 Grade
  row label            Age     Sex      Race  Average

  Age                 1.00    0.42    -0.11     -0.09
  Sex                 0.42    1.00     0.00     -0.09
  Race               -0.11    0.00     1.00     -0.44
  Grade.Average      -0.09   -0.09    -0.44      1.00


 CORRELATE  MyFile,  OUT  Cor,  DES  DesFile  $

 Correlate completed.
 432 cases were read.
 There was no missing data.
 2 passes were made through the input file.


 PR  'PrtFile'  $

 LIST  Cor,  PLACES 2,  STUB  $


 LIST  DesFile  $

  PR  $
```

If you are running under Windows you must first close the print file before you choose the print icon to sent the file to the printer. If you forget to close the file, you will print an incomplete or empty file. This is because the output gets "buffered" and only placed in the actual print queue when a large number of lines have been written. The command to close a file is CLOSE followed by the file name in quotes.

```
CLOSE 'PrtFile' $
```

If you close a print file, any parameters in a PRINTER.SETTINGS for that file are forgotten. You must reissue the PRINTER.SETTINGS command if you wish to reuse that print file with the original settings.

When output is sent to a different print destination, it is reformatted to take advantage of the attributes of that destination. This means that listings and tables contain more columns of information if they are sent to a printer with a width of 132, than they do on the terminal set to a width of 80. Similarly, plots and histograms are reformatted to use the entire printed page. If the output on the alternate printer is to look exactly like the output on the primary printer, be sure to make the settings the same. It is not necessary to use all 132 columns of the printed page if only 80 columns are desired.

2.7 Setting the Number of Lines of Output

LINES can be used in both batch and interactive runs. LINES sets the number of lines on the output page. This controls page changes, printing of titles and headings, and the layout of frequencies, plots and histograms. When LINES is used as a *command*, all subsequent output on the **current** printer, such as histograms and listings, use that setting:

```
LINES 50      $
HIST   MyFile  $
LIST   MyFile  $
```

When LINES is used as an *identifier* within a command, it applies to that command alone. In the following example, the histograms are all formatted to fit within a page of 40 lines, yet the list is formatted for a page with 30 lines:

```
LINES 30  $
HIST  MyFile,  LINES 40  $
LIST  MyFile   $
```

In an interactive run, commands such as PLOT format their terminal output to fit within the SCREEN size unless LINES is used as a subcommand. (See the SCREEN command in the next chapter.)

2.8 Using General Identifiers

Identifiers that may be used in any command are considered to be *general* identifiers. Some of the general identifiers are ERROR, LINES, OW, PAGE, PR, TEMP and VERBOSITY. See the index entry “General identifiers” for other general identifiers. PAGE causes a page change at the end of the command. TEMP indicates that the command and its associated data should not be added to the editor file. TEMP is only useful in interactive runs. VERBOSITY sets the verbosity level to a number between 1 and 4. Level 1 produces less output; level 4 produces a great deal of output. VERBOSITY can be abbreviated to the singles character ‘V’.

General identifiers can be used in any command. For example:

```
LIST  Cor,  PR  'PrtFile'  $
```

This command causes the output from the LIST command to be printed in the external file named “PrtFile”. When general identifiers are used within a command, they apply to that command alone. ERROR, LINES, OW, PR and VERBOSITY may also be issued at the command level to set parameters for all subsequent commands. Settings defined at the command level remain in effect until they are specifically reset.

Other general identifiers which control how variable names are presented are: TAGS, FULL, TEXT, SHORT.TAGS, SHORT.16, SHORT.OLD. These general identifiers can only be used as identifiers, not as commands.

1. FULL — use the full label
2. TEXT — use text or full label
3. TAGS — get tags or 1st 16 character.. (also SHORT.TAGS).
4. SHORT.16 — use first 16 characters.

5. SHORT.OLD get first 16 characters as legal names for the older version 2

2.9 BUILDING A DICTIONARY

SHOW which was documented in the previous chapter lists the names of the variables with their positions and their data types. If a P-STAT file has full variable names with tags and descriptive text, the SHOW command can be used to produce a very satisfactory dictionary. If, however, the variables in the file have short labels with a corresponding “labels” file, a dictionary can be built from the labels file. In Figure 2.2, the SHOW command used with NO HEAD, COLUMNS 1, and NO BREAK is just the first step needed to join the information from the labels file and produce a dictionary.

Figure 2.2 Creating a Dictionary of Variables

The commands

```
1.  PRINTER.SETTINGS 'dict.prt', PAGE.CHARACTER ' ' $
2.  SHOW paceold, NO BREAK, NO HEAD, COLUMNS 1, TAGS,
         PR 'dict.prt'$
3.  CLOSE 'dict.prt' $
4.  MAKE work, FILE 'dict.prt',
               DEL BLANK;
          Num Type:c Name:c $
5.  SUBSTITUTE.XL work [IF Num MISSING, DELETE], VAR name,
         LABELS 'paceold.lab', OUT work2 $
6.  JOIN work work2 [ RENAME name TO Label ], OUT dict $
7.  LIST dict [ ROWS 2 .ON. ], NO FOLD $
```

The result

```
Num  Type  Name           Label

  1  D     ID             ID
  2  D     Age            Age
  3  D     Sex            Sex
  4  D     Income         Income
  5  D     Num.TV         Number of television sets in your home:
  6  D     VCR            Video Cassette Recorder
  7  D     CD             Compact Disc Player
  8  D     Port.Phone     Portable Telephone
  9  D     Ans.Mach       Answering Machine
 10  D     Store.1        Type of store item purchased in:
 11  D     Store.2        Store.2
 12  D     Store.3        Store.3
 13  D     Store.4        Store.4
 14  D     Income.Groups  Income of respondent
 15  C16   own            Respondent ownership
```

The first four commands are used to get the SHOW output into a P-STAT system file.

1. The PRINTER.SETTINGS command is used so that the SHOW print file will not have any special page change characters.
2. The SHOW command itself has no breaks or headings and is a single column which is written to a print file
3. The CLOSE command flushes the print buffers so that all of the printout is available.
4. The MAKE command uses the print file as input to create a P-STAT system file with three variables: the position of the variable, its data type and its name.

The fifth command uses the SUBSTITUTE.XL command to create a new P-STAT system file with the extended variable labels replacing the values of variable name. There are now two P-STAT system files. Both have the three variables Num, Type and Name. In order to join the files together to get a four variable output, variable Name must be renamed in one of the commands as the files are input to the JOIN command.

Figure 2.2 shows the final result with the variables ordered by their input positions in the original file. When the file has many variables a more useful arrangement may be to have the variables in alphabetical order by the variable name. This is easily accomplished by adding SORT to the SHOW command and changing the JOIN command to a LOOKUP.

```
SHOW paceset, NO BREAK, NO HEAD, COLUMNS 1, SORT, ....
LOOKUP work, TABLE work2 [ RENAME name to Label ], ....
```

2.10 THE TRANSFER COMMAND

The TRANSFER command is used to submit an external file containing one or many commands for execution. Because the transfer file is an external file, P-STAT expects to find it on the PSDATA directory. (This is also true for the SHOWBYTES, REFORMAT and EXAMINE commands discussed in the previous chapter.) Directories are discussed in full in “P-STAT: Introductory Manual”. Files are either:

1. P-STAT system files stored in the PSAUTO directory
2. external data files such as the PSTART file discussed later in this chapter, which are stored in the PSDATA directory or
3. temporary files, created during a P-STAT session and erased when it ends, stored in the PSTEMP directory.

These directories may be the same, defined as a single PSFILES directory. If the transfer file is in the PSDATA directory, the file name in quotes is all that is needed to locate it. If the input to any of these commands is in some other directory, the full path and file name in quotes is required. When a P-STAT run is aborted without the terminating ‘END $’ command any temporary files created during the run are not deleted and your directory may be littered with files starting with ‘W_...... PS1 or W_.....TMP. These files can be freely deleted at the end of a run or they can be placed in a folder pointed to by the PSTEMP command at the start of a run.

2.11 The TRANSFER Command

In an interactive run, the input normally comes from the terminal. In a batch run, the input normally comes from the standard input device. This is typically a disk file — also called an external file. These may be commands created using the operating system editor or commands stored by using the EXPORT instruction in a previous interactive P-STAT run. (See the second editor chapter, “The Editor in Detail,” for information on EXPORT.)

The TRANSFER command may be used in either batch or interactive runs to transfer control from the normal input stream to an external file. The commands in that external file are then executed, one at a time until control is returned to the primary input device by a RETURN command or the run ends with an END command.

The TRANSFER command may be used to transfer control to any file of P-STAT commands. The file of commands may itself call a TRANSFER to a file which also has a TRANSFER. There is no limit on the depth of the transfers except your ability to construct and understand them. TRANSFER can also be used within a macro.

Consider these commands in a disk file named “AddCases”:

```
CONCAT  Master  NewCases,
        OUT  NewMas,  DES  NewMasD  $
LIST    NewMasD  $
RETURN  $
```

The TRANSFER command is issued to transfer control to the file named “AddCases.trn”:

```
TRANSFER  'AddCases.trn'  $
```

Each of these commands are executed in turn and control is given back to the primary input device (terminal) when the RETURN command is encountered. Note that if the RETURN command is the last command in the transfer file, control returns to the primary input file. If it is a batch run and the transfer file does not end with RETURN, the run terminates as if the END command had been executed. If the run is interactive, and there is no RETURN, control is returned to the terminal regardless.

Groups of commands each ending with “RETURN, HOLD $” can be stacked in a transfer file and executed in sequence. HOLD causes the TRANSFER file to maintain its position rather than being repositioned at the beginning. The use of the identifier EXIT in the TRANSFER command causes the run to end when the RETURN$ is executed.

```
TRANSFER  'My.trn',  EXIT $
```

2.12 Errors in Transfer Files

When a TRANSFER command is executed, the external file becomes the input stream. Commands, subcommands, and data records are processed in the same way they are processed from the primary input source. As always, the handling of errors depends on whether the run is interactive or batch.

When an error occurs in the transfer file during a batch run, an error message is printed and execution continues with the next command. The run terminates if a fatal error is encountered or if there are too many consecutive errors. A fatal error is one that appears to invalidate all subsequent commands. The limit for consecutive errors may be controlled with the MAXERROR command. It is set to 5 at the beginning of a batch run. (See the chapter on batch usage for more information on the MAXERROR command.)

When the run is interactive, an error in the transfer file causes an error message to print at the terminal. Control is given to the P-STAT editor so that the command may be corrected and re-executed. Any subcommand and data records that follow that command are also placed in the edit file. The error message will be something like:

```
Error... BUILX is not the name of a P-Stat command.

An  error has occurred while a transfer file is active.
The error can be corrected and re-executed from the editor.

After that, should processing continue from the transfer file ?
Enter YES or NO:
```

If you reply “yes”, you are placed in the editor where the error can be corrected.

```
 EDITOR entered.
    5...BUILX work,
           DATA dict.prt;
 EDITOR:
c/X/D
    5...BUILD work,
           DATA dict.prt;
 EDITOR:
x
```

Now, if the BUILD command is successful, control returns to the transfer file. The transfer file still contains the error even though you fixed the text in the editor.

2.13 PSTART

The PSTART file is nothing but a transfer file that is executed before any other commands in a P-STAT run. If you find that there are several commands that you always enter at the beginning of a P-STAT session, it makes sense to store them in a single file rather than re-enter them individually every session. You can create such a file and use the TRANSFER command yourself. The advantage of designating that file as a PSTART file is that you do not have to remember to do the transfer. P-STAT does it for you.

The PSTART file is found in one of two places:

1. If there is a PSTART environment variable and the file is found, that is the file that is used. Environment variables are described in detail in the first chapter of “P-STAT: Introductory Manual”.
2. The second choice is a file named PSTART or pstart in the PSDATA directory. P-STAT’s use of directories is described in full in “P-STAT: Introductory Manual”.

Figure 2.3 PSTART File: Prompt for the Autosave Directory

```
File PSTART

GEN ##Adir:c80 = ' ' $
GEN ##Ddir:c80 = ' ' $
DIALOG ##Adir
'-------------------------------------------------'
'Enter the full path for your autosave directory' $
IF .RESPONSE. .NE. 16 BRANCH exit $

PSAUTO '##Adir' $

DIALOG ##Ddir
'--------------------------------------------'
'Enter the full path for your data directory' $
IF .RESPONSE. .NE. 16 SET ##Ddir = ##Adir, BRANCH exit $

PSDATA '##Ddir' $

exit: PUT <<  >> $
PUT <<The autosave directory is >> ##Adir $
PUT <<The directory for external files is >> ##Ddir $

NO MENUS $
ALLOWABLE.NAME 10 $
DEFAULT.SETTINGS, PAGE.CHARACTER 12, NO ECHO, OW 80, LINES 56 $
RETURN $

----The P-STAT Run Executes the PSTART File ---------

 P-STAT, version 2.23 Rev 3 (Mar 18, 2005)
 WHOPPER 2 (6000 variable) size with storage options 222.
 Copyright (c) 1972 to 2005, P-STAT Inc.
 Use  HELP NEWS$  for general news about this version.
```

```
 P-STAT starting... 17:06:33  Tue Mar 31  2005
 -----------------------------------------------
 Enter the full path for your autosave directory
/usr2/project2/psf
 -------------------------------------------
 Enter the full path for your data directory
/usr2/project2/data

 Your autosave directory is /usr2/project2/psf
 Your directory for external files is /usr2/project2/data

 The maximum length of an autosave file name,
 which has been 16 characters, is now 10 characters.
```

If there are many commands that are always executed at the beginning of each P-STAT session, it makes sense to install PSTART as an environment variable. If the contents of the PSTART file change with every project, it should probably live in the PSFILES/PSDATA directory.

PSTART is the one P-STAT environment variable that is not set as part of the Windows installation process. It is up to the user to decide if there is a need for a PSTART file, what it should contain; where it should reside; and if it should be defined as an environment variable. Since the full path and file name are provided when you create the PSTART environment variable, the file name can be any name of your choice.

Perhaps the most flexible way to use the environment variable/command combinations is to use environment variables for the settings that will probably not change: PSMENU, PSHELP and PSTEMP. of any files that have been left around from previous runs.

If you do not have a PSTEMP environment variable or wish to use a different directory, the PSTEMP command must (along with PSFLAG) be at the beginning of the PSTART file. PSTEMP is included here because having a directory designated for temporary files that can be used by all the programs you install has the big advantage that you do not have to back it up, and there is no risk when you boot your system in cleaning

It is also a good idea set a PSFILES environment variable. This gives P-STAT a known base which can be easily changed with a PSTART file in that directory. If you have very few different projects, the PSFILES directory may be where you want your P-STAT system files and external data files to live. If you are working on many projects, the PSFILES directory might contain nothing but a PSTART file which issues your start-up commands and sets the PSAUTO and PSDATA directories for your current project. If the PSFILES command is used at the start of the run, PSTEMP, PSAUTO and PSDATA are all set to the PSFILES values. If the PSFILES command is used durning a run only the PSAUTO and PSDATA directories are changed.

If you are working on many projects simultaneously, you can design a PSTART file that prompts for the directories to use in the current session. Figure 2.3 contains a sample pstart file which prompts for the PSAUTO and PSDATA directories. The PSTART file can be as simple or complex as you wish. For example, instead of a simple DIALOG, a macro could test the results and prompt again after an invalid reply.

The first commands in the PSTART file, create two scratch character variables, ##Adir and ##Ddir, to hold the directory names. The DIALOG commands follow. A DIALOG command is composed of a scratch variable name to hold the reply, and the text of the dialog which is contained in a series of quoted strings

```
DIALOG ##Adir
'-----------------------------------------------'
'Enter the full path for your autosave directory' $
```

At this point nothing happens until the user enters some sort of a reply. That reply is stored in the scratch variable and a system variable .RESPONSE. is set which indicates the type of response. A value of 16 indicates that the response is a character string.

```
IF .RESPONSE. .NE. 16 BRANCH exit $
```

If an invalid reply is made to the DIALOG, a branch is made out of the DIALOG sequence and the remaining commands in the PSTART file are executed. In Figure 2.3 the reply that is made to the first DIALOG is:

```
/usr2/project2/psf
```

Since this is a valid reply, it is stored in scratch variable ##Adir and the command “PSAUTO '##Adir' $” is executed as:

```
PSAUTO '/usr2/project2/psf' $
```

The second DIALOG is similar. This time, if an invalid reply is found, the PSDATA directory is assumed to be the same as the PSAUTO directory. Following the two DIALOG commands, PUT statements tell the user what has been done.

Finally the PSTART file contains other commands that are to be executed when a run starts. Figure 2.3 also shows the beginnings of a P-STAT session using the PSTART file and the replies that are made.

SUMMARY

BYPASS

```
BYPASS  'string'  $
commands or text to be bypassed
follow until
string
```

Bypass is used in a transfer file when only part of a job needs to be executed. The parts that are not to be executed can be bypassed and the command stream left intact. A record with a single $ ends the command unless BYPASS is followed by a character string. The string is enclosed in single or double quotes. If a character string is provided, BYPASS ends when exactly those characters are found at the beginning of a record.

C

```
C  'Supply comment character string'  $
```

The Comment command may be used any time a command is expected. The text is enclosed in single or double quotes. A more flexible way to enter comments which may be used between commands, within a command, within PPL and even within subcommands is:

```
/*  Comment text which can even cross lines.  It continues
    until the final star/slash is found */
```

CLOSE

```
CLOSE 'external.fil' $
```

The CLOSE command can be used to close any open external file. Most external files such as labels files or data input to the BUILD command are automatically closed when the command completes. Print files are an exception. They are not closed until the end of the run.

COMMAND.WIDTH

```
COMMAND.WIDTH 72  $
```

COMMAND.WIDTH is used when the input commands are not 80 characters long. This is useful when the commands have been built in a system text editor which sequences the file in columns 73 - 80. It is also useful when the terminal has a width greater than 80 characters. Note that if a COMMAND.WIDTH of exactly 72 is requested, P-STAT assumes that data records from the input stream also have only 72 usable characters. COMMAND.WIDTH may be set from 40 through 132.

DEFAULT.SETTINGS

```
DEFAULT.SETTINGS,  ECHO, PAGE.CHARACTER 12,   OUTPUT.WIDTH 132 $
```

DEFAULT.SETTINGS causes all printers that have not yet been defined to use the specified values as the default values.

Optional Identifiers:

ECHO

causes commands to be echoed or printed before they are executed. NO ECHO causes no echoing.

LINES nn

defines the number of lines per page

OUTPUT.WIDTH nn

defines the width of the output destination where nn is a number between 70 and 132.

PAGE.CHARACTER nn / ‘c’

defines the page change (i.e., form feed) character to be used. The argument can be a single character in quotes like “1”, or an integer such as 12 (an ascii form feed character). If no argument is supplied or if PAGE.CHARACTER is not used, the default character will be used.

UNDERLINE

causes underlining of appropriate output segments to occur. NO UNDERLINE can be used to turn underlining off

VBAR

causes the character defined as the vertical bar for your machine to be used. A character in quotes may be supplied. If the argument is a single character in quotes, that character is used as the vertical bar character.

ECHO

```
ECHO  $
```

ECHO causes commands to be echoed on the current printer before they are executed. ECHO may also be used as an identifier within the PRINTER.SETTINGS command. ECHO may be turned off by using NO ECHO $.

END

```
END  $
```

END ends a P-STAT run. If VERBOSITY (V) is set to 4, a summary of input and output activity is printed.

ERROR

```
ERROR   LONG   $
ERROR   SHORT  $

CORRELATE  NYC.Pop ,  ERROR LONG $
```

ERROR changes the error setting and may be given at any time during a run. ERROR is a command and also a general identifier. It may be used within any command to change the error setting for that particular command. The most frequent use of ERROR is in an interactive run when the short error message does not provide enough information to locate the problem.

Arguments:

SHORT

causes error messages to be short. This is the assumed setting in an interactive run.

LONG

causes error messages to be long. This is the assumed setting in a batch run

ERROR.UNIT

```
. ERROR.UNIT  'External'  $
```

ERROR.UNIT causes all error messages and the closing messages printed by the END command to be printed both in an error file and on the primary printer. This is usually used when PR has set the print destination to some device other than the terminal. (When the output is formatted for PostScript, error messages go only to the error file and not also to the printer.)

HEAD

```
HEAD  'Provide a Heading for a System File'  $
```

Headings may be up to 88 characters long. The heading is carried with the file and is printed when the file is listed. Headings may be changed any time during a run. Titles, when they are used, replace headings.

HELP

```
HELP  $
HELP  EVERYTHING,  PR  'PrtFile'  $
```

HELP provides general information about P-STAT and using the HELP command. It may be used in either batch or interactive runs. HELP is provided for the most commonly used commands and for some general topics such as LEGAL.NAMES and STARTING.A.RUN. HELP COMMANDS $ gives a list of the commands that are documented in the help file. HELP TOPICS $ gives a list of general topics. HELP EVERYTHING $ produces a printout of the entire help file. The HELP commands are not placed in the edit file.

HELPFILE

```
HELPFILE 'C:\PSTAT\PSTAT.HLP' $
```

The HELPFILE command provides the full path and name of the P-STAT help file. This is only needed when there is an installation problem and the help file is not where the program expects it to be.

LINES

```
LINES 30  $
PLOT  MyFile,  LINES 24  $
```

The LINES command sets the number of lines per page for the current printer. LINES may also be used as an identifier within any command to change the number of lines per page just for the duration of that command.

NEWS

```
NEWS  $
```

NEWS prints a short report on the new features in the version of P-STAT that is being used.

NULL

The NULL command does nothing but provide a space holder to be used, perhaps, as a branch in a macro.

OUTPUT.WIDTH

```
OUTPUT.WIDTH 120  $
```

OUTPUT.WIDTH is used to set the output width. The argument may be any number between 70 and 132. If OUTPUT.WIDTH is not used, a width of 132 is assumed except for the primary output device in an interactive session, which is assumed to have a width of 80. Many commands accept wider output widths. OW is an abbreviation.

PAGE.CHARACTER

```
PAGE.CHARACTER ' '$
```

The character used to indicate a page change control character for the current output device may be explicitly changed by supplying either a character in quotes or a decimal number which represents the character to be used as the page change character, such as a 12 for an ASCII form feed. PAGE.CHARACTER$ resets the page change character to its original value.

PR

```
PR  'PrtFile'  $
PR  $
```

PR changes the print destination of output. The normal (primary) output destination is the terminal in an interactive run and the printer in a batch run. When PR is used without a file name, as in the second example, PR restores printing to the primary output device. PR can also be used as an identifier within any command to change the output destination for just that command.

PRINT

```
PRINT 'PrtFile' $
```

PRINT closes the designated print file and then executes either a DOS BAT file (PSTATLP.BAT) or a Unix shell script (pstatlp). Examples of these files are available in the directory where P-STAT is installed. They should be edited to properly access the local printer or print queue.

PRINTER.SETTINGS

```
PRINTER.SETTINGS  'Printer',  ECHO, PAGE.CHARACTER 12,
    OUTPUT.WIDTH 132,  VBAR   ':',   NO UNDERLINE  $
```

PRINTER.SETTINGS sets the parameters for a print output destination. If a file name is not supplied, the settings are defined for the current output destination. PP is an abbreviation.

Required:

PRINTER.SETTINGS 'fn'

supplies the name of the output destination. If the name is omitted, the settings apply to the current printer.

Optional Identifiers:

ECHO

causes commands to be echoed or printed before they are executed. NO ECHO causes no echoing.

LINES nn

defines the number of lines per page

OUTPUT.WIDTH nn

defines the width of the output destination where nn is a number between 70 and 132.

PAGE.CHARACTER arg

defines the page change (i.e., form feed) character to be used. The argument can be a single character in quotes like “1”, or an integer such as 12 (an ascii form feed character). If no argument is supplied or if PAGE.CHARACTER is not used, the default character will be used.

UNDERLINE

turns underlining on. NO UNDERLINE turns underlining off.

VBAR

causes the default vertical bar character to be used. A character in quotes may be supplied. If the argument is a single character in quotes, that character is used as the vertical bar character.

SHOW.ENV.VARS

This command produces a report of the environment variables that are currently defined.

SHOW.KEY

This command produces a report showing the current PSKEY and the module settings that are supported.

SLEEP

```
SLEEP 60 $
```

This command causes P-STAT to pause for the indicated number of seconds. The argument should be an integer, like 30.

```
AGAIN: SLEEP 5 $
INQUIRE.EXTERNAL 'abc', VERBOSITY 0 $
IF .XINQUIRE. EQ 0, BRANCH again $
```

The above commands could be used in a macro that needs file 'abc' to be available before it continues. Using SLEEP avoids incessant inquires.

TEXT

```
TEXT ;
The following commands plot regression residuals.
$
```

TEXT is used for documentation purposes. The lines of text that follow the command are printed exactly as they are encountered. The command is ended by a $ at the start of a record. The generalized comment form of /* */ can also be used for this purpose.

Optional Identifiers:

CLEAR

requests that the screen be cleared before displaying the text.

TRANSFER

```
TRANSFER  'ComFile'  $
```

TRANSFER is used to transfer control to a file of P-STAT commands. The argument is the name of the external file that has the command stream. Control is returned to the normal input file when the command RETURN is encountered in the transfer file.

Optional Identifiers:

EXIT

causes the transfer file to exit from P-STAT when the transfer is complete.

UNDERLINE

```
UNDERLINE  $
```

The UNDERLINE command is used to turn underlining on or off. Underlining is usually not possible at a display terminal. NO UNDERLINE turns underlining off.

VBAR

```
VBAR ':'  $
```

VBAR is used to change the vertical bar character. The vertical bar character may be changed by specifying any character available on the printer. The character must be enclosed in single quotes.

VERBOSITY

```
VERBOSITY 3   $

ANOVA  J104,  VERBOSITY 4 ;
  MODEL  MPG =  Weight | Origin,  MEANS  $
```

VERBOSITY (V) changes the verbosity level. The verbosity level controls the amount of printout produced. The argument can be from 1 to 4. 1 is used for minimal printout while 4 is used to print everything possible. The normal setting for interactive runs is 2. The normal setting for batch runs is 3. VERBOSITY may be used either as a command or as an identifier within other P-STAT commands.

VERSION

```
VERSION  $
```

The VERSION command prints the P-STAT banner that gives the date and version number of the P-STAT software.

General Identifiers:

IDEN

```
LIST,  IDEN  $
```

IDEN is a general identifier that may be used in any command to list all the identifiers for that command. When IDEN is used in an interactive run, the command is not added to the editor file.

LINES

```
list,  lines 60 $
```

LINES can be used as a general identifier in any command to set the number of lines per page during that command.

PAGE

```
LIST  Exper591,  PAGE  $
```

PAGE is a general identifier that can be used within any command to cause a page change before the next command is processed.

PR fn

```
LIST  Exper591,   PR  'Exper591.prt' $
```

PR can be used as a general identifier in any command to change the output destination. If it is used without an argument the default printer is used.

OUTPUT.WIDTH

```
LIST  Exper591,   OUTPUT.WIDTH  132 $
```

OUTPUT.WIDTH can be used as a general identifier for any command. It sets the width for the printout during that command. OUTPUT.WIDTH may be abbreviated to OW

.

3
Utility Commands
For Interactive Use

There are a number of commands that are mostly of use in interactive runs. They fall into three general categories:

1. Commands and identifiers that provide help in using P-STAT;
2. Commands that control the interactive environment;
3. Commands that control the edit file.

Many of the utility commands are self-explanatory. These commands appear in the summary section at the end of this chapter.

The most important command for interactive use is INTERACTIVE. This command indicates that the run is interactive, with someone at a terminal to answer questions and correct errors. In most environments, P-STAT assumes that a run is interactive when the session begins. If P-STAT begins with a banner at the top followed by the prompt:

```
Enter a command:
```

the session has been recognized as interactive. In such cases, the INTERACTIVE command does not need to be explicitly issued; it is generated automatically. If, however, the run begins only with a banner announcing P-STAT and there is no request for a command, INTERACTIVE must be issued to begin interactive execution.

3.1 CONTROLLING THE INTERACTIVE ENVIRONMENT

Many of the commands described in the previous chapter may be used in either batch or interactive modes to define the attributes of the primary input and output devices. In a batch environment, the characteristics of the terminal are of little importance, even when the batch run is initiated from a terminal. However, when a run is interactive, both the primary input device and output device are the terminal. The INTERACTIVE command not only initiates prompting and begins an edit file, but also defines the terminal as a device with an output width of 80 and a screen size of 22. These attributes may be changed if they are not appropriate for your terminal.

3.2 Starting an Interactive Run

When the assumed settings are not correct for your environment, they need to be reset each time P-STAT is used. It is tedious to have to reenter the same *initializing* commands from the terminal every time an interactive session begins. The best way to initiate a run is to store any frequently needed P-STAT commands in an external file. These commands can then be executed at the beginning of the run by using the TRANSFER command. Even easier, initializing commands may be stored in a PSTART file. PSTART is discussed in full in the previous chapter.

When the P-STAT software is evoked, any commands in a file provided by the PSTART file are considered initializing commands. Commands in the start-up file are automatically executed *first* (before the command prompt appears). P-STAT users with menu systems often include:

```
NO MENU $
```

in their start-up file if they regularly prefer to work in command mode, or if they are redirecting or piping batch jobs directly into and out of P-STAT. (Menu mode is for interactive use only.)

Figure 3.1 shows a typical series of commands for starting an interactive P-STAT session in the start-up file “PSTART”. RETURN is the P-STAT command that transfers control back to the primary input device (the ter-

minal in an interactive session). The only command in Figure 3.1 that is absolutely necessary for an interactive session is INTERACTIVE and in most systems it is generated automatically. However, if the terminal does not conform to P-STAT's initial settings, the printed output may not be satisfactory until the settings for the terminal are redefined.

Figure 3.1 Commands to Begin an Interactive Run in the PSTART File

External File PSTART:

```
PSAUTO 'D:\PROJIACM\PSF' $
PSDATA 'D:\PROJIACM\DATA' $

INTERACTIVE  $
SCREEN 22  $

DEFAULT.SETTINGS,  OUTPUT.WIDTH 80,   VBAR,  NO UNDERLINE,  NO ECHO,
                   PAGE.CHARACTER $

PRINTER.SETTINGS   PrtFile,
                   OUTPUT.WIDTH 132,  UNDERLINE,     ECHO,
                   PAGE.CHARACTER 12$

RETURN  $
```

The DEFAULT.SETTINGS command is used to define the default attributes for all output destinations. In this example, the assumed output width is set to 80, underlining and echo are turned off, and the system supplied vertical bar and page control characters are used. PRINTER.SETTINGS is used to define an output destination named “PrtFile” with an output width of 132, underlining and echo turned on, and a page control character set to an ASCII 12 (form feed). Because the vertical bar character is not mentioned, it assumes the default value. See the previous chapter for a complete discussion of output destinations and use of the PRINTER.SETTINGS and DEFAULT.SETTINGS commands.

3.3 Controlling Output on the Terminal

Most computer terminals scroll lines off the screen when they become full. The SCREEN command is designed to control the scrolling so that you have time to read the screen's contents before proceeding. This command is often used when initializing a run. It defines the number of lines that can print before the screen is full. When SCREEN has been set, P-STAT stops sending output to the terminal when the screen is full or when the next group of lines would cause output to scroll off the screen. The message:

```
HOLDING...
```

appears and the screen remains frozen. When the HOLDING message appears on the screen, the options are to enter a null line, hit the return key or enter one of the appropriate QUIT options listed below (Q, Q2 or Q3). Since one or two lines should be allowed for the system HOLDING message, SCREEN 22 is a reasonable setting for a 24 line terminal.

There are three possible QUIT options available to stop screen holding. They are:

1. Q QUIT the printout;
2. Q2 QUIT the current command;
3. Q3 QUIT the current command, and any expanded macros or editor-generated commands.

QUIT (or Q), by itself, is used to quit the current activity when there is no question as to what activity should be terminated. Normally, this is the situation that exists when only one command has been executed. QUIT is interpreted as Q1 in such a case. However, when more than one command has been executed (X4) and Q1 is entered, P-STAT needs to know: 1) whether to only quit the current command but to continue processing all others, or 2) whether to halt execution of all commands. In such ambiguous situations, the user is prompted for either Q2 or Q3:

```
        2 quit levels are possible here.

          Use  Q1  to quit this section of output
                   while continuing the command.
          Use  Q2  to quit the current command.

          Use  Q3  to quit the current command,
                   as well as any expanded macros,
                   active transfer files, subfiles loops,
                   or editor-generated commands.
       holding...
```

QUIT is interpreted as Q1 unless Q2 or Q3 are possible.

When more than one command has been executed, Q2 quits the current command but continues to process all others. Q3, on the other hand, stops processing all commands and printout. In addition, Q3 is used to halt execution of macros and to stop execution when X* has been issued.

3.4 Using LINES to Control the Output

LINES, as either a command or an identifier, is used in both batch and interactive runs. It is described in some detail in the prior chapter. The SCREEN command is used in interactive runs only. The LINES and SCREEN settings interact to some degree in interactive usage.

LINES sets the number of lines on the output page, and SCREEN sets the number of lines on the terminal. Some programs such as HIST, the histogram command, format their page layouts according to both the output width and the settings for LINES and SCREEN. If the run is interactive, the SCREEN setting is used to format the page layout unless LINES is used as an identifier in the command. If LINES is used, it determines the layout. This is useful when the screen is too small to convey all the information that is present. The display is laid out according to the LINES setting, but the SCREEN setting still prevents it from scrolling off the terminal.

Graphics command such as PLOT, HIST and EDA use the SCREEN setting as the formatting control. A picture that requires multiple screens is not going to be a clear picture. When these commands are re-run and sent to a printer, they are automatically reformatted.

A command such as LIST or SURVEY uses the LINES settings to control page changes and titles, and the screen setting so that the output does not scroll of the screen before the user has a chance to read it. If LIST or SURVEY output is sent first to the terminal and then to a printer, changes will only occur if the output widths are different. Wider output widths mean that LIST can show more variables on a page and that SURVEY can show more columns of the table. This reformatting is done automatically when the print destination is changed.

3.5 Re-executing Commands

AGAIN is a P-STAT command that may be used to execute a command a second time. It is similar to using EXECUTE (X) within the editor. Both EXECUTE (issued within the editor) and AGAIN (issued as a command outside the editor) process all subcommand and data records “owned” by a given command. If the re-execution command needs additional subcommands or data, the user is prompted for it. These new records are placed appropriately in the edit file. Thus, the edit file grows as new information is supplied.

The differences between the editor instruction X and the command AGAIN are that:

1. X can be used to execute a number of commands while AGAIN executes only the most recent command,
2. X is used only within the editor while AGAIN is used only outside the editor, and
3. X executes the command exactly as it stands while AGAIN can execute it with modifications.

Often some change is needed when a command is re-executed. AGAIN can be used with any of the system's general identifiers. For example:

```
 AGAIN,  LINES 56,  PR  'PrtFile'  $
```

repeats the previous command with the number of lines per page set to 56 and the output routed to the file named "PrtFile". AGAIN may also be used with any identifiers that are appropriate for the command being re-executed. For example, if LIST was the last command, it may be re-executed with an increased gap between the columns of variables:

```
AGAIN,  GAP 4  $
```

AGAIN will cause an error if it is used in a BATCH run.

3.6 CONTROLLING THE EDIT FILE AND ITS CONTENTS

The P-STAT editor becomes available in interactive runs — it is a log of all commands entered into P-STAT. The structure of the editor and the instructions that correct and insert commands into the edit file are described later in this manual. This section describes how to control which commands go into the edit file, how to save the edit file, and how to access prior edit files.

When an interactive run begins, the edit file is a temporary file that exists for the duration of the P-STAT run and is then deleted from the system. However, it is often useful to keep edit files for future use. It is also useful to control the contents of the edit files, particularly when an edit file is a permanent file or is to be saved as a permanent file for later use.

TEMP is a general identifier that may be used in any command to keep that command from being placed in the edit file. NEW.EDIT.FILE, OLD.EDIT.FILE and SYSTEM.EDIT.FILE are the commands that control the movement between edit files — they themselves are not placed in the edit file. HELP is never placed in the edit file, nor is any command that contains the general identifier IDEN.

ERASE.EDIT.FILE is a command that deletes all commands from the edit file. It is often used after a series of initializing commands have been executed. It may be used only in interactive runs.

Only one edit file may be active at any one time. At the beginning of a run, the active edit file is the system edit file. This is usually a temporary file that is available only during the current interactive session. Other edit files can be explicitly defined as external files, which are then available for use in subsequent sessions.

3.7 System and External Edit Files

There are three commands that control transfer between the system edit file and external edit files: NEW.EDIT.FILE, OLD.EDIT.FILE and SYSTEM.EDIT.FILE. NEW.EDIT.FILE creates an edit file in an external file:

```
NEW.EDIT.FILE  'Cars.Edt'  $
```

The name for the external file is supplied in quotes. The name is your choice, but it is often helpful to use a suffix in the name to help you remember that this file is an edit file (as opposed to a label file or a data file). This new edit file starts out empty unless the identifier ADD is included in the command. When ADD is used, the contents of the current edit file, whether it is the temporary system edit file or another permanent edit file, are placed in the new edit file.

The NEW.EDIT.FILE command causes the specified edit file to become the current edit file. All commands, subcommands and data records that are entered after the NEW.EDIT.FILE command has been executed are placed

in that external file. It continues to grow as the P-STAT run progresses, until another edit file command is executed or the P-STAT session ends.

The command OLD.EDIT.FILE causes a previously created edit file to become the current edit file. As new commands, subcommands and data records are entered, they are added to that file. The ADD identifier may also be used with the OLD.EDIT.FILE command to add all commands currently in the edit file to the file that is to become the current edit file.

SYSTEM.EDIT.FILE is the command that makes the system edit file into the current edit file. This is assumed at the start of the P-STAT interactive session. Commands that are entered after the SYSTEM.EDIT.FILE command are added to that file and are only available in the *current* P-STAT run. When the current edit file is a permanent external file because either NEW.EDIT.FILE or OLD.EDIT.FILE has been used, entering SYSTEM.EDIT.FILE causes the temporary system edit file to become the current edit file — this is illustrated in Figure 3.2.

In the second editor chapter, there is a description of the editor *instructions* BACKUP and RESTORE. They are used from within the editor to save and access edit files in permanent locations on disk. The difference between using the editor instructions and the three edit file commands is in what happens when new commands are added.

When BACKUP is used, a copy of the edit file *as it currently exists* is created in an external file. As new commands are entered, they are *not* placed in that backed-up external file — they are entered in the *current* edit file. When RESTORE is used, the edit file that is restored is exactly what was backed up earlier. The backed-up version on disk does not change even though the restored edit file continues to grow. RESTORE is normally used when the edit file contains macros or a command stream that should not be changed.

When you are using a permanent edit file as a log of a project, the NEW.EDIT.FILE and OLD.EDIT.FILE commands are the easiest way to keep that log going. When you are using a permanent edit file as a self-contained run that gets re-executed periodically but does not grow, restoring the edit file is more appropriate. The backed-up edit file, as it is saved on disk, does not get filled with any extra commands issued during the run.

The edit file commands and the BACKUP and RESTORE instructions work with *binary* edit files. The edit instructions EXPORT and IMPORT parallel the BACKUP and RESTORE instructions, but they work with *formatted* (“readable”) versions of the edit file. EXPORT saves the edit file as a formatted file on disk; IMPORT accesses a formatted disk file (or any formatted file of P-STAT commands). The difference between IMPORT and TRANSFER is that IMPORT which is an editor instruction, copies the contents of the import file into the editor but it does not execute them while TRANSFER is a command which causes the contents of the file to be executed. The contents of the TRANSFER file are placed in the current edit file as they are executed except for the special initializing commands of PSFILES, PSTEMP and PSFLAG.

3.8 Adding a Command to an Edit File

It is very common to execute a command that has a number of identifiers or data modifications, and then to execute it again with minor changes. Sometimes both forms of the command are used repeatedly on new files or subsets. If the original command has a great many data modifications, it is better to copy it and then make the minor changes, than to retype it with the possibility of making errors along the way. The editor instruction COPY may be used as described in the second of the three chapters on the P-STAT editor.

Copying can also be done by using the edit file commands with the identifier ADD. If the command to be copied is the tenth command in an external edit file named “Quest.Edt”, the procedure illustrated in Figure 3.2 may be followed. Here, the system edit file is made the current edit file with the SYSTEM.EDIT.FILE command. The EDITOR command (E) is issued and a message appears that the edit file is empty. The RESTORE instruction is used to restore all of the commands located in the external file “Quest.Edt”:

```
RESTORE  'Quest.Edt'
```

A message is issued stating that ten commands have been restored and the pointer is located at the top of the edit file. The editor instruction FIRST places the pointer at the first command in the restored edit file. DELETE 9

removes that command and the next eight commands, leaving the pointer at the tenth command in the file. This tenth command now becomes the first and only command in the current edit file. The command is revised as necessary, using the CHANGE (C) instruction.

Q is entered to leave the editor and return to the main (command) editor. The command:

```
OLD.EDIT.FILE  'Quest.Edt',  ADD  $
```

is then issued to copy this single command onto the bottom of "Quest.Edt". The file now has eleven commands. As illustrated in this example, it is possible to move between permanent edit files and the temporary system edit file.

Figure 3.2 Adding a Revised Command to an Edit File

```
>>    SYSTEM.EDIT.FILE  $
      Returning to the System Edit File.
      It has 0 commands.

      Enter a command:
>>    E

      The editor file is empty.
      At the TOP
      EDITOR:

>>    RESTORE  'Quest.Edt'

      10 commands successfully restored
      in binary form from external file Quest.Edt .

      At the TOP
      EDITOR:
>>    FIRST
      1...LINES 50 $
>>    DELETE 9
      1...LIST  Quest [ GEN Q1.A ....
      EDITOR:
>>    C/Quest/Quest2
      LIST  Quest2 [ GEN Q1.A ....
      EDITOR:
>>    Q

      Leaving the editor.

      Enter a command:
>>    OLD.EDIT.FILE  'Quest.Edt',  ADD  $
         10 Commands found on File Quest.Edt
          1 more has been added to it.
```

SUMMARY

AGAIN

```
AGAIN   $
AGAIN,  PR  'PrtFile'  $
AGAIN,  LINES 30  $
```

AGAIN may be used only in an interactive run to re-execute the previous command. It may have general identifiers such as LINES, PR, PAGE, ERROR and VERBOSITY.

ERASE.EDIT.FILE

```
ERASE.EDIT.FILE  $
```

ERASE.EDIT.FILE is an interactive command that erases the edit file.

FILES

```
FILES  $
```

FILES is used to list the currently active files.

INTERACTIVE

```
INTERACTIVE  $
```

The INTERACTIVE command specifies that the current session is an interactive or conversational one. On most systems the operating system can determine if a session is interactive. Many of the initial settings for terminals and printers are done automatically under such operating systems. The most frequent use of the INTERACTIVE command is at the end of a BATCH job to return control to P-STAT for further processing.

NEW.EDIT.FILE

```
NEW.EDIT.FILE  'Filename'   $

NEW.EDIT.FILE  'Filename',  ADD  $
```

NEW.EDIT.FILE creates a new edit file on disk. The edit file is a *binary* file that “grows” with the run, as each command is placed in it. The file is available for use in subsequent runs.

Required:

NEW.EDIT.FILE 'fn'

specifies a name for the external file.

Optional Identifiers:

ADD

adds the contents of the *current* edit file to the new edit file.

OLD.EDIT.FILE

```
OLD.EDIT.FILE  'Filename'   $

OLD.EDIT.FILE  'Filename',  ADD  $
```

OLD.EDIT.FILE makes an existing *binary* edit file on disk into the current edit file. The edit file “grows” with the run, as each command is placed in it. The file is available for use in subsequent runs.

Required:

OLD.EDIT.FILE 'fn'

specifies the name of the external file.

Optional Identifiers:

ADD

adds the contents of the current edit file to the old edit file.

PROMPT

```
prompt long $     is assumed

prompt short $    produces short prompts
```

When PROMPT SHORT is in effect, "$$" replaces "Enter a command:".

SCREEN

```
SCREEN  22  $
```

SCREEN sets the number of lines that are available on a CRT (video display) terminal to prevent output from scrolling off the screen. SCREEN 0 turns off screen holding — printed output “flashes” by unless the terminal has some sort of screen holding of its own. This command is used only in interactive runs.

STATUS

```
STATUS  $
```

STATUS is used in interactive sessions to get a report on the current settings of VERBOSITY, SCREEN, OUTPUT.WIDTH, and so on.

SYSTEM

```
SYSTEM  'ls *.PS*'  $
```

The SYSTEM command passes the argument string to the host operating system for execution. The quoted string in the command above, for example, is suitable for a UNIX system. It lists all files in the current directory that have ".PS" in their names — that is, the P-STAT system files on disk. SYSTEM works on many, but not all, computers.

On Unix you can use SYSTEM without a string to spawn another process. This cannot be done under PC/Windows

SYSTEM.EDIT.FILE

```
SYSTEM.EDIT.FILE  $
```

SYSTEM.EDIT.FILE makes the system edit file, located in temporary storage, into the current edit file. The system edit file is the normal edit file when an interactive run begins.

General Identifiers:

IDEN

```
PLOT, IDEN  $
```

IDEN is a general identifier that may be used with any command to list the valid identifiers for that command. When IDEN is used, the command is not put in the edit file.

TEMP

```
FILES, TEMP  $
```

TEMP is a general identifier that may be used interactively as an identifier in any command. TEMP prevents that command (and its data) from being copied into the editor file.

4
Batch Usage
For Continuous Execution

Batch computing permits the background execution of one or more P-STAT jobs without cluttering up the terminal with yet another large window. It is particularly useful when you are running TURF with a large number of combinations or a SURVEY with many cases and many tables to complete. On PC/Windows the progress of a batch job is reported in a small progress window. The results of a batch run in any environment are contained in text files suitable for printing or viewing with a text editor.

Batch execution of a P-STAT run requires only:

- the system command that invokes the P-STAT program, and
- the P-STAT commands that comprise the run, including any commands that enhance batch usage.

The P-STAT commands comprising the batch run are usually entered in a text file using an editor or word processing software or by saving a current P-STAT session for reuse. The saved P-STAT commands are executed by:

1. invoking P-STAT from a terminal with 2 arguments; the name of the file containing P-STAT commands and the name of a file to contain printout. Used this way you can submit several P-STAT jobs and let the operating system determine how they are executed.
2. using the TRANSFER command from within an interactive P-STAT session.

When P-STAT is used in a batch environment, there are no prompt messages and the internal P-STAT editor is not activated. Batch P-STAT reads and executes commands from the input stream until:

1. too many consecutive errors are found;
2. an end-of-file marker or the END command is encountered; or
3. the INTERACTIVE command is executed.

4.1 Starting a Batch Run

You can begin a batch run in several ways:

1. use the TRANSFER command from an interactive session;
2. using command line arguments when calling the module;
3. or having the entire run as part of your PSTART file.

TRANSFER files and the PSTART file are discussed in full in previous chapters. Under PC/Windows use:

```
PSTAT.EXE  input-file-name  output-file-name
```

The two file names follow the P-STAT command name. The first file contains the P-STAT commands to be executed and the second is an output file. This should be provided even if you have provided an output destination in your command file. P-STAT's initial printout is a "hello world" message giving the version number and date. This message is printed before any commands are executed, therefore, before a secondary output destination is known. . If you have no PR command, all the output from the run will be in the second file provided in the command line. If there is a PSTART file available, it is executed before the commands in the "input" file.

In Windows, using this mode to start the batch run starts a progress window which prints the name of each command as it is processed. This window is automatically closed when the run is finished. If you execute a batch

run from a TRANSFER file, the full P-STAT window is on the screen unless you press the minimize button to make an icon.

```
p-stat <input file >output.file &
```

is the comparable Unix command. The use of the “&” in a UNIX environment causes the job to run in the background. This frees up the terminal for other uses. NOTE: this type of redirection does not work on the PC under Windows or DOS. They require command line arguments without the "<" and ">" symbols.

4.2 The BATCH Command

The BATCH command indicates that a run is to be a BATCH job. This must be near the beginning of the run after any directories have been defined. It sets a variety of parameters to settings appropriate for batch operation. Most computers assume an interactive mode of operation while others assume a batch mode. Therefore, it is best to issue the BATCH command to ensure appropriate parameter settings and to commence batch execution.

It is possible to go back and forth between batch and interactive processing. P-STAT is able to tell from the way it is invoked whether the run is batch or interactive and sets the appropriate output parameters when the run begins.

```
p-stat  <test.trn    on Unix/Linux
P-STAT  TEST.TRN     on PC/Windows
```

With a single argument ,the contents of the file "TEST.TRN" are executed. However, what happens next depends on whether the file TEST.TRN has any instructions on the mode of operation. If nothing is specified, the run is considered an interactive run and control returns to the terminal. If BATCH $ is specified, the commands are executed and the run ends unless a final "INTERACTIVE" sets the mode back. Providing just the input argument can result in a successful run where you never see the results.

BATCH causes the output width for the primary printer to be set to 132 characters. VERBOSITY is set to level 3 and ERROR is set to LONG. These settings may be changed when they are not appropriate for a particular situation. INTERACTIVE causes the output width for the primary printer to be set to 80 characters/ VERBOSITY is set to level 2 and ERROR is set to SHORT.

4.3 Differences Between Batch and Interactive Computing

There are some distinct differences between batch and interactive runs. The first difference is in the user’s presence. In interactive processing, the user “interacts” with P-STAT by issuing commands and any changes necessary to successfully execute a series of commands. However, in batch processing the final form for the entire series of commands must be entered before execution begins.

A second difference between interactive and batch processing is in the destination of printed output. In an interactive session, output is usually directed to a terminal. In a batch run, all output is normally sent to a disk file or directly to a printer. However, in either mode, output may be explicitly directed to the terminal, the printer or a disk file. Since batch jobs are normally sent to a disk file or printer, the print parameters assumed for batch jobs are a width of 132, a vertical bar character and underlining capabilities. (See the first of the Utility Commands chapters for a detailed explanation on defining parameters for output print destinations.)

There is a difference between interactive and batch modes in the handling of error situations. As stated earlier, in interactive execution the user is able to decide at any time on the particular processing steps and may make any necessary changes to the command using the P-STAT editor. In batch processing, the editor is not available. Once the run commences, changes may not be made and errors may not be corrected. When errors are found, the run continues until the maximum number of consecutive errors defined at the beginning of the run or the default of *five consecutive* errors has been reached. If a *fatal* error is encountered in processing, that error causes the entire run to terminate. A fatal error is one that automatically prevents any subsequent commands from being successfully processed once this error has occurred.

Another difference between batch and interactive processing occurs in commands such as ANOVA and SURVEY, which permit the interactive user to examine the output and then decide what to do next within the same

command. In a batch run, all choices must be made before the run is submitted for execution. Therefore, there are some features in P-STAT that are available only in interactive mode.

Figure 4.1 Fatal Errors in a Batch Run

External File PSEG.Trn (P-STAT Commands):

```
BATCH  $
MAXERROR 1  $
PR  'PSEG.Prt'   $

SHOW  PSEG  $
MODIFY  PSEG  [ KEEP  Month  Gas?  Elec? ,
  OUT  PSEGMod  $

SHOW  PSEGMod  $
END $
```

Within P-STAT:

```
     TRANSFER  'PSEG.Trn'  $
```

External File PSEG.Prt (Output Destination):

```
Fri Apr  3  1998: positions, types and names in file PSEG

  1 D   Year                 6 D   Elec.Cost
  2 D   Month                7 D   Averaged
  3 D   Gas.CCF
  4 D   Gas.Cost
  5 D   Elec.KWHR

     EEEEEEEEEEEEEEEEEEEE
     E
     E   A dollar has been found ending a record:
     E
     E      MODIFY PSEG [ KEEP Month Gas? Elec? , OUT PSEGMod $
     E
     E   The brackets, however, are not balanced.
     E
     E   The earliest left bracket still open began here:
     E
     E      [ KEEP Month Gas? Elec? , OUT PSEGMod $
     E
     EEEEEEEEEEEEEEEEEEEE

Too many errors in a row.

P-STAT ending... 16:48:26  Fri Apr  3  1998

number of errors during this run was     1
```

4.4 Handling Errors in Batch Mode

The MAXERROR command sets a limit to the number of consecutive errors allowed in a batch run before processing should be terminated. MAXERROR is automatically set to 5 when a batch run begins, but it may be reset. If more than this number of errors occur in a row, the run ends.

Figure 4.1 also illustrates this. In this example, MAXERROR is set to one, indicating that the run should terminate when one fatal error is encountered. Commands in the external file named “PSEG.Trn” are accessed using the TRANSFER command from within P-STAT:

```
TRANSFER  'PSEG.Trn'  $
```

The PSEG file is successfully located, and its variable names and data types are listed in the specified output destination. However, an attempt to modify the file with a variable selection clause:

```
MODIFY  PSEG  [ KEEP  Month  Gas?  Elec? ,
  OUT  PSEGMod  $
```

does not take place. The brackets are not balanced and P-STAT issues an error message to that effect:

```
 EEEEEEEEEEEEEEEEEEEE
E
E   A dollar has been found ending a record:
E
E      MODIFY PSEG [ KEEP Month Gas? Elec? , OUT PSEGMod $
E
E   The brackets, however, are not balanced.
E
E   The earliest left bracket still open began here:
E
E      [ KEEP Month Gas? Elec? , OUT PSEGMod $
E
EEEEEEEEEEEEEEEEEEEE
```

Since MAXERROR is set to 1 at the beginning of the run, the job terminates because of this fatal error. A MAXERROR setting of 1 is the number of errors set by most batch users because it does not allow the run to continue if an error is made. Setting MAXERROR to a large number causes the run to continue processing commands until either the END command or a fatal error is encountered:

```
MAXERROR 9999  $
```

Setting MAXERROR to 1 causes the run to end after the first error is found:

```
MAXERROR 1     $
```

Reading through large files takes both time and computer resources. Therefore, repeated attempts to recover after an unexpected error can be expensive. As a result, MAXERROR 1 is often the appropriate setting when large files are processed. The best way to ensure that a large run completes in a batch environment is to run it first on a small sample. If the test run is done interactively, all that is needed is a case selection. If the test run is done as a batch job, a large MAXERROR setting is appropriate. When the test run is error free, removing the case selection and setting MAXERROR to 1 readies the job for submission for final execution.

It is possible to continue processing in a batch run when there are more errors than defined with the MAXERROR command as long as the errors do not occur consecutively. Each time that a command executes successfully, the count of consecutive errors is reset to zero.

4.5 Printing the Input Stream

The command PRINT.INPUT causes the entire input stream to be read and printed *before* execution occurs. This command may be used only in batch runs and must be placed at the *beginning* of the run.

The PRINT.INPUT command is not available in interactive sessions. The interactive user can, however, get a record of his or her session by using the EXPORT instruction to write the contents of the edit file (the commands) on the printer or in a disk file. Alternatively, ECHO may be used as a command or as a general identifier in specific commands to echo the command above the output that it produces.

4.6 HELP IN BATCH RUNS

The on-line help file is usually considered an interactive feature. However, it is available in batch runs just as it is in interactive runs. In an interactive run, the usual use of the help file is to ask for help on a single topic or command.

```
HELP LIST $
```

In a batch run the output from the HELP command is sent to the current print destination. A very good way to use the help file if you use P-STAT only as a batch process to save it as a file which you can access from whatever text editor you use to construct your batch runs.

```
PR 'Help.txt' $
HELP EVERYTHING $
```

SUMMARY

BATCH

```
BATCH  $
```

The BATCH command signals batch execution. BATCH causes output width for the primary output file to be set to 132, VERBOSITY to level 3 and ERROR to LONG. The editor is *not* available in batch execution. Some interactive features such as full screen DATA.ENTRY, are not available in batch mode. In addition, some utility commands are useful only in interactive sessions (see the second Utility Commands chapter).

MAXERROR

```
MAXERROR 1   $
```

MAXERROR sets the maximum number of *consecutive* errors permitted before the run terminates. MAXERROR is set to 5 automatically by the BATCH command.

PRINT.INPUT

```
PRINT.INPUT  $
```

PRINT.INPUT prints the entire contents of the input stream at the beginning of the output before executing any of the commands. This may be used only in batch runs and must be placed at the *beginning* of the command stream.

HELP

```
PR 'Help.txt' $
HELP EVERYTHING $
```

These two command store the entire P-STAT help file in an external disk file which can be accessed and searched by your operating system editor.

5
Interactive Usage and the Editor

This chapter is a general introduction to interactive computing and the essentials of the P-STAT editor. The use of INTERACTIVE and the utility commands that are useful in both batch and interactive environments, are covered in the chapter "Utility Commands for General Use". A full discussion of editor instructions can be found in the following chapters. The commands for saving and restoring editor files and for defining the interactive environment are discussed in the chapter "Utility Commands for Interactive Use".

There are some special commands available when P-STAT is used interactively. These commands can be divided into three categories:

1. Commands to define the terminal or to set general run-time parameters;
2. Editor instructions to correct errors and build a run;
3. Commands to save and restore editor files.

5.1 INTERACTIVE MODE

In most environments, P-STAT is able to distinguish whether a run is interactive or batch. When this is the case, it is unnecessary to issue either the INTERACTIVE or the BATCH command unless the user wishes to change the mode of operation. If, at the beginning of an interactive P-STAT session, there "Enter a command:" prompt does not appear, the particular computing system is *not* automatically differentiating between a batch and an interactive user. In such a case, issuing the INTERACTIVE command causes prompting to begin.

INTERACT is an abbreviation for INTERACTIVE. The following settings are in effect when INTERACTIVE is executed:

1. The primary input device becomes the terminal.
2. The primary output device becomes the terminal.
3. The P-STAT editor is made available.
4. Short error messages are given.
5. Verbosity is set to 2.

The error setting, the verbosity setting and the attributes for the terminal may each be changed. See the chapter "Utility Commands for General Use" for details.

5.2 Listing Identifiers for a Command

The HELP command and the general identifier IDEN, both of which may also be used in batch mode, are particularly useful in interactive computing. HELP by itself explains how to use the on-line HELP file. HELP, followed by a command name, provides information about a specific command. IDEN, after a command name, provides a list of the identifiers for that command. (The command name must be followed by a comma.) In Figure 5.1, a list of all the identifiers supported for the LIST command prints on the terminal in an interactive run or on the printer in a batch situation.

Figure 5.1 Determining the Identifiers for a Command

```
LIST, IDEN  $

These identifiers can be used.....


by                  expand            lows              percentages
by.n                fill              margin            print.positions
case.percentages    flag              max.passes        re.identify
cases               fold              max.places        skip.counter
cases.per.page      fold.stub         max.vl            skip.omit
center              gap               means             skip.var
commas              highs             min.places        stub
dashes              identify          n                 totals
data.only           labels            newpage           trim.zeros
dollar              left              no.cases          use.xl
double              list              ow
```

P-STAT commands such as ANOVA, DISCRIM, EDA and SURVEY, which have extensive subcommand structures, also have extensive help messages. When using one of these commands, the single letter “H” is all that is necessary to gain access to appropriate help text.

Figure 5.2 Long and Short Prompt Messages

```
The most common prompt messages are:

ORIGIN              LONG PROMPTS                         SHORT PROMPTS

P-STAT              Enter a command:                     $$:
Commands            Continue the command, or reply       more $$:
                    H/E/Q:


Subcommands         Begin case 1:                        1:
                    Begin subcommand, or type Q or H:    Enter subcommand:

Editor              EDITOR:                              EDIT.C:
                    EDITING DATA RECORDS:                EDIT.D:
```

5.3 Prompt Messages

In interactive mode, control returns to the user at his terminal when:

1. P-STAT needs more command information;
2. a command needs more data; or

3. an error occurs.

When control returns to the terminal, a prompt message is issued for the next command, subcommand or data record. The prompts differ depending on whether long or short prompts have been requested. At the start of an interactive run PROMPT LONG is set. This can be changed by issuing:

```
PROMPT SHORT  $
```

Figure 5.2 illustrates the different messages given with long and short prompts and shows their origins.

Whenever any prompt occurs, the reply “Q” (for QUIT) causes P-STAT to return to the “Enter a command:” level. When the prompt is for a command or for command continuation, the reply “H” (for HELP) prints a short description of what is expected next at the terminal. It is also possible to reply with the letter “E” (for EDITOR), which places the user directly in P-STAT’s internal editor. If “E” is entered when a command is partially complete, that command is not discarded. It is written in the editor and the user may complete the command, make any desired modifications and resume execution.

The prompts for data and subcommand records are more varied because they are determined by the command that needs the data or subcommands. For example, the prompt for the EDA command is:

```
Exploratory Data Analysis
Enter    HELP;  For Details on Usage
```

If an error is found in an interactive run, the error message is printed at the terminal. If command text is being entered, the P-STAT editor assumes control. Editor instructions may then be used to correct any errors. As a result, execution may be resumed without retyping the entire command or data record. If subcommand information is being entered, those incorrect subcommands are placed in the editor. They must be corrected or deleted before execution can resume.

5.4 THE P-STAT EDITOR

The P-STAT editor should not be confused with an operating system editor. It is located entirely within P-STAT and is available during the course of a P-STAT run. When an error occurs, it may be corrected on-the-spot without losing any terminal settings, calculations or intermediate files.

The P-STAT editor operates on an *edit file* of P-STAT commands, subcommands and data records. The editor automatically differentiates between command text and any subcommands and data records that the command requires. Subcommands are defined as any additional information required by a command, aside from the user’s raw input data. As far as the editor is concerned, however, subcommands and data records are identical.

The editor can be used to enter, change or delete commands, subcommands and data records. It also has the ability to check for correct command syntax. Entire runs (streams of P-STAT commands) may be built inside the editor and checked for syntax before execution begins. Other errors, such as the misspelling of a variable name, can only be detected when the command is executed. When such an error occurs, the editor is automatically positioned at the command or data record that caused the error. At this point, the incorrect command or data record may be corrected and execution may be resumed.

5.5 The Edit File

The edit file is empty at the beginning of an interactive P-STAT run. It is a temporary file that exists only during the current P-STAT session unless it is explicitly saved or written to an external file. Saving the edit file is done with the editor *instructions* BACKUP and RESTORE or EXPORT and IMPORT, which are described in the next Editor chapter. Using a permanent external file in place of the temporary system edit file is done with the *commands* NEW.EDIT.FILE and OLD.EDIT.FILE, described in the chapter “Utility Commands for General Use.”

The edit file acquires commands and their data in three ways. First, commands and data entered from the terminal (or from a transfer file) during interactive execution are automatically added to the edit file. Second, the editor has an input mode allowing commands and/or data to be added or inserted directly, without execution.

Third, an edit file can be backed up in one run and then restored as the edit file in later runs (allowing the development of edit file libraries).

5.6 Execution from the Edit File

A command in the edit file can be executed from *within the editor* by entering the EXECUTE (or X) *instruction*:

```
          EDITOR:
             3...LIST Myfile $
      >>  X
```

or from *outside the editor* (in P-STAT command mode) by entering the AGAIN *command*:

```
          Enter a command:
      >>  AGAIN $
```

EXECUTE (X) causes one or more commands to be executed from within the edit file. AGAIN causes only the most recently executed command to be re-executed. The AGAIN command is only available in an interactive session.

Both EXECUTE and AGAIN process all subcommand and data records associated with a given command. If the re-executed command needs additional subcommands or data, the user is prompted for them. These new records are placed in the edit file in the order in which they are entered. Thus, the edit file grows as new information is supplied.

5.7 BASIC STRATEGIES FOR EDITOR USE

The editor provides a powerful tool for command and data input, syntax checking and error recovery. It has its own structure and keyword instructions. However, it is only necessary to know a few of these instructions to perform the basic editor operations. The four fundamental instructions are:

1. QUIT (Q)
2. CHANGE (C)
3. DELETE (DEL)
4. EXECUTE (X)

On systems which support full screen operations the instructions, “Z” can be used to permit full screen editing of a command and its subcommand records. In addition, on the PC running Windows or NT the edit instructions and the full screen editor are available from the top line pull-down menus. Details for use of these menus are provided in the chapter “Full Screen Editing”.

5.8 Correcting a Command Error Using QUIT

“Q” for QUIT is the most useful interactive instruction in P-STAT. It provides a way to get back to command entry if an error has been made. This is true whether the user is in the middle of a command, executing a macro or inside the editor. QUIT works whether the error is in a command or a data record. For example, if this line is entered:

```
MAKE  MyFile,  VAARS  Name:C32 Age Sex Weight ;
```

the following error message prints at the terminal along with a list of identifiers available when using the BUILD command:

```
Unknown identifier vaars used in the MAKE command.

The MAKE command has these identifiers.....

blank.delimiter   honor.strings     nv
change            ignore.strings    one.pass
```

```
definitions        left.justify       options

( followed by the rest of the MAKE identifiers )
```

Control then returns to the P-STAT editor:

```
     EDITOR entered.
       1.. MAKE  MyFile,
           VAARS  Name:C Age Sex  Weight ;
     EDITOR:
```

The editor is waiting for an instruction. If you do not wish to use the editor to correct the command, simply reply "Q":

```
>>   Q

     Leaving the editor.
     Enter a command:
```

You are now back in normal command entry mode and may reenter the correct command:

```
>>   MAKE  File1, VARS  Name:C Age Sex  Weight ;
     begin case 1, Name (c40):
```

When the command is entered correctly, P-STAT prompts the user for the first case.

5.9 Deleting a Command Error

To retain just correct commands in the editor, the DELETE instruction could have been used in the preceding example in addition to "Q":

```
     EDITOR entered.
        1...MAKE  MyFile,
            VAARS  Name:C32  Age  Sex  Weight ;
     EDITOR:
>>   DELETE

     At the TOP
     EDITOR:
>>   Q
     Leaving the editor.

     Enter a command:
>>   MAKE  MyFile,  VARS  Name:C32  Age  Sex  Weight ;

     begin case 1, Name (c32):
```

Once the incorrect command is deleted, the user quits the editor. The correct command is entered and P-STAT prompts the user for the first case. DELETE, like all editor instructions can be abbreviated. Because it is a destructive instruction, the abbreviation requires the 3 characters "DEL" rather than a single character.

5.10 Correcting an Error in the Command Text

While QUIT or DELETE may seem the easy way out at first when errors in typing occur, the entire command must be reentered. In cases when command text is long, using the full screen edit instruction (Z) or the editor instruction CHANGE eliminates the need to retype the entire command. The CHANGE instruction is used to change selected portions of the command text:

```
        EDITOR entered.
           1...MAKE  MyFile,
               VAARS  Name:C32  Age  Sex  Weight ;
        EDITOR:
>>      CHANGE /VAARS/VARS/
           1...MAKE  MyFile,
                VARS   Name:C32  Age  Sex  Weight ;
        EDITOR:
>>      EXECUTE
        begin case 1, Name (C):
```

CHANGE requires two arguments: 1) the incorrect character string, and 2) the correct character string. Each string must be delimited by a special character such as a slash (/) or colon (:), selected so that the character chosen does not appear in either string:

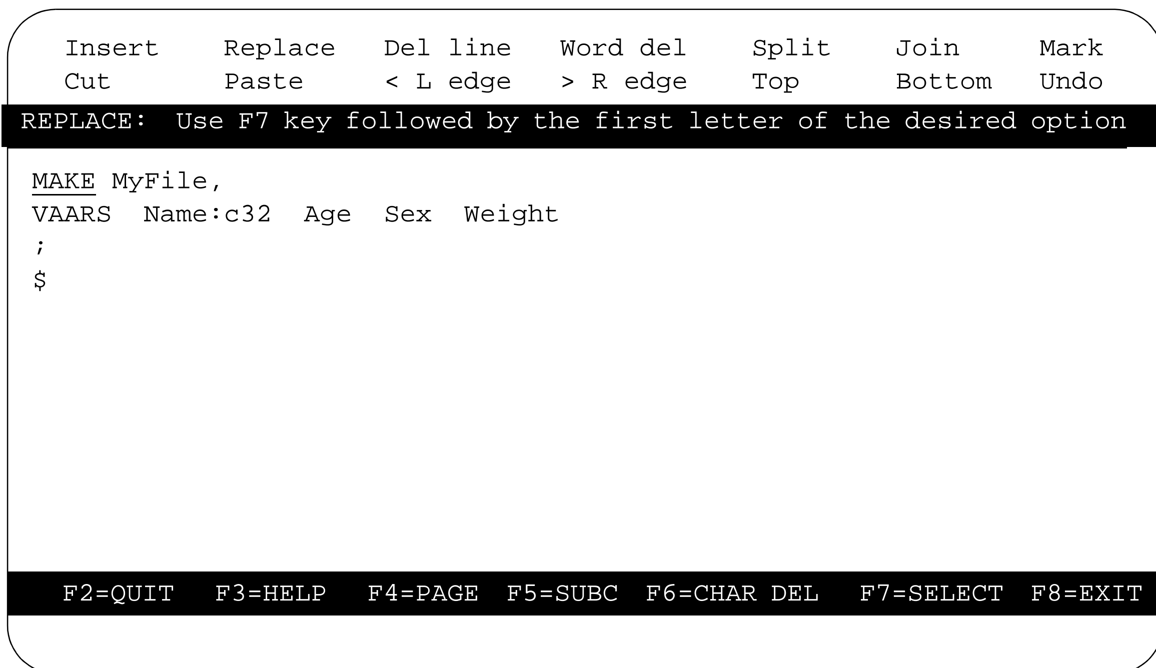
```
>>  CHANGE :VAARS:VARS
```

Notice that the final delimiter (the colon or slash) is not necessary. CHANGE is typically abbreviated to “C” and EXECUTE to “X”, which reduces the chance of typing errors within the editor.

```
>>  C :VAARS:VARS
>>  X
```

If you are running P-STAT on a PC, a VAX, or any computer running Unix, the full screen editor is available by using the single letter “Z”. Figure 5.3 illustrates the screen that appears in Unix/Linux in Unix/Linux when the Z instruction is entered.

Figure 5.3 Full Screen Editor



In the full screen editor you can use the arrow keys or (PC/Windows only) a mouse to move the cursor to the desired location. The cursor is initially located at the first character, the “M” of MAKE in Figure 5.3. Pressing the down arrow key followed by the right arrow key places the cursor at the extra letter “A”. The F6 key is used to delete the extra letter and the F8 key is used to access the menu of Exit options.

5.11 Using Mixed Case Within the Editor

On computer systems that permit mixed uppercase and lowercase data entry, the use of the uppercase and lowercase letters in the editor is exactly the same as it is when entering commands or data. This means that all comparisons are case independent and all input is retained exactly as it is entered:

```
   1...BUILD    MyFile,
          VAARS  Name:C  AGE  Sex

CHANGE /VAARS/vars
   1...BUILD    MyFile,
          vars   Name:C  AGE  Sex
```

The first or comparison string can be entered in either uppercase or lowercase. “VAARS” and “vaars” both produce a match against either “vaars” or “VAARS”. The second or input string in the example above is in lowercase and therefore becomes lowercase in the command. The case of command names and identifiers makes no difference to P-STAT.

The case in which variable names, file names and headings are entered or changed makes a difference because they are used exactly as they are entered in any subsequent printout. The variable Age will now be printed in mixed case letters on listings, tables and reports:

```
C /AGE/Age/
   1...BUILD  MyFile,
          VARS  Name:C  Age  Sex
```

In the full screen editor the CHANGE instruction is unnecessary. Once the cursor is located at the start of the string to be changed, simply overtype with the correction. If the string that is being corrected is in the middle of the line and the new string is longer you can change from REPLACE MODE (the default) to INSERT mode. This change is done by pressing the F7 key followed by the letter “I” or by moving the mouse, if it is supported, and clicking on “Insert” in the top line.

SUMMARY

INTERACTIVE

INTERACTIVE $

When INTERACTIVE or the abbreviation INTERACT is executed, a P-STAT run becomes interactive:

1. The primary input device becomes the terminal.
2. The primary output device becomes the terminal.
3. Error is set to SHORT.
4. Verbosity is set to 2.
5. The P-STAT editor is made available.

The basic editor instructions are:

1. Q QUIT the editor;
2. C CHANGE /bad string/good string/;
3. DEL DELETE a record;
4. X EXECUTE the command;
5. W WHERE to find out where you are;
6. Z On Unix of PC/Windows for full screen edit of a command.

HELP

```
HELP LIST $
```

1. HELP $ gives help on using the P-STAT HELP file.
2. HELP COMMANDS $ provides a list of commands supported by HELP.
3. HELP MENU $ provides a list of other topics.
4. HELP EVERYTHING $ lists the entire HELP file.
5. HELP NEWS $ describes the latest features in the current P-STAT version.

The HELP command may be used in batch or interactive modes. It provides help on the most frequently used commands and also on general topics such as LEGAL.NAMES or STARTING.A.RUN. HELP, with no arguments, explains how to use the HELP file.

New features in the system that may not be documented in the P-STAT users manuals are documented in the HELP file. HELP NEWS provides a list of the new features in the system.

PROMPT

```
PROMPT  SHORT  $
PROMPT  LONG   $
```

The PROMPT command sets the length of prompt messages.

Arguments:

SHORT

requests that prompt messages be short.

LONG

requests that prompt messages be long. LONG is assumed unless SHORT is explicitly set.

General Identifiers:

IDEN

```
CORRELATE,  IDEN  $
```

The general identifier IDEN, like HELP, may be used in batch mode but is especially useful in an interactive session. It can follow any command name and produces a list of the identifiers recognized by that command.

6
The Editor in Detail

The first editor chapter describes the basics of interactive computing in P-STAT, including the INTERACTIVE command and the use of the editor for simple error correction. This chapter, the second of the editor chapters, provides a detailed description of the editor as it is used for error correction and syntax checking. The third editor chapter covers the use of the editor for building runs by inputting commands.

6.1 EDIT MODE FOR CORRECTION

The P-STAT editor has two modes: edit mode and input mode. In *edit* mode, you can delete, change, check the syntax of, and execute commands and their data or subcommands. In *input* mode, you can input commands and data or subcommands, without executing them. These two modes are similar to those in many word processing packages — you can edit text in one mode and you can input text in the other mode. In the P-STAT editor, you edit commands in edit mode and you input commands in input mode.

Edit mode is the most important and the most commonly used mode of the editor because it permits recovery from errors. Commands are entered into P-STAT at *command* level. This is outside of the editor — it is in response to the prompt that P-STAT uses:

```
Enter a command:
```

Commands are executed *and* simultaneously written in the edit file (the editor's file). Thus, the P-STAT editor differs from a word processor in that it gets input without your invoking input mode. All commands are "logged" into the editor. When a mistake occurs, you are placed in the editor in edit mode so that you can correct the error.

This chapter discusses the edit mode of the P-STAT editor. Input mode is discussed in the next chapter. This section discusses:

1. the structure of the edit file,
2. how to enter the edit file,
3. how to move about within the editor, and
4. how to correct errors.

6.2 Structure of the Edit File

The edit file has a sequential structure. The first command that is entered is followed by the second command, and so on. Each command may or may not have subcommand or data records associated with it. In this example, the edit file has three commands:

```
1...MAKE  MyFile,  VARS  Age  Sex ;
  1=21 1
  2=23 2
  3=$
2...LIST  $
3...SURVEY;
  1=BANNER  Age,  STUB  Sex  ;
  2=$
```

The numbers are not part of the commands, but are supplied by the editor. MAKE has three data records. SURVEY has two subcommands. They are also numbered by the editor. (Data and subcommand records are identical as far as the editor is concerned. Both are considered to be extra information required and processed by a given command.).

There is a pointer in the editor that points to one command at a time. That command is the target of any editor instructions until the pointer is moved. If the command pointer is at LIST, MAKE cannot be changed and MAKE's data records cannot be accessed. The data and subcommand records "owned" by a given command are available only when the command pointer is located at that command.

There are editor instructions to control the movement between the commands and their data records. There are editor instructions to control the movement of the pointer from one command to another command or from one data record to another data record. In addition, there are editor instructions to modify the command or data records in some way.

6.3 Entering the Editor

The P-STAT editor is really three different editors. The *main editor* is concerned only with commands. The *data editor* is concerned with either subcommand or data records. The data editor is available for a given command only when the command pointer is positioned at that command.

To enter the main or command editor, type the EDITOR command or "E" when a command is expected:

```
      Enter a command:
 >>   E

       EDITOR entered.
       The editor file has 1 command .
          1...MAKE  MyFile,
                 VARS  Name:C32 Age Sex Weight ;
       EDITOR:
```

When the editor is entered, the pointer is at the last command. It is in command edit mode. You can control *where* to enter the editor by following "E" with an EDITOR instruction. This instruction places the pointer in the editor at the top:

```
      Enter a command:
 >>   E TOP

      At the TOP
      EDITOR:
```

This instruction places the pointer in the editor located at the character string "MAKE":

```
      Enter a command:
 >>   E LOCATE /MAKE
          1...MAKE  MyFile,
                 VARS  Name:C32 Age Sex Weight ;
      EDITOR:
```

When the editor is entered automatically because of an error, entry is either in the command editor or in the data editor, depending on whether the last record (line) read was a command or a subcommand / data record. NOTE: most of the editor instructions can be abbreviated to 1 or 2 characters. Thus L by itself will be recognized as LOCATE.

6.4 Moving Between the Command and Data Editor

The *data* editor is entered from the command editor by using the editor instruction DATA or the abbreviation “D”. Only the subcommand and data records for the *current* command are available when the data editor is entered:

```
>>   DATA
     This command has 8 data or subcommand records.
     The pointer is at the last one.
          8=$
     EDITING DATA RECORDS:
```

All the data records for the current command are now available. The pointer can be moved up and down these data records until control is returned to the command editor. There are several ways to return to the command editor. “E” or EDITOR always moves control to the *command* editor, whether you are in the editor or outside the editor. COMMAND or “C” gives control to the *command* editor when you are inside the editor. RETURN or “R” gives control to the *previous higher level of the editor*, which is usually, but not always, the command editor:

```
     EDITING DATA RECORDS:
>>   RETURN

     Returning to the main ( command ) editor.
     EDITOR:
```

Editor instructions are *identical* in both the command and data editors except for:

1. DATA or D which is used in the command editor to enter the data editor,
2. COMMAND or C, which is used in the data editor to return to the command editor,
3. instructions to backup and restore or export and import the edit file, which only work in the command editor, and
4. instructions to get and put data, which only work in the data editor.

In Figure 6.1, when an error is made in the MAKE command, the editor is automatically entered and the first line (or a portion of the line) of the command that caused the error is printed. This is followed by the prompt “EDITOR:” to indicate that the editor is the command editor and that it is waiting for command instructions. Possible replies to the “EDITOR:” prompt are:

1. QUIT to exit the editor,
2. HELP to get assistance,
3. TYPE to type the entire command,
4. DATA to enter data editor, and
5. CHANGE, DELETE, and other editor instructions.

Figure 6.1 Using the Editor To Correct a Command Error

```
        Enter a command:
 1. >>  MAKE  MyFile,  VAARS  First.Name:C16  Last.Name:C16  Age  Sex
                                Weight ;

UNKNOWN IDENTIFIER---           VAARS
```

```
The MAKE command has these identifiers.....
blank.delimiter    honor.strings      nv
change             ignore.strings     one.pass
definitions        left.justify       options
delimiter          make               string.boundary
drop               missing            strip
drop.lead.blanks   no.del             summary
eoc                no.eoc             template
file               no.missing         vars
filesize           no.stripe

          EDITOR entered.
             1...MAKE  MyFile, .....
  2. >>   CHANGE /VAARS/VARS/                 ( C /bad/good/ )
             1...MAKE  MyFile,
                   VARS  First.Name:C16  Last.Name:C16  Age  Sex  Weight ;
          EDITOR:
  3. >>   EXECUTE                             ( X is short for eXecute )
          begin case 1, First.Name (C):
  4. >>   Mary Smith 28 2 125
          begin case 2:
  5. >>   Don Smith 19 1 160
          begin case 3:
  6. >>   Sally Stang 26 2 140
          begin case 4:
  7. >>   Nancy Howe 24 1 130
          begin case 5:
  8. >>   Ed Elliot 21 1 165
           begin case 6:
  9. >>   Allison Wing 23 2 118
          begin case 7:
 10. >>   Ed Elliot 21 1 165
          begin case 8:
 11. >>   $
          Input terminated by a single $.

          7 cases and 5 variables processed.


          Enter a command:
 12. >>   LIST  MyFile  $

          First     Last
          Name      Name        Age   Sex  Weight

          Mary      Smith        28     2     125
          Don       Smith        19     1     160
          Sally     Stang        26     2     140
          Nancy     Howe         24     1     130
          Ed        Elliot       21     1     165
          Allison   Wing         23     2     118
          Ed        Elliot       21     1     165
```

When the error is in a data or subcommand record, control is given to the data editor. The prompt is "EDITING DATA RECORDS:". Possible replies to this prompt are the same as the replies to the command prompt except that DATA is not used — the COMMAND (C) instruction returns to the command editor.

6.5 Fixing an Error in a Command

In Figure 6.1, the problem in the MAKE command is a misspelling of the identifier VARS. The editor instruction CHANGE (or its abbreviation "C") can be used to correct this error. **C**HANGE is followed by two character strings. If the characters in the first string are found in the command, they are changed to the characters in the second string. If the string to be changed is typed incorrectly, the editor is not able to find it. A message to that effect is printed on the terminal:

```
CHANGE /VAARS/VARS/
THESE CHARACTERS WERE NOT FOUND.
VAARS
EDITOR
```

In Figure 6.1, the first string, "VAARS", that is misspelled is changed to the correct spelling, "VARS". (The summary at the end of this chapter contains optional shorter forms of the editor instructions. For example, C/AA/A is sufficient to change the first occurrence of "AA" to "A".) The numbers on the left in the figure are *not* part of the commands, but merely reference numbers.)

The character that *delimits* the strings may be any character other than a blank, a letter or a number that is not found in the string:

```
CHANGE :VAARS:VARS:
```

A slash or a colon is frequently used. The choice does not matter as long as the delimiting character itself does not appear within the strings. In Figure 6.1, this could be used to *delete* the variable name "Sex":

```
>>   C / Sex//
       1...MAKE  MyFile,
              VARS  First.Name:C16 Last.Name:C16 Age Weight ;
```

If the two delimiters are used together as the first character string in the change, the *null* or empty string points to the *beginning* of the record. It is possible to change a null string at the beginning of a command to a second string. This inserts the characters "SORT" at the beginning of the command:

```
CHANGE //SORT /
```

Sometimes the same error occurs *more than once* in the command. For example, in a command, a variable name misspelled once is often misspelled the same way a second time. CHANGE may be used to change *all* occurrences of a character string within a command or data record. This sequence changes every occurrence of the character string "x" into the character string "xx" *across* the entire record:

```
CHANGE /x/xx/ A
```

Sometimes a universal change may be more of a hindrance than a help:

```
1...MAKE  MyFile,
       VARS  First.Nama:C16 Last.Nama:C16 Age Sex Weight ;

CHANGE /A/E/ A 5 CHANGES MADE.
1...MAKE  MyFile,
       VERS  First.NEmE:C16 LEst.NEmE:C16 Ege Sex Weight ;
EDITOR:
```

Fortunately, there is an OLD instruction that recovers the command as it existed before the change was made:

```
OLD
Restoring the old form of the command and its data.
```

```
1...MAKE  MyFile, .....
EDITOR:
```

A more specific change may now be used to correct the misspellings.

CHANGE (**C**) is the most fundamental of the editor instructions. It works in both the command and the data editors. There are two similar instructions named BEGINNING (**B**) and END (**E**). This,

```
BEGINNING /Social.Security/
```

inserts the string "Social.Security" at the beginning of the command or data. It achieves the same result as the CHANGE instruction:

```
CHANGE //Social.Security/
```

Even more useful is the END instruction. Its format is the same as the BEGINNING instruction. This,

```
END /Profession/
```

inserts the string "Profession" at the end of the command. In the *command* editor, the string is inserted immediately before the final $ or semicolon. In the *data* editor, the string is inserted after the last non-blank in the data record. The CHANGE instruction achieves the same result:

```
C /;/ Profession ;
```

END could be used to add a variable to the list of variables in the MAKE command:

```
>>    E / Income/
       1...MAKE  MyFile,
           VARS  First.Name:C16 Last.Name:C16 Age Sex Weight Income ;
```

The new variable name is placed just before the semicolon. Notice that the string in the prior example begins with a blank. Without the blank, the variable names Weight and Income run together and look like a single name:

```
>>    E /Income/
       1...MAKE  MyFile,
            VARS  First.Name:C16 Last.Name:C16 Age Sex WeightIncome ;
```

6.6 Executing a Corrected Command

As discussed in the first editor chapter, whenever changes such as the preceding ones are made in the editor to either commands or data records, the changed commands must be executed. If a file has been successfully built and then the MAKE command is changed in the editor, the command must be re-executed in order to rebuild the file with the changes. (See Figure 6.2.) If the command is not re-executed, the only thing changed is the *text* of the command in the edit file. EXECUTE (**X**) may be entered from either the command editor or the data editor.

EXECUTE (or **X**) processes the command currently pointed to and returns to command entry mode. If **X** is followed by an integer (**X** 3), the specified *number* of commands are processed. If **X** is followed by an asterisk (**X ***), *all* commands from the current position of the pointer through the end of the edit file are executed. If **E** (for ECHO) follows (**X 2 E**), the specified number of commands are printed or echoed on the terminal before they are executed.

It is possible to execute commands *not* located in the editor from the editor. The abbreviation for EXECUTE, the X, is followed by the desired command, which is typically a request for some information about the identifiers for a P-STAT command or about the variables in a file. For example, when X and the SHOW command is issued from anywhere in the editor, the list of the names of the variables in the most recently referenced file is displayed:

```
      EDITOR:
>>    X  SHOW  $
```

```
        Leaving the editor.

  Thu Apr 30  2005: positions, types and names in file Tests

           1  D    Age
           2  D    Sex
           3  D    Race
           4  D    Grade.Average

        Returning into the editor.
           3...LIST  Tests  (DROP  GradeAverage),
                  GAP 4 $
        EDITOR:
```

Once the SHOW command completes, control returns to the editor with the pointer located where it was before the SHOW command was processed. Now that the names of the variables are displayed, an incorrect name (GradeAverage) may be corrected, and the command may be re-executed (**X**). The SHOW command itself is *not* entered into the edit file. This type of execution may be used with any brief P-STAT command — it must fit on one line so the editor reads the concluding dollar sign and recognizes it as a command and not an editor instruction.

6.7 Single-Letter Instructions

Many of the editor instructions may be abbreviated to a single letter. A few of them are special because they work everywhere, even in input mode, where the longer forms do *not* work — they are interpreted as characters to be entered into the command or data record. The single-letter instructions are:

1. **E** enter the main **E**ditor;
2. **H** get **H**elp;
3. **Q** **Q**uit the editor;
4. **R** **R**eturn to previous level of editor;
5. **W** find out **W**here you are;
6. **X** begin command e**X**ecution;
7. **B** generate a line of **B**lanks (in data input mode only).

B is provided because of the confusion on different operating systems between a line of blanks or spaces and a null line. A null line has nothing in it, not even a blank.

6.8 MOVING ABOUT WITHIN THE EDIT FILE

When the P-STAT editor is used only for correction of errors, it may not be necessary to move about in the editor. When the editor is used to examine to contents of the edit file, to build a run, to change an error in data values or to correct the logic of the run, it is almost always necessary to move about.. The pointer in the edit file is moved both by specific instructions whose functions are just that — moving the pointer, and by other instructions whose functions are associated with locating and correcting commands and data.

6.9 Instructions that Move the Pointer

There are several editor instructions whose only purpose is to move the pointer. They permit you to move about in the edit file. They may be abbreviated to the initial bold portion of their names:

1. **TOP** places the pointer before the first record;

2. **F**IRST places the pointer at the first record;
3. **BOT**TOM places the pointer after the last record;
4. **L**AST places the pointer at the last record;
5. **DO**WN moves the pointer down;
6. NEXT moves the pointer to the next record;
7. **U**P moves the pointer up.

DOWN, NEXT and UP may each be used with a numeric argument to indicate *how many* records (lines) the pointer is to move. For example, UP 3 moves the pointer up three records. When the editor is in command mode, the command pointer is moved the appropriate number of *commands*. When the editor is in data mode, the data pointer is moved the appropriate number of *data records*.

6.10 Other Instructions that May Move the Pointer

A number of other editor instructions also move the pointer in the course of performing a given task. They may also be abbreviated to the initial bold portion of their names:

1. **AGAIN** repeats the previous instruction;
2. **DEL**ETE deletes one or more records;
3. **I**NPUT inserts a record or enters input mode;
4. **KEEP** keeps one or more records;
5. **L**OCATE locates a particular record;
6. **SY**NTAX checks one or more commands for syntax;
7. **T**YPE types one or more records.

AGAIN repeats the previous editor instruction. It moves the pointer only if that instruction moves the pointer. The equal sign (=) is a synonym for AGAIN. In this example, LOCATE is an instruction that moves the pointer, and AGAIN repeats that instruction to locate a second occurrence of the same string:

```
      EDITING DATA RECORDS:
>>    LOCATE /Ed
          5=Ed Elliot 21 1 165
      EDITING DATA RECORDS:
>>    AGAIN
          7=Ed Elliot 21 1 165
```

TYPE and **SY**NTAX move the pointer only when they are entered with arguments. **T**YPE 3 causes three lines to print at the terminal and leaves the pointer at the last record typed. **T**YPE ***** types *all* the remaining records and leaves the pointer at the bottom. The bottom, however, is beyond the last record. **T**YPE * *****, issued in the command editor, types all remaining commands and their data, leaving the pointer at the bottom. **T**YPE **1 ***, issued in the command editor, prints a single command and all its associated subcommand and data records. The pointer does not move.

DELETE deletes the record indicated by the pointer and then moves the pointer to the next record. **DEL**ETE **3** deletes three records (the current record plus the next two) and moves the pointer to the next surviving record. **DEL**ETE ***** deletes the remaining records and leaves the pointer at the bottom.

6.11 Locating a Specific Record

The LOCATE instruction, which is used to find a particular string or record, has two forms:

```
LOCATE nn          (nn is the number of the record to be located)
LOCATE /string     (string is a particular phrase to be located)
```

LOCATE 1 is the same as FIRST. LOCATE /1 is a request to locate the first record that contains the character "1". Using LOCATE with a string argument is illustrated in Figure 6.2, line 8.

Figure 6.2 Using the Editor To Correct an Error in Data

```
1. >>   E

        The editor file has 2 commands.
        The pointer is at the last command.
           2...LIST   MyFile  $
        EDITOR:

2. >>   TOP

        At the TOP
        EDITOR:

3. >>   T  * *
           1...MAKE  MyFile,
                  VARS  First.Name:C16 Last.Name:C16 Age Sex Weight ;
             1=Mary Smith 28 2 125
             2=Don Smith 19 1 160
             3=Debbie Bunker 26 2 140
             4=Nancy Howe 24 1 130
             5=Ed Elliot 21 1 165
             6=Allison Wing 23 2 118
             7=Ed Elliot 21 1 165
             8=$
           2...LIST   MyFile  $

        At the BOTTOM
        EDITOR:

4. >>   UP 2
           1...MAKE  MyFile, .....
        EDITOR:

5. >>   DATA
        This command has 8 data or subcommand records.
        The pointer is at the last one.
             8=$
        EDITING DATA RECORDS:

6. >>   FIRST
            1=Mary Smith 28 2 125
        EDITING DATA RECORDS:

7. >>   CHANGE /28/25/
            1=Mary Smith 25 2 125
        EDITING DATA RECORDS:

8. >>   LOCATE /ed/
            5=Ed Elliot 21 1 165
        EDITING DATA RECORDS:

9. >>   DELETE
            5=Allison Wing 23 2 118
```

```
          EDITING DATA RECORDS:

 10. >>   X
          Input terminated by a single $.

          6 cases and 5 variables processed.
```

When LOCATE is used with a *string*, the search for that string begins with the next record. The pointer is moved to the record that contains the string unless the string is not found. If the string is not found, the pointer moves to the bottom. LOCATE /string moves *downwards* from the current record towards the bottom.

LOCATE *nn* moves the pointer either up or down. When the pointer is at record 7, LOCATE 2 moves the pointer up to record 2 and LOCATE 11 moves the pointer down to record 11.

LOCATE /string, TYPE, SYNTAX, DELETE, KEEP and EXECUTE all work from the current location of the pointer downwards to the bottom of the edit file. If the pointer is at the seventh record, SYNTAX * causes all the records from the seventh record onward to be checked for syntax and leaves the pointer at the bottom.

TOP and FIRST function in exactly the same way except when records are inserted. If the pointer is at the top, any records that are inserted are placed *above* the first record. If the pointer is at the first record, newly inserted records are placed *below* that record. BOTTOM and LAST are logically the same. The effect of LOCATE /string when the pointer is at the bottom is to move the pointer to the top and search the entire edit file.

In Figure 6.1, when the file was created, Ed Elliot's record was entered twice by mistake. Figure 6.2 illustrates how to use the editor to delete one data record and recreate the file. The letter "E" alone on a record is a short notation for the EDITOR command. Before the data records for the MAKE command can be corrected, the editor must be entered and the pointer must be moved to that command:

```
>>   UP 2
          1...MAKE  MyFile,
                 VARS  First.Name:C16 Last.Name:C16
                       Age Sex Weight ;
     EDITOR:
```

Once that command is located, the DATA instruction allows access to the data records:

```
>>   DATA
     This command has 8 data or subcommand records.
     The pointer is at the last one.
          8=$
     EDITING DATA RECORDS:
```

FIRST (**F**) moves the pointer to the first of those data records. CHANGE (C) nis now used to correct that data record:

```
>>   FIRST
         1=Mary Smith 28 2 125
     EDITING DATA RECORDS:
>>   CHANGE /28/25/
         1=Mary Smith 25 2 125
     EDITING DATA RECORDS:
```

LOCATE (L) moves the pointer to the first record for Ed Elliot. DELETE removes that record and finally EXECUTE (or just X) re-executes the command, creating an updated version of the file:

```
>>   LOCATE /ed/
         5=Ed Elliot 21 1 165
     EDITING DATA RECORDS:
```

```
>>   DELETE
         5=Allison Wing 23 2 118
     EDITING DATA RECORDS:
>>   X
```

Figure 6.3 provides another illustration of how various editor instructions move the pointer and make corrections. In this example, there are four commands in the edit file. The pointer is initially at the bottom but is moved to the top by the first instruction, **TOP**. TYPE * * (T * *) causes all commands *and* data / subcommand records to be typed on the terminal and leaves the pointer at the bottom. If the instruction had been TYPE *, all the commands would be typed, but not their data or subcommands.

Figure 6.3 Moving the Pointer

```
     >>   E

          The editor file has 4 commands.
          The pointer is at the last command.
             4...TABLES  X2 ;

          EDITOR:
 1.  >>   TOP

          At the TOP
          EDITOR:
 2.  >>   T  * *
             1...HELP  CORRELATE  $
             2...CORRELATE  X2,
                       OUT  Xcor       $
             3...LIST  Xcor,
                    PLACES  2          $
             4...SURVEY  X2 ;
               1=BANNER  Age,  STUB  Sex ;
               2=$
          At the BOTTOM
          EDITOR:
 3.  >>   UP 3
             2...CORRELATE  X2,
                       OUT  Xcor  $
          EDITOR:
 4.  >>   NEXT
             3...LIST  Xcor,
                    PLACES  2  $
          EDITOR:
 5.  >>   DOWN 1
             4...SURVEY  X2 ;
          EDITOR:
 6.  >>   WHERE

          You are in the main P-STAT editor, editing commands.
          EDITOR:
 7.  >>   DATA
```

```
           This command has 2 data or subcommand records.
           The pointer is at the last one.
                2=$
           EDITING DATA RECORDS:
  8.  >>   UP
                1=BANNER  Age,  STUB  Sex ;
           EDITING DATA RECORDS:
  9.  >>   CHANGE /Age/Education
                1=BANNER  Education,  STUB  Sex ;
           EDITING DATA RECORDS:
 10. >>    WHERE

           You are in the data or subcommand editor.
           R  returns you to the command editor.
           EDITING DATA RECORDS:
 11. >>    R

           Returning to the main ( command ) editor.
           EDITOR:
```

The fifth reply, DOWN 1, moves the pointer to the SURVEY command. WHERE is used to find out where you are — which editor is in control. Since it is the main P-STAT editor, the data editor must be entered before the data records for the SURVEY command may be accessed. DATA, the seventh instruction, enters the data editor and the pointer is now located at the last data record of the SURVEY command.

The second WHERE instruction indicates that the user is editing data records. At this point, using TOP, TYPE, and so on, affects only those data records (that is, the table definitions) that belong to the current command SURVEY.

6.12 Checking for Correct Syntax

The EDITOR instruction SYNTAX can be used to check one or more commands to be sure that they follow the rules for correct syntax in P-STAT. This is particularly useful when the command has many programming language phrases.

Figure 6.4 illustrates using **SY**NTAX after a change has been made in the editor. Since SYNTAX is used without an argument, only the command that is currently pointed to is checked for syntax. The pointer is not moved. In this example, a close examination of the command reveals that the syntax is incorrect. The editor prints commands according to the structure of the phrases. Each new modification phrase begins on a new line on the terminal. This is not the case in this example because the KEEP phrase does not end with a semi-colon or a balancing right bracket.:

```
2...LIST  MyFile  [ KEEP  Age  Sex  IF Age < 21,
            DELETE ;  IF Sex = 1,  RETAIN ]  $
```

The difference between the formatting of a command with a semi-colon missing at the end of a phrase or an incorrect pairing of the square brakcets and the same command correctly entered can be of help in determining some of the syntactical problems with a command. When the phrases are properly terminated, the command looks like this in the editor:

```
2...LIST  MyFile  [ KEEP  Age  Sex ;
          IF Age < 21, DELETE ;
          IF Sex = 1,  RETAIN ]  $
```

Figure 6.4 Syntax Check in the Editor

```
           2...LIST  MyFile  [ KEEP Age Sex
 IF Age < 21, DELETE;                            IF Sex = 1,   RETAIN ]  $
        EDITOR:
 1. >>   CHANGE /[ /[ KEEP  Age  Sex
           2...LIST  MyFile  [ KEEP  Age  Sex IF Age < 21, DELETE;
                     IF Sex = 1, RETAIN ]  $
        EDITOR:
 2. >>   SYNTAX

          Error... In the KEEP or DROP phrase starting with:
           [ KEEP Age Sex IF Age <21, DELETE; IF Sex = 1, RETAIN ] $
          an invalid keep or drop element has been found at:
           <21, DELETE; IF Sex = 1, RETAIN ] $

 Syntax error in the current command.
    2...LIST MyFile [ KEEP Age Sex IF Age <21, DELETE;
             IF Sex = 1, RETAIN ] $
E         EDITOR:
 3. >>   CHANGE /x i/x; I
           2...LIST  MyFile  [ KEEP  Age  Sex ;
                      IF Age < 21,  DELETE ;
                      IF Sex = 1,   RETAIN   ]  $
        EDITOR:
```

6.13 Inserting a Single Record in the Edit File

The editor instruction INPUT can be used *in edit mode* to add a single record to the edit file:

```
INPUT (followed by the record to be added)
```

The new record is placed immediately after the record indicated by the pointer and the pointer now points to the new record.

There are two ways to enter a new case into the BUILD command in Figure 6.2. In each case the first few steps are the same. The following sequence of EDITOR instructions may be used:

1. E — to enter the editor;
2. TOP (T) — to place the pointer at the top of the editor;
3. LOCATE /MAKE — to locate the first MAKE command in the file (E L/MAKE may be entered to combine the procedures in steps 1 and 3);
4. DATA (D) — to enter the DATA editor once the proper MAKE command has been located;
5. UP (U) — to move the pointer just above the terminating $ in the data records.

At this point the INPUT instruction can be used in the edit mode to insert a single record. The record is inserted immediately following the record at which the pointer is located, and the pointer is moved to the new record:

```
7=Ed Elliot 21 1 165
EDITING DATA RECORDS:
```

```
>>    I Margaret Johnson 29 2 153
      EDITING DATA RECORDS:
>>    T
      8=Margaret Johnson 29 2 153
```

Any of the editor instructions can now be given. The second way to input these records is to go into *input* mode. Input mode is described in the next editor chapter.

These instructions for adding another case to a P-STAT system file are suitable when the file is a small one that has been built interactively at the terminal. They also illustrate how another subcommand could be added to those belonging to a command, such as REGRESSION or PLOT. There are more efficient ways to add cases to existing P-STAT files than rebuilding the original file. CONCAT joins a file of new cases to an existing file and UPDATE revises an existing file.

6.14 MORE ADVANCED EDITOR INSTRUCTIONS

Additional editor instructions provide for:

1. global locates and changes,
2. saving and accessing edit files,
3. putting and getting data records,
4. moving and copying commands, and
5. fixing larger or elusive errors.

6.15 Repeating Editor Instructions for Global Corrections

It is often necessary to repeat one or two editor instructions across a number of commands. For example, consider an edit file that was created and used to process a file named January. It is now February and the same commands are to be done again. It is not difficult to access the edit file, enter it, LOCATE /January and CHANGE /January/ February. However, it is a tedious process if this must be done more than once.

There are two user defined editor instructions, Y1 and Y2, that can become whatever instructions the user wishes. They are defined in the editor:

```
      EDITOR:
>>    Y1 LOCATE /January
      EDITOR:
>>    Y2 CHANGE /January/February
```

To use these newly defined instructions, enter Y1 to do the LOCATE, Y2 to do the CHANGE, Y1 to do the next LOCATE, and so on.

6.16 Saving and Restoring the Edit File

There are four editor instructions that are only used in the *command* editor. They save and access the edit file. One set of instructions deals with a *binary* representation of the edit file for quick reading and writing of the file by P-STAT; the other set deals with an *alphanumeric* representation that permits editing of the file outside of P-STAT:

1. `BACKUP 'fn'` saves a binary representation,
2. `RESTORE 'fn'` accesses a binary representation,
3. `EXPORT 'fn'` saves an alphanumeric representation, and
4. `IMPORT 'fn'` accesses an alphanumeric representation.

Each of these instructions requires the name of the edit file in quotes, is issued in the editor, and does not require a dollar sign (they are not commands, but editor instructions):

```
         EDITOR:
    >>   BACKUP  'June12.edt'

         About to rewind and WRITE to file June12.edt .
         Reply  YES  to verify this should be done.
    >>   yes

         8 commands successfully backed up
         in binary form on external file June12.edt .
```

BACKUP saves the entire edit file in an external binary file with the specified name. RESTORE is used in subsequent interactive sessions to restore that edit file — to *replace* the current edit file. These edit files are in a special editor format that only the P-STAT editor can easily read and write. BACKUP and RESTORE are typically used to save and access a series of commands or macros for use in subsequent P-STAT sessions.

EXPORT saves the edit file in an alphanumeric (card image) format that can be read by an operating system editor or submitted to a batch operating system for execution. IMPORT reads an alphanumeric file, created by EXPORT or by an operating system editor, and *adds it to the bottom* of the current edit file. EXPORT and IMPORT are typically used to save and access a series of commands for editing, printing, and using in subsequent P-STAT sessions as *transfer* files.

The P-STAT commands NEW.EDIT.FILE and OLD.EDIT.FILE, described in the chapter “Utility Commands for Interactive Use”, also save and access edit files. These commands differ from BACKUP / RESTORE and EXPORT / IMPORT because the edit files that they write and access *grow* — as commands are entered into P-STAT, they are written in this *permanent* edit file. These edit files are just like the temporary edit file that exists in all interactive P-STAT sessions. All additions, deletions and corrections to commands and data or subcommands are reflected in the edit file. The permanent edit file differs only in that it can be accessed in a later P-STAT session, whereas the temporary edit file disappears after a P-STAT session ends.

When an edit file is saved using BACKUP from within the editor, it is stored in the specified external file *as it exists at that time*. The current edit file, whether it is another permanent external file or a temporary work file, continues to grow as the run progresses. The external file that was created using BACKUP *does not change*. When RESTORE is used later in the session or in a subsequent session, the edit file that is restored is exactly what was backed up earlier. The version on disk does not change even though the restored edit file continues to grow.

When the edit file is placed in an external file by the NEW.EDIT.FILE and OLD.EDIT.FILE commands, that external file becomes the current edit file. Any new command, subcommand and data records are placed directly in that external file, which continues to grow as the run progresses.

RESTORE is normally used when the edit file that was saved is *static* and is not intended to be changed. OLD.EDIT.FILE is normally used for on-going projects where new commands are continually executed and the edit file is used as a *log* for that project.

6.17 Saving and Restoring Subcommand and Data Records

There are two instructions that work only in the *data* editor — GET and PUT. Both require the name of an external file in quotes. GET copies *alphanumeric* data records from an external file into the P-STAT edit file, where they are then available for changes, deletions, and so on. PUT copies all the data for the current command into an external file, using alphanumeric representation.

Figure 6.5 illustrates using GET to copy data records for the MAKE command from an external file named “Input.Dat” into the data editor. A change is made to one of the records and then PUT writes the corrected records back out to the same external file. The MAKE$ command can now be executed with the corrected data. The PUT instruction need not be used unless a corrected version of the raw data is desired in an external file.

Figure 6.5 GET and PUT with Data Records

```
            6...MAKE  MyFile,
                   VARS  First.Name:C16 Last.Name:C16 Age Sex Occupation ;
         EDITOR:
 1. >>   DATA

         This command has no data or subcommand records.

         At the top of the data
         EDITING DATA RECORDS:
 2. >>   GET  'Input.Dat'

         45 data or subcommand records read from file Input.Dat .
         EDITING DATA RECORDS:
 3. >>   TYPE
              46=$
         EDITING DATA RECORDS:
 4. >>   UP
              45=Sophia Miller 31 2 54
         EDITING DATA RECORDS:
 5. >>   C /31/13
              45=Sophia Miller 13 2 54
         EDITING DATA RECORDS:
 6. >>   PUT  'Input.Dat'
         About to rewind and WRITE to file Input.Dat .
         Reply  YES  to verify it should be done.
 7. >>   YES

         45 data or subcommand records written to file Input.Dat .

         At the bottom of the data.
         EDITING DATA RECORDS:
```

An alternate way to edit an external file is to use the operating system editor or a word processing package. (Do not use tabs or other unusual characters if you use a word processor.) This may be more efficient with a large external file of data. The GET / PUT procedure is useful when you are not familiar with the operating system editor or other software. It is useful, for example, for correcting an external file of value and variable labels that was originally created using the SAVE.LABELS command within P-STAT. Labels are considered to be data by P-STAT.

Enter the SAVE.LABELS command with a *new* external filename and then enter just the "$" to get out of label input mode:

```
      Enter a command:
 >>   SAVE.LABELS 'Tree2.lab' ;
      Enter labels:
 >>   $
      1 record was written to new labels file Tree2.lab.
      The file will now be checked for errors. No errors were found*u.
```

Enter the editor and then the data editor. Delete the dollar sign and get the labels from the original label file:

```
     Enter a command:
>>   E

     The editor file has 3 commands.
     The pointer is at the last command.
         3...SAVE.LABELS 'Tree2.lab';
     EDITOR:
>>   D
     This command has 1 data or subcommand record .
          1=$
     EDITING DATA RECORDS:
>>   DEL

     At the top of the data.
     EDITING DATA RECORDS:
>>   GET 'Tree.lab'

    13 data or subcommand records read from file Tree.lab .
     EDITING DATA RECORDS:
```

Make the necessary corrections to the labels and then execute the SAVE.LABELS command. The old label file may be erased and the new label file renamed with the old file's name, if desired.

6.18 Moving and Copying Commands

MOVE and COPY are editor instructions that move or copy one or more *commands* in the edit file. They are used only in the *command* editor. The placement of the moved or copied commands can be specified either by the pointer number or by an asterisk (*) if the commands are to go at the end:

`COPY`	Copy this command after itself.
`COPY 6`	Copy 6 commands, place after the 6th.
`COPY 6 AFTER 11`	Copy 6 commands, place after the 11th.
`COPY 2 TO *`	Copy 2 commands, place at the end.
`COPY 1 BEFORE 5`	Copy 1 command, place before 5th.

The word TO is a synonym for BEFORE. MOVE works exactly the same way except that MOVE by itself makes no sense. MOVE and COPY aide in "cleaning up" (organizing) the edit file prior to saving it for use at a subsequent time.

6.19 Using the SQUEEZE Operator

SQUEEZE is a string operator that can be used to delete whatever is between two character strings. For example:

```
SQUEEZE /Xcor/ $/
```

applied to the third command in Figure 6.3:

```
LIST  Xcor,  PLACES 2  $
```

squeezes out the comma and the string “PLACES 2” and leaves “LIST Xcor $”. This is an easy way to delete a long string of characters.

SQUEEZE //A/ squeezes out all characters from the *beginning* of the record to the character string “A”. SQUEEZE /A// squeezes out everything from the string “A” to the *end* of the record. SQUEEZE /// blanks out the entire record.

Figure 6.6 Fixing a Transmission Error

```
 1. >>   CORRELATE  File2,  OUT  FileCor  $

         ERROR...  The previous identifier phrase did not end with a comma
                   2, O

         EDITOR entered.
            1...CORRELATE  File2,
                      OUT  FileCor  $
         EDITOR:
 2. >>   CHANGE /E File2,/E File2,/

         These characters were not found.
         E File2,
         EDITOR:
 3. >>   VALIDATE

            1...CORRELATE  File?2,
                      OUT  FileCor   $
         EDITOR:
 4. >>   CHANGE/?2/2/
            1...CORRELATE  File2,
                      OUT  FileCor   $
         EDITOR:
 5. >>   EXECUTE

         Correlate completed.
         4 cases were read.
         There was no missing data.
```

6.20 Fixing a Transmission Error

Stray characters occasionally appear on a terminal because of transmission errors on telephone lines or computer cables. If the bit pattern of the transmission error has a representation on the keyboard, it can be seen and corrected with the CHANGE instruction. If, as it sometimes happens, it has no keyboard representation, it may appear as an extra blank or may not show up at all. Figure 6.6 illustrates fixing a transmission error.

A spurious character exists between “e” and “2” in “File2”. Depending on the character, the operating system, and other things, it may print on a line printer as “File 2” but show on a terminal screen as “File2”, with no apparent problem until P-STAT tries to interpret it. An example of a stray character is one that clears the screen when it types out. The result is an error that seems to have no cause. This is the situation in Figure 6.6 — there does not appear to be an error.

When the record is short, the easiest correction may be with the editor instruction REPLACE. For example, this replaces the current command with the text following REPLACE:

```
REPLACE CORRELATE  File2,  OUT FileCor $
```

If the record is long, it may be easier to locate the transmission error and remove it. In Figure 6.6, the editor instruction VALIDATE is used to locate any invalid characters. VALIDATE changes anything that does not have a representation on a normal keyboard into a “?”. This makes it easy to see the stray character. The CHANGE instruction is then used to change “File?2” to “File2”.

SUMMARY

EDITOR

```
EDITOR  $
 E
```

Many editor instructions may be abbreviated to just their essential portion. In the list that follows, the full instruction is given with the essential portion in *uppercase bold* characters. For example, **C**hange may be entered as “C”, while EXPORT must be entered in full.

Several of the instructions are followed by (nn) or (nn or *). This indicates an instruction that can be used by itself to apply to only the current record or can be followed by a numeric argument indicating the number of records to which the instruction applies. When the argument is a “*” and the editor is the *command* editor, the instruction applies to all the commands from the current one through the last one in the edit file. When the editor is the *data* editor, the instruction applies to all the data records from the current one to the end of the data records for that command.

AGAIN

requests that the previous EDITOR instruction be repeated. An equal sign (=) is a synonym for AGAIN.

BACKUP 'fn'

saves the edit file by making a copy of it (as it exists at this point) in the external file named “fn”.

B

inserts a blank line. This is used in data input only.

Beginning /S

inserts string S at the beginning of the command or data record.

BOTtom

sets the pointer beyond the last command or data record in the edit file.

Change /S/T/

changes string S into string T.

Change /S/T/A

changes all occurrences of string S into string T across the entire record.

Command

returns from data mode to command mode. This is used in data mode.

COPY nn BEFORE p

copies one or more (*nn*) commands with their associated data records BEFORE (TO) or AFTER *p*, where *p* is the position of a command: This instruction copies two commands (the current one and the one below it) and places them after the sixth command:

```
COPY 2 AFTER 6
```

Data

goes from the command editor to the data editor. This is used in the command editor.

DELete (nn or *)

deletes commands and corresponding data when in the command editor. It deletes only data records when in the data editor.

DOwn (nn)

moves to the next command or data record.

Editor

returns to the main or *command* editor from the data editor. This is used in the data editor.

End /S

places string S at the end of the record, after the last non-blank.

E**X**ecute (nn or *)

executes (does) the current command — the one pointed to by the pointer, and returns to the command-entry mode. This is typically the action taken after making an error, entering the editor, and correcting the error.

X is an abbreviation for EXECUTE. If an integer follows (X 2), the specified number of commands are executed. If * follows X (X *), all the commands from the current command through the last one in the edit file are executed. If E follows X 2 (X 2 E), the specified number of commands are echoed (printed) and executed. If no argument follows, just the current command is executed.

In addition, brief commands not currently in the editor may be executed. This command (given anywhere within the editor) executes the FILES command and returns to the editor at the current location:

```
X  FILES  $
```

The command itself is not entered into the edit file.

EXPORT 'fn'

writes the entire edit file (as it exists at this point) in alphanumeric (card-image) format in the external file named “fn”.

First

moves the pointer to the first command or data record in the edit file.

GET 'fn'

inserts data, currently in alphanumeric (card-image) format in the external file named “fn”, into the data editor associated with the current command. This instruction is used only in the data editor.

Help

prints help messages.

IMPORT 'fn'

reads an alphanumeric (card-image) file of P-STAT commands and data or subcommand records. These are added to the bottom of the current edit file.

Input

enters input mode, when the rest of the line is blank. If it is not blank, the string following the INPUT instruction is inserted after the current record. This may be used in either the command or data editors.

KEEP (nn or *)

keeps the specified number of commands or data records, beginning at the pointer. *All others are deleted*.

Last

moves the pointer to the last command or data record.

Locate nn

locates the *nth* command or data record, and positions the pointer there.

Locate /S

locates string *S* and positions the pointer there. Searching is downward. If the pointer is at the bottom when starting, it shifts to the top and searching commences. If the string is not located, the pointer ends up positioned at the bottom.

MOVE nn AFTER p

moves one or more (*nn*) commands with their associated data records BEFORE (TO) or AFTER *p*, where *p* is the position of the command. (See COPY, which is similar to MOVE.)

Next (nn)

moves down one or the specified number of command or data records.

OLD

restores the *original* version of a command or data record before any changes had been made.

PUT 'fn'

copies all data records associated with the current command to the external file named “fn”. This instruction is used only in the data editor.

Quit

leaves the editor. P-STAT then prompts for a command.

Return

returns to the previous higher level of the editor.

REPlace S

replaces the current command or data record with string s.

RESTORE 'fn'

restores or copies a previously saved edit file, located in the external file named “fn”, into the current edit file. This *replaces* the current edit file.

Squeeze /S/T

drops (squeezes, deletes, blanks out) all characters between the first occurrence of string *S* and the first occurrence of string *T*.

Squeeze can also be used as “//S”, “/S” and “///” to delete everything up to string *S*, after string *S*, or to blank out the entire record, respectively.

SYntax (nn or *)

requests a syntax check of one or more commands. Checking stops when the first error is encountered and the pointer is located at the record with the error.

TOP

sets the pointer *above* the first command or data record in the edit file.

Type (nn or *)

types the command or data record.

Type * *

types *all* commands *and their data*. "T 1 *" can be used to type the *current* command and all its data records.

Up (nn)

moves the pointer up one or the specified number of command or data records.

Where

indicates where you are located in the editor.

VALidate

requests that each character in a command be checked to determine that it is a valid character. If a character is not a valid (printable) character, it is replaced with a question mark (?).

Y1

may be defined by the user for repeated execution of any EDITOR instruction:

```
Y1   LOCATE /January
Y2   CHANGE /January/February
```

Y2

may be defined by the user for repeated execution of any EDITOR instruction. (See Y1 above.)

Z

use the full screen editor to make changes to the current command. Z can be used from within both the command and data editors and provides access to the entire command including all subcommands and data records. Z is described in full in the next chapter.

7
The Editor For Input

The editor may be used for input as well as for editing. In *edit mode* existing records may be changed, deleted or replaced. A single record can be inserted into the edit file, command syntax can be checked, and edit files can be backed up and restored. (The previous three chapters on the P-STAT editor described edit mode and full screen edit.) In *input mode*, only the addition of records to the edit file is possible. No corrections can be made until edit mode is resumed. Input mode is entered by using the editor instruction INPUT or the abbreviation “I”.

The distinction between the command editor and the data editor exists in input mode, as well as in edit mode. This means that there are four distinct functions in the P-STAT editor:

1. COMMAND INPUT inputting commands;
2. DATA INPUT inputting data;
3. COMMAND EDITOR editing commands;
4. DATA EDITOR editing data.

Note: the full screen editor described in the previous chapter can also be used for input of a single command as long as there is at least one character in the command buffer.

7.1 INPUT MODE

Whether input mode is ever used depends on the user’s personal preference. It is possible to enter one command at a time, correct errors as they occur and examine the results of each step before proceeding with the next. In such cases, it is not necessary to use input mode. On the other hand, if the user prefers to enter several commands at a time, check them for syntax and proceed with execution only when everything appears to be correct, input mode is appropriate.

When the *command* editor is in input mode, the prompt is:

```
Input a command:
```

The prompt continues to be for commands until either a $ or a semicolon is processed. When the *data* editor is in input mode, the prompt is:

```
Input data:
```

The prompt continues to be for data information until either a $ or a null return (blank line) is processed.

Instructions, such as “E” for EDITOR, “D” for DATA, and “R” for RETURN, move between command and data edit and input modes. “H”, for HELP, provides extensive messages for both data and command input. “W”, for WHERE, briefly describes whether the user is in command input mode, data input mode, the main P-STAT command editor, or the data editor.

7.2 The INPUT Instruction

The INPUT instruction works two ways in the editor. When the instruction INPUT (or “I”) is entered *by itself*, the editor enters input mode where commands and their data may be entered into the edit file. Input mode continues until a return to edit mode is made.

When the INPUT instruction is immediately *followed by characters* to be inserted at a particular location in the edit file, input mode is *not* entered. Only those characters are added to the edit file and the editor remains in edit mode with the pointer positioned at the new record.

7.3 Location of the Pointer

When the INPUT instruction is entered by itself, new input records are placed *below* the current location of the pointer. The pointer moves as each new record is added. After all records have been entered and a return to edit mode is made, the pointer is located at the last record entered. If input mode is entered when the pointer is at the bottom, the result is the same as if the pointer had been at the last record. The new record is placed at the end and the pointer is located at that new record. It is possible to insert new records at any location in the file by moving the pointer up or down.

The editor does not know when the input of *all* commands and data records is complete until a specific indication is given by the user. When in data or command input mode, the *single letter* “R” or a null record indicates that input is finished. Either of these signals cause the editor to move from input mode back to edit mode. A null record is indicated by entering either a totally blank line or by pressing return (enter) on the keyboard.

A blank line and a null record are treated in the same way and serve to transfer control from one mode to another. Therefore, a special convention, the use of the single letter “B”, is needed to produce an entirely blank data record.

7.4 Editor Prompting

The editor issues the correct prompts according to the structure of each P-STAT command. Command text that ends with a semicolon is a signal that the command has data or subcommand records. In such a case, the next editor prompt is “Input data:”. A $, either at the end of command text or at the end of subcommand or data records, is a signal that the command is complete and the next prompt should be for a new command. As a result, the editor is able to move from command input to data input automatically.

7.5 BUILDING A RUN

Figure 7.1 illustrates how to use the editor to build a run (command stream) interactively. For purposes of illustration, each user entry is numbered and flagged on the left. Two commands are entered and both are checked for syntax before the commands are executed.

The editor is entered by typing the EDITOR command, followed by the required “$”, or just “E”. The editor reports that it has one command and then issues the prompt message “EDITOR:”. The user enters INPUT which places the editor in command input mode. All new records are placed *after* the current location of the pointer. If the pointer is at the top, the new records are positioned at the top of the file before any other existing records. If the pointer is located at the first record, as in this example, the new records follow that first record. The pointer moves to each record as it is added to the edit file.

Once the editor instruction INPUT is typed, P-STAT prompts with “Input a command:”. In input mode, commands may be entered continuously without being checked for syntax or executed. Each time a command is entered and a $ is encountered, the editor assumes that the command is complete and prompts the user for the next command. The editor at this point has no idea whether the commands have been entered correctly or are syntactically correct. It is only when they are specifically checked for syntax or executed that the commands are checked for errors.

Figure 7.1 Building a Run Using the Editor

```
 1. >>   EDITOR  $

          The editor file has 1 command .
            1...BUILD  Job1, .....
         EDITOR:
 2. >>   INPUT

         Entering command input mode:

         Input a command:
 3. >>   MODIFY  Job1 [ IF  Age < 30,  DELETE,  OUT  Job2   $

         Command completed.
         Enter D, or another command, or R, etc.
 4. >>   LIST  Job2,  GAP 3  $

         Command completed.
         Enter D, or another command, or R, etc.
 5. >>   R

         Returning from command input mode to the main editor.
         EDITOR:
 6. >>   TOP

         At the TOP
         EDITOR:
 7. >>   SYNTAX *

         Error... A dollar has been found ending a record:
                     MODIFY Job1 [ IF Age < 30, DELETE, OUT Job2 $
                  The brackets, however, are not balanced.
                  The earliest left bracket still open began here:
                     [ IF Age < 30, DELETE, OUT Job2 $

         Syntax error in the current command.
         The command before it had no errors.
            2...MODIFY Job1 [ IF Age < 30, DELETE, OUT Job2 $
         EDITOR:
 8. >>   CHANGE /UDE,/UDE ],,
            2...MODIFY  Job [ IF  Age < 30,  DELETE ],
                   OUT  Job2   $
         EDITOR:
 9. >>   SYNTAX *
         2 commands were checked for syntax.
         No errors were found.
         EDITOR:

10. >>   UP
            2...MODIFY  Job [ IF  Age < 30,  DELETE ],
```

```
                  OUT  Job2   $
        EDITOR:

 11. >>   EXECUTE *
```

7.6 Checking For Syntax

In Figure 7.1, two commands are input into the edit file, and then "R" for RETURN is entered (entry 5). Control now passes from input mode back to edit mode. The pointer is at the last command entered. The instruction TOP moves the pointer to the top of the edit file. The instruction SYNTAX begins syntax checking. If SYNTAX is used without an argument, only the current command indicated by the pointer is checked. If SYNTAX 2 is used, two commands are checked for syntax, beginning with the one indicated by the current position of the pointer. The use of "*" means to check *all* the commands from the current command through the *end* of the edit file.

In this example, the syntax check does find one error. As a result, control returns to the command editor with the pointer located at the command that contains the error. The problem is unbalanced brackets in the MODIFY command.

The CHANGE instruction (or "C") corrects the error and SYNTAX * resumes syntax checking. When it appears that the run is syntactically correct, UP moves the pointer to the MODIFY command. EXECUTE * begins the execution of the commands. If no execution errors are found, both commands are executed before control returns to the terminal. If an execution error does occur, the editor is entered automatically. When the error is corrected, execution can resume from the edit file.

7.7 Correcting the Edit File

In input mode, all entries are entered into the edit file with the exception of the special instructions "E" and "R" that return control to edit mode. Corrections can be made only when edit mode is resumed. A common error is to enter a number of records and then make a change *without returning* to edit mode. As a result that change is added to the edit file, as are any subsequent editor instructions, until a return to edit mode is made. For example:

```
      Input a command:
>>    LIST  MyFile  $

      Command completed.
      Enter D, or another command, or R, etc.
>>    CHANGE /MyFile/NewFile/

      Continue inputting the command:
```

At this point the "Continue inputting the command:" prompt is an indication that the change was entered in command input mode and that it is being added as a command to the edit file. An easy way to fix this is to enter a $, which completes command entry as far as the editor is concerned:

```
>>    $

      Command completed.
      Enter D, or another command, or R, etc.
>>    R
```

Now that the command is complete, "R" returns control to edit mode and the change instruction can be deleted from the edit file. In this example, the instruction TYPE prints the last command entered:

```
      Returning from command input mode to the main editor.
      EDITOR:
>>    TYPE
         9...CHANGE /MyFile/NewFile/  $
```

```
        EDITOR:
>>      DELETE

        At the BOTTOM
        EDITOR:
```

The command is then removed by entering the instruction DELETE. Once this is done, the pointer moves to the next command in the edit file or to the bottom if there are no other commands. The UP instruction may now be entered to locate the pointer at the LIST command that needs correction.

7.8 ENTERING COMMAND AND DATA RECORDS

Figure 7.2 shows the input of both command and data records. As in Figure 7.1, “E” (or EDITOR $) is entered. The pointer is positioned at the top of the edit file and the INPUT instruction is entered. The BUILD command is input. It could build File2 with three variables named Grade, Sex and Age. When the semicolon is detected at the end of the BUILD command, the editor prompts for data entry. The data entry prompts continue until the dollar sign is processed. The editor assumes that a single dollar sign on the data record indicates the end of the command and the prompt should be for the next command. (Note that a blank or null line could be used instead to end data entry.)

Figure 7.2 Editor input: Commands and Data

```
1. >>   E

        The editor file is empty.

        At the TOP
        EDITOR:
2. >>   INPUT

        Entering command input mode:

        Input a command:
3. >>   BUILD  File2,  VARS  Grade  Sex  Age ;

        Command completed.

        Input data:
4. >>   6 1 2

        Input data:
5. >>   (more data records entered)

        Input data:
6. >>   $
        A  $  was the last non-blank in the record.
        This data-subcommand input is completed.
        Enter D, or another command, or R, etc.
7. >>   LIST  $

        Command completed.
        Enter D, or another command, or R, etc.
8. >>   R
```

```
         Returning from command input mode to the main editor.
         EDITOR:
  9. >>  FIRST
            1...BUILD  File2,
                   VARS  Grade  Sex  Age ;
         EDITOR:
 10. >>  TYPE * *
            1...BUILD  File2,
                   VARS  Grade  Sex  Age ;
              1=6 1 2
              2=7 2 12
              3=7 1 11
              4=8 2 14
              5=$
            2...LIST  $

         At the BOTTOM
         EDITOR:
 11. >>  UP 2
            1...BUILD  File2,
                   VARS  Grade  Sex  Age ;
         EDITOR:
 12. >>  eXecute *

         BUILD of file FILE2 completed.
         It has 4 cases and 3 variables.
         Input was terminated by a single $.


          Grade   Sex   Age

             6      1     2
             7      2    12
             7      1    11
             8      2    14

         Enter a command:
```

Finally, a LIST command is entered and input is complete. The single letter "R" for RETURN goes back to the next higher level of the editor, which in this case is the main (command) editor. FIRST sets the pointer to the first command. TYPE * * types the command, along with all of its associated data, and any other commands that follow until the bottom of the edit file is reached. UP 2 moves back to the BUILD command. EXECUTE * causes execution to continue from the edit file until: an error is found, more data are needed, or the last command in the edit file is completed. In this case, processing is successful and File2 is listed.

7.9 Moving to Different Levels of the Editor

After moving from the command editor to command input to data input, using "R" for RETURN moves the user back one level from data input to command input. Issuing "R" again moves from command input to command edit. It is also possible to go from command edit to data edit to data input. In that situation, using "R" moves the user first from data input to data edit, and then from data edit to command edit.

In summary, “R” returns the user to the *next higher level* of the editor, using the same path the user took to enter that level. The chart in the summary section illustrates moving between the different levels of the P-STAT editor.

7.10 Entering Data Input From the Data Editor

Figure 7.3 shows how to input data directly from the data editor. It illustrates how help operates and the progression back from data input to data edit to command edit. In this example, “H” is entered when P-STAT prompts for “Input data:”. The help text briefly describes which part of the editor is in operation and prints all possible user replies. When help is completed, the user is placed back in data input mode, with the pointer located exactly where it was before help was requested.

Figure 7.3 Help with Data Input in the Editor

```
 1. >>   TYPE
         1='Ed Elliot' 21 1 165
         EDITING DATA RECORDS:
 2. >>   INPUT

         Input data:
 3. >>   H

         YOU ARE IN DATA INPUT MODE.

         3 TYPES OF INPUT ARE POSSIBLE HERE...

           (1) A NULL LINE, END OF FILE OR A TOTALLY BLANK LINE.
               THESE INDICATE THAT INPUT IS FINISHED.
               CONTROL RETURNS TO THE DATA EDITOR.

           (2) AN INPUT CONTROL CHARACTER ( LIKE H ),
               WHICH IS ALWAYS A SINGLE LETTER.

           (3) ANYTHING ELSE, WHICH BECOMES A DATA
               RECORD FOR THE CURRENT COMMAND.
               NOTE: DATA INPUT ENDS IF A RECORD HAS A $
                     AS ITS LAST NON-BLANK CHARACTER.

         CONTINUE HELP...REPLY YES OR NO.
 4. >>   YES
         THESE SINGLE LETTER INPUT RECORDS ARE TREATED
         AS EDITOR INPUT CONTROL CHARACTERS...
         B  GENERATE A BLANK DATA RECORD.
         E  GO BACK TO THE MAIN EDITOR.

         H  PROVIDE HELP.
         Q  QUIT THIS ACTIVITY. LEAVE THE EDITOR, GO BACK TO P-STAT ITSELF.

         R  RETURN ONE LEVEL, IN THIS CASE BACK TO THE DATA EDITOR.
         W  SAYS WHERE YOU ARE.
```

```
        X  EXECUTE THE CURRENT COMMAND AND ITS DATA.

        Input data:
 5. >>  'Mildred Murphy' 61 2 105

        Input data:
 6. >>  W
        You are now in data-subcommand input mode.
        R  returns you to the data editor.
        E  returns you to the main editor.

        Input data:
 7. >>  R

        Returning from data input mode to the
        data-subcommand part of the editor.
        EDITING DATA RECORDS:
 8. >>  TYPE 2
          8='Mildred Murphy' 61 2 105
          9=$
        EDITING DATA RECORDS:
 9. >>  X

        Input terminated by an end of file.
        8 cases and 4 variables processed.
```

In entry 5, information on Mildred Murphy is entered. At any point, the user may enter "W" for WHERE to find out which edit mode is in control at that particular time. Once this information is given, the pointer again returns to its previous location. The user issues "R" to get out of input mode and returns to data edit mode. TYPE 2 is then entered to see whether the data on Mildred Murphy was correctly input. Since this record is in the proper location, the command is executed.

Note that the "W" for WHERE and the "R" for RETURN are instructions and are not entered as data records. This means that it is not possible to enter data records that contain only these single letters into the editor unless they are enclosed in quotes. Also, note that if you wish to use HELP or WHERE from either command or data input modes, it is necessary to use just the *single letter* forms "H" and "W". The long forms are treated as data records.

As discussed previously, if data input is entered from the data editor, the return to the command editor is through the data editor. If data input is entered from command input, the return to the command editor is through command input.

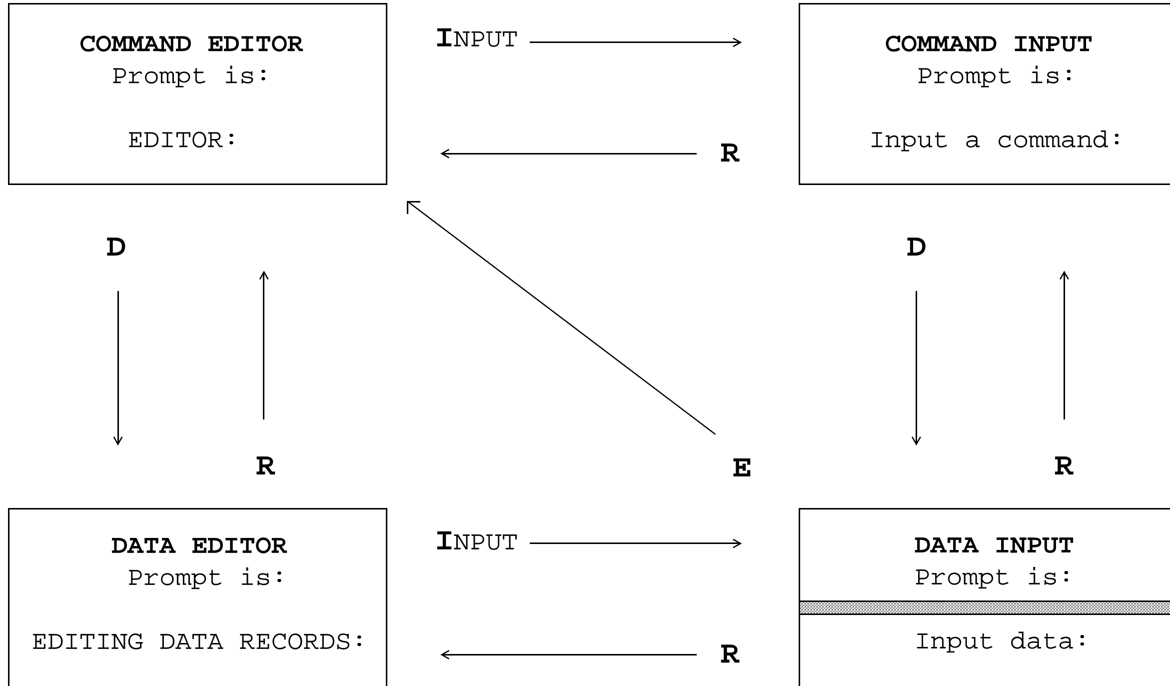
Figure 7.4 illustrates how to get back to data edit mode for error correction when data input mode is entered from command input. Here, a BUILD command that could create a file named MyFile is entered. When data records are complete, a null line is entered to indicate the end of the data. "R" for return is typed to move back to the main level of the editor — command edit mode. The data editor is then entered by using the DATA instruction. The record can now be corrected using the CHANGE instruction. If more data records are to be added, the user can go back into data input mode from the data editor. Once data entry is complete, the command can be executed from data edit mode (as shown) or directly from data input mode if desired.

If more commands are to be added when you are positioned in the data input mode, it is necessary to get back to command input. You can do this by first returning to the data editor, then moving to the command editor and finally moving to command input mode, or you can enter "E" to move directly from data input to the command editor. The chart in the summary section of this chapter shows the different paths about the P-STAT editor.

Figure 7.4 Correcting a Data Record

```
        EDITOR:
 1. >>  INPUT

        Entering command input mode:
        Input a command:
 2. >>  BUILD  MyFile,  VARS  Age  Sex ;

        Command completed.

        Input data:
 3. >>  44 2

        Input data:
 4. >>  24 1

        Input data:
 5. >>

        Response was end-of-file, or a totally blank line.
        Data input for this command is therefore completed.
        You can enter  D  to input more data,
        or R or X or H, or another command.
 6. >>  R

        Returning from command input mode to the main editor.
        EDITOR:
 7. >>  DATA

        This command has 2 data or subcommand records.
        The pointer is at the last one.
             2=24 1
        EDITING DATA RECORDS:

 8. >>  CHANGE /24/26
             2=26 1
        EDITING DATA RECORDS:
 9. >>  INPUT

        Input data:
        32 2
        Input data:
10. >>
        Response was end-of-file, or a totally blank line.

        Returning from data input mode to the
        data-subcommand part of the editor.

        EDITING DATA RECORDS:
11. >>  eXecute
```

SUMMARY



P-STAT Editor

The editor may be visualized as four separate "compartments", each of which has access to some of the other compartments, but not necessarily to all of them. Entry into the editor is always through the COMMAND EDITOR box. The DATA INPUT box has a "wall" across the middle so that the access is not circular.

The effect of the single letters "R" and "E" is identical except from DATA INPUT, where "E" gives control directly to the main or COMMAND EDITOR. "R" when used from DATA INPUT returns you to the DATA EDITOR if entry was from the DATA EDITOR. If entry was from COMMAND INPUT, "R" returns you to COMMAND INPUT.

In *input* mode all entries, except the special replies that return control to *edit* mode, are entered into the edit file. Corrections can be made only when edit mode is resumed.

Symbols
.RESPONSE. 2.12
.XINQUIRE. 2.18
A
ADD
- in NEW.EDIT.FILE command 3.5
- in OLD.EDIT.FILE command 3.5

AGAIN **3.3**, 5.4
- EDITOR instruction 6.8
- summary 3.7

B
BACKUP
- EDITOR instruction 3.5

BATCH 2.1, **4.1**
- error handling 4.2, 4.4
- output destination 4.2
- summary 4.6

BEGINNING
- EDITOR instruction 6.6

BOTTOM
- EDITOR instruction 6.8

BYPASS **2.13**
C
C, comment command 2.13
CHANGE
- EDITOR instruction 5.5, 6.5

CLOSE **2.5**, 2.8, 2.13
COMMAND
- EDITOR instruction 6.3

COMMAND.WIDTH 2.13
COPY
- EDITOR instruction 3.5, 6.18

D
DATA
- EDITOR instruction 6.3, 7.1

DEFAULT.SETTINGS **2.2**, 3.2
- identifiers
  - ECHO 2.2, 2.14
  - LINES 2.2, 2.14
  - OUTPUT.WIDTH 2.2, 2.14
  - PAGE.CHARACTER 2.2, 2.14
  - UNDERLINE 2.14
  - VBAR 2.2, 2.14
- summary 2.14

DELETE
- EDITOR instruction 5.5, 6.8, 7.5

DIALOG 2.10
- example 2.10

DOWN
- EDITOR instruction 6.8

E
ECHO **2.14**
- in DEFAULT.SETTINGS command 2.2
- in PRINTER.SETTINGS command 2.2

Edit file
- changing 3.4
- saving and restoring 6.15
- structure 5.4, 6.1

EDITOR **5.1**, 6.1, 6.21, 7.1
- building a run 7.2
- command 7.2
- full screen 5.4
  - insert text 5.7
  - replace text 5.7
- full screen mode 5.6
- input 7.1
- instructions
  - AGAIN 6.8
  - BACKUP 3.5
  - BEGINNING 6.6
  - BOTTOM 6.8
  - CHANGE 5.5, 6.5
  - COPY 3.5, 6.18
  - DATA 6.3
  - DELETE 5.5, 6.8, 7.5
  - DOWN 6.8
  - EDITOR 5.3, 6.2
  - END 6.6
  - EXECUTE 5.4, 6.6
  - EXPORT 6.16
  - FIRST 6.8, 7.6
  - GET 6.16
  - HELP 5.1, 7.1
  - IMPORT 6.16
  - INPUT 6.8, 7.1
  - KEEP 6.8
  - LAST 6.8
  - LOCATE 6.8
  - MOVE 6.18
  - NEXT 6.8
  - OLD 6.5
  - QUIT 5.4

REPLACE 6.20
RESTORE 3.5, 6.16
RETURN 7.6, 7.8
SQUEEZE 6.18
SYNTAX 6.8, 7.4
TOP 6.7, 7.4
TYPE 6.8, 7.6
UP 6.8
VALIDATE 6.20
WHERE 6.13, 7.1, 7.8
X 5.4, 6.6
Y1 6.15
Y2 6.15
Z full screen mode 5.6
interactive usage 5.1
summary 5.8
mixed case entries 5.7
moving the pointer 6.7
prompt messages 5.2, 7.2
single-letter controls 6.7
syntax checking 7.4
Z full screen mode 6.24
END 2.14
EDITOR instruction 6.6
Environment variable
PSTART 2.10
ERASE.EDIT.FILE **3.4**, 3.7
ERROR **2.15**
general identifier 2.6
ERROR.UNIT **2.15**
Errors
in batch run 4.2, 4.4
EXAMINE 1.8, 1.11
identifiers
EOC 1.11
EXECUTE
EDITOR instruction 5.4, 6.6
EXPORT
EDITOR instruction 6.16
F
FILES
summary 3.7
FIRST
EDITOR instruction 6.8, 7.6
FULL
general identifiers 2.6
Full screen editing 5.6
G
General identifiers
ERROR 2.6
FULL 2.6
IDEN 5.1
LINES 2.2, 2.6, 3.3
OUTPUT.WIDTH 2.2
OW 2.6
PAGE 2.6
PR 2.4
SHORT.16 2.6
SHORT.OLD 2.6
SHORT.TAGS 2.6
TAGS 2.6
TEMP 2.6, 3.4
TEXT 2.6
VERBOSITY (V) 2.6
GET
EDITOR instruction 6.16
H
HEAD **2.15**
HELP **2.15**, 4.6, 5.1
EDITOR instruction 5.1, 7.1
in batch runs 4.5
HOLDING message 3.2
I
IDEN
general identifier 2.20, 5.1
summary 3.9
IMPORT
EDITOR instruction 6.16
INPUT
EDITOR 7.1
location of pointer 7.2
EDITOR instruction 6.8, 7.1
Insert
text in full screen command editor 5.7
INTERACTIVE **2.1**, 3.7, 5.1
prompt messages 5.2
Interactive usage
EDITOR 5.1
summary 5.8
K
KEEP
EDITOR instruction 6.8

L
Large files
  processing in batch mode 4.4
LAST
  EDITOR instruction 6.8
LINES **2.6**, 2.16
  general identifier 2.6, 2.20, 3.3
  in DEFAULT.SETTINGS command 2.2
  in PRINTER.SETTINGS command 2.2
LOCATE
  EDITOR instruction 6.8
LONG
  in PROMPT command 5.3
M
MAXERROR 4.4, **4.6**
MOVE
  EDITOR instruction 6.18
N
NEW.EDIT.FILE **3.4**, 6.16
  identifiers
    ADD 3.4, 3.8
  summary 3.7
NEWS **2.16**
NEXT
  EDITOR instruction 6.8
NULL **2.16**
O
OLD
  EDITOR instruction 6.5
OLD.EDIT.FILE **3.4**, 6.16
  identifiers
    ADD 3.5, 3.8
  summary 3.8
OUTPUT.WIDTH **2.16**
  general identifier 2.20
  in DEFAULT.SETTINGS command 2.2
  in PRINTER.SETTINGS 2.2
OW
  general identifier 2.6
P
PAGE
  general identifier 2.6, 2.20
Page change character
  PAGE.CHARACTER 2.3
PAGE.CHARACTER **2.3**, 2.16
  in DEFAULT.SETTINGS 2.2
  in PRINTER.SETTINGS 2.2
PATCH 1.7, 1.11
  identifiers
    AFTER 1.8, 1.12
    AT 1.12
    AT END 1.8
    AT START 1.8
    OUT 1.7, 1.11, 1.12
    REMOVE 1.8, 1.12
PP
  see PRINTER.SETTINGS
PR **2.4**, 2.17
  general identifier 2.4, 2.20
PRINT **2.4**, 2.17
  printing a file 2.4
Print
  changing destination 2.4
  defining the destination 2.2
  destination 2.2
PRINT.INPUT 4.5, **4.6**
PRINTER.SETTINGS **2.2**, 2.8, 2.17, 3.2
  identifiers
    ECHO 2.2, 2.17
    LINES 2.2, 2.17
    OUTPUT.WIDTH 2.2, 2.17
    PAGE.CHARACTER 2.2, 2.17
    UNDERLINE 2.2, 2.18
    VBAR 2.2, 2.18
PROMPT **3.8**
  arguments
    LONG 5.3
    SHORT 5.3
  messages 5.2
PSAUTO 2.11
PSDATA 2.11
PSFILES 2.8, 2.11
PSTART 2.10
Q
Q2
  quit command 3.2
Q3
  quit macro 3.2
QUIT
  EDITOR instruction 5.4
  output 3.2

R
REFORMAT 1.12
  identifiers
    AFTER 1.7, 1.12
    ANYWHERE 1.7, 1.12
    BYTES 1.7, 1.12
    FIRST 1.7, 1.12
    INSERT 1.6, 1.12
    LAST 1.7, 1.12
    OUT 1.6, 1.12
    REMOVE 1.6, 1.13
    VALUES 1.7, 1.13
REPLACE
  EDITOR instruction 6.20
Replace
  text in full screen command editor 5.7
RESTORE
  EDITOR instruction 3.5, 6.16
RETURN 2.8, 3.1
  EDITOR instruction 7.6
  identifiers
    HOLD 2.9
  in transfer file 2.9
S
SCREEN **3.2**, 3.3, **3.8**, 3.8
SHORT.16
  general identifier 2.6
SHORT.OLD
  general identifier 2.6
SHORT.TAGS
  general identifier 2.6
SHOW **2.7**
  identifiers
    ACROSS 1.13
    COLUMNS 1.13
    NO BREAK 1.13
    NO HEAD 1.13
    SINGLE 1.13
    SORT 1.13
  summary 1.13
SHOW.ENV.VARS **2.18**
SHOW.KEY **2.18**
SHOWBYTES **1.4**, 1.14
  identifiers
    BREAK and NO BREAK 1.6, 1.14
    CHUNK 1.6, 1.14
    COLUMNS 1.6, 1.14
    FIND 1.5, 1.14
    FIRST 1.5, 1.14
    GAP 1.6, 1.14, 2.18
    HEAD and NO HEAD 1.14
    HEAD and NO HEAD, 1.6
    LAST 1.5, 1.14
    MAX 1.5, 1.14
    PREVIOUS 1.14
    ZERO 1.6, 1.14
SLEEP **2.18**
SQUEEZE
  EDITOR instruction 6.18
STATUS **3.9**
SUBSTITUTE.XL 2.8
SYNTAX
  EDITOR instruction 6.8, 7.4
SYSTEM **3.9**
SYSTEM.EDIT.FILE **3.4**
  summary 3.9
T
TAGS
  general identifier 2.6
TEMP
  general identifier 2.6, 3.4
    summary 3.9
TEXT **2.18**
  general identifier 2.6
TOP
  EDITOR instruction 6.7, 7.4
TRANSFER **2.8**, 2.19, 3.1
  identifiers
    EXIT 2.9, 2.19
TYPE
  EDITOR instruction 6.8, 7.6
U
UNDERLINE **2.19**
  in PRINT.SETTINGS command 2.2
UP
  EDITOR instruction 6.8
Utility commands
  interactive use 3.1
V
V
  see VERBOSITY
VALIDATE

EDITOR instruction 6.20
VBAR **2.3**, 2.19
in DEFAULT.SETTINGS command 2.2
in PRINTER.SETTINGS command 2.2
VERBOSITY **2.19**
general identifier 2.6
VERSION **2.19**
Vertical bar character 2.3
W
WHERE
EDITOR instruction 6.13, 7.1
Y
Y1
EDITOR instruction 6.15
Y2
EDITOR instruction 6.15