P-STAT®

Introductory

Manual

P-STAT®

P-STAT: Introductory Manual

January 2013

This publication corresponds to **P-STAT Version 3 Revision 3**, January 2013 This publication is designed to provide a general introduction to the P-STAT system as implemented under PC/Windows.

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

CONTENTS

Labels and Titles

LIST: Creating Listings and Reports

COUNTS: Frequencies and Descriptive Statistics

TURF

Simple Commands and Keywords

AUTOSAVE: P-STAT System Files

LIST OF FIGURES

1
P-STAT: Overview

P-STAT® is a computing system for data analysis and information management and display. An overview of its features highlights P-STAT's potential in a wide variety of applications. This manual describes P-STAT version 3 available November, 2012. There are some major differences between P-STAT version 2 and version 3.

On **PC/Windows** P-STAT version 3 can read the files created by version 2, but **P-STAT system files created or modified by version 3 CANNOT be read by a version 2 release**. The second major change which pertains to all new releases (PC and Linux) is the increased size for variable names from 16 to 64 characters. The new rules for legal variable names are:

- may contain letters, numbers, decimal points AND underscore characters (_)
- 1-64 characters long
- optional beginning tag of 1-16 characters terminated by a double colon (::)
- short versions of variable names (indicated by the use of a "?") are allowed in commands or subcommands

Each variable name must be unique and the optional tags, which serve as a nickname, must also be unique. Many commands use a shortened version of the 64-character names for labelling output. These have 16 characters or less.

While most of the features and commands found in P-STAT version 2 are available in version 3, a few have been dropped. If you find that a command you need is no longer available, please let us know and we will see if it can be included in a future release. Many commands have changes in their output which make use of the new variable names. For example: the SURVEY command has enhancements to the SPREADSHEET and CONTENTS output. The COUNTS output has a major enhancement in the layout of its output. The LIST command takes advantage of the enhanced variable names.

1.1 P-STAT'S CAPABILITIES

P-STAT capabilities include:

- retrieval of numeric and character information;
- file management;
- data modification;
- data display;
- statistical analyses;
- in-stream and block macros;
- interactive and batch processing.

1.2 Storage and Retrieval of Data

P-STAT accepts information in many different forms. Information may be *numeric*, as is average yearly rainfall or total automobile production, or it may be text or *character*, as is a name or an address. P-STAT accepts information from a variety of sources. The information may be formatted or unformatted. It may be produced by another software system. For example: tab delimited files from Excel and SPSS portable files, may be input to

P-STAT. Data taken from surveys on the Web, even those with unusual or extraneous characters, can also be cleaned up and readied for analysis with the MAKE and MAKE.FIXED commands.

P-STAT holds related information in a rectangular format called a “P-STAT system file”. The rows are often referred to as “cases” and the columns as “variables”. This file may contain both numeric and character information and is stored in a form that P-STAT can read, write and manipulate quickly. Information may be missing, and representations for three different types of missing data are available. P-STAT system files are *packed* to reduce storage requirements and *automatically saved* unless the file name begins with “WORK”. Once a file is changed, a *backup* version exists and is available in case the current version is lost or incorrectly modified. P-STAT system files can ONLY be read by other P-STAT commands. You cannot print a P-STAT system file. There are commands such as LIST which display the contents of a system file in printable or readable (text) format. The output from commands such as LIST, TEXTWRITER and SURVEY can be directed to a hard-copy printer.

MAKE and MAKE.FIXED are commands for entering raw data in almost any format to create a P-STAT system file. MAKE reads data in which the values are separated by delimiters (like commas). The input may be entered interactively, or it may be read from an external ascii input file. MAKE.FIXED reads fixed format data records from an external ascii file to create a P-STAT system file. Both of these commands permit you to change values in the input as data are read. Both commands have sensible controls for handling string values. The TEXTFILE.IN command can read a delimited ascii file, perhaps created by EXCEL, to create a P-STAT system file. The delimiter may be a tab character, a comma, or a blank.

Other types of files used by P-STAT may be made either within P-STAT or within the host computer operating system. They hold value labels, command sequences, edit files, and data resulting from modifications or analyses.

1.3 File Management

Many commands permit two or more files to be accessed simultaneously. There are commands to join and merge files with either *direct* or *hierarchical* linkages. These commands provide most of the capabilities normally associated with a relational database system. In addition, files can be sorted, divided into subsets, concatenated, interleaved, updated and transposed. Duplicate cases may be detected and deleted, or summarized into one case. There are also commands to do matrix operations and aggregation. A list of command names indicates the scope of file management capabilities available in P-STAT.

```
AGGREGATE     COLLATE      COMPARE     CONCAT
DUPLICATES    INTERLEAVE   JOIN        LOOKUP
MERGE         MODIFY       RANK        SEPARATE
SORT          UPDATE
```

A utility command, SHOWBYTES, is available to display the exact contents of an external file. Another utility command, REFORMAT, can be used to change fields in an external file. A simple example would be to change all tab characters to blanks. 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 is found. This is an excellent tool to use when a supposedly “clean” data file proves unreadable. The PATCH command is similar in spirit to REFORMAT.The PATCH 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.

1.4 Data Modification

The P-STAT Programming Language (PPL) is a powerful tool for *modifying* existing variables, *generating* new variables, and *selecting* cases or variables. PPL can be applied to any P-STAT system file as it is read by any P-STAT command. It can be used on the variables within a single case or across groups of related cases.

The ability to perform data modification across successive cases means that complex operations, which in most systems would require multiple passes through a file, can be done with a single pass using PPL. For instance,

given a file of people arranged in household order, it takes only a single step to calculate family size and append that information to the record of each member in that household.

Operators and functions include:

1. wildcards and masks in variable selection,
2. DO loops for repeated calculations,
3. logical testing with IF statements and IF/THEN/ELSE blocks,
4. recoding of data values,
5. system values such as .DATE., the current date, and a suite of functions to manipulate and compare dates. FORMAT.DATE is a function which provides a template for printing date/time information. See chapter 9 “Simple Commands and Keywords” for an example. Full documentation of the date functions is contained in “P-STAT: A Guide to the P-STAT Programming Language (PPL)”.
6. scratch variables and user-defined arrays permit cross-case and cross-command manipulations.
7. A case may be SPLIT into several cases or several cases can be COLLECTed into a single case..

Special functions for *character* variables permit calculation of the length and position of character strings, formation of substrings, conversion of character variables to numeric variables, and location and compression of specific characters. A single character variable may contain as many as 50,000 individual characters. This is enough to contain the contents of an open-ended survey question from even the most voluble employee.

One result of having variables with 64-character names is the need to be able to refer to them without always having to enter their full names. This is done in version 3 with wildcard matching, comparing full names and comparing tags. They are attempted in that order.

1.5 Data Display

Report writing is straightforward with the LIST and TITLES commands. Options exist for page layout, data formatting, subgrouping, hierarchical arrangement of variables, headings and summary statistics. Multiple top titles and bottom titles are possible. Simple lists are automatically formatted appropriately for the output device in use. Different output devices with varying attributes may be defined.

The TEXTWRITER command produces reports with precision formatting and complex layouts. In addition, information in P-STAT files may be tested and differentially incorporated in the reports. All PPL instructions and functions for data selection and modification may be used in TEXTWRITER. TEXTWRITER output can be used with PostScript controls to produce camera-ready copy with control over the page layout including choice of font and color.

The *crosstabulation* command, SURVEY, may be used to process both numeric and character questionnaire data. It produces stub and banner reports, with titles, multiple banners, nested banners, nets, and other features familiar to survey researchers. Extensive options provide for multiple response tables including nets and subtotals. Other options provide for cells with counts, different percentages, means and sums, flexible formats and layouts, and weighting. SURVEY supports PostScript printers and offers a variety of font choices when PostScript is available.

Auxiliary commands for the market research analyst include:

1. TURF: a command to do Total Unduplicated Reach and Frequency analysis. TURF.SCORES a tool to analyze the TURF output and identify the subjects who contributed most to the TURF results.
2. BALANCE: a command which uses sample balancing techniques to produce weights which have the least impact on cell counts.
3. SAMPLE: a command to produce a random sample from a larger file. By variables can be supplied to assure that the sample matches the larger file in critical ways.

1.6 Statistical Analyses

P-STAT provides a wide range of statistical procedures from simple descriptive statistics to complex multivariate procedures. *Description files* give means, standard deviations, highs, lows, and counts of good and missing data items. The COUNTS command provides an initial overview of data. It produces frequencies and percentages of *all unique values* of character and numeric variables, as well as a multitude of summary statistics. COUNTS can also be used to produce frequency distributions for subgroups. COUNTS output can optionally be saved in a P-STAT system file. The PERCENTILES command provides medians, quartiles, deciles, centiles or arbitrarily specified *quantiles*.

Groups of independent or paired data may be tested using *t tests*. When the measurements are not qualitative data, *nonparametric* tests may be used. *Correlations* for continuous, dichotomous, and ranked variable pairs may be requested, as well as significance values and corrections for grouping errors. Canonical correlation is a multivariate procedure that analyzes the relationship between two sets or groups of variables.

Multiple *regression* analysis can fit linear equations to observed data values, predict dependent values and calculate residuals. The analysis may be a forward stepwise regression, with stepping either program or user controlled, or it may be non-stepwise.

Analysis of variance (ANOVA) determines the effects of multiple treatments in both balanced and unbalanced designs, and detects and reports confounding of effects. Designs may be modified interactively, and error terms and blocks specified. Factorial, block, nested, split-plot and repeated measures designs may be analyzed. Covariates and partitioning of sums of squares may be specified with a cohesive algebraic notation.

Survival analysis is used to analyze lifetime (survival time) data. Both the lifetable (actuarial) and product limit methods are supported. *Cluster analysis* puts cases into a specified number of groups. The cases may be weighted. *Discriminant and factor analyses* permit identification of significant groups and factors. Various factor matrix rotations may be requested. In addition, *matrix* operator commands and a *macro* facility, used in conjunction with the programming language, permit a user to define statistical procedures not included in the P-STAT system.

1.7 USER FEATURES

P-STAT user features include:

- same command language in both interactive and batch modes;
- in-stream and block macros for frequently used activities;
- ease of use with menus on PC/Windows,
- English-like language;
- available on both PC/Windows and Linux.

1.8 Version Differences

There are two areas of difference between versions of P-STAT. On all supported computers P-STAT is available in six different sizes: Whopper 2 - Whopper 7. Whopper 2, the basic size permits a single file to contain 6,000 variables. Whopper 7 can handle a file with 250,000 variables. The number of cases depends only on disk storage space. Various work areas also increase as needed to support the additional variables. The Whopper 2 size is adequate for everyone except

1. those with VERY large data files
2. those who make extensive use of the P-STAT programming language (PPL).

The major difference between P-STAT versions for PC/windows and versions for Unix based operating systems is the availability of front-end menus on PC/Windows. These menus are designed to provide an easy solution for relatively simple problems such as TURF (Total Unduplicated Reach and Frequency), file management, and basic statistics. The menus can also be used for simple PPL such as keeping or dropping variables.

1.9 Interactive and Batch Computing

The P-STAT system can be used in either *interactive* or *batch* mode; the commands are identical in both modes. In interactive mode, P-STAT provides immediate error correction. Simple errors or ambiguous situations are handled with a question and reply, or a request to retype a subcommand. *Interactive* computing describes an environment where each step is entered and immediately executed. Then, depending on the results of that step, the next step is entered.

Batch computing describes an environment in which the entire run is prepared and submitted to the computer for execution. When the run is complete, the results are printed or returned to the terminal. When errors are found, the commands or data in the run must be corrected and resubmitted for execution. A job is a batch job if the entire job is sent to the computer and the results come back later. This is true even if a terminal is used for entering the job and the time it takes to receive the results is only a few minutes.

P-STAT is a c*ommand driven* system, with menu and query options available for PC/Windows. The menu system, where available, helps the novice or occasional user perform analyses that do not require extensive use of the programming language. It lets them select desired procedures, it prompts for required filenames, and it presents lists of options to choose among. The command that results from the selections is displayed and then executed. Many P-STAT users learn to write commands as they observe those composed with menus.

Usually the way that the P-STAT module is invoked determines whether this is a batch run with all the commands stacked up and ready to go or an interactive run with a user at his terminal directing the show. In general, if you invoke the P-STAT module from your terminal, the assumption is that you are there and in control. If you invoke the P-STAT module and supply command line information providing names for both the input and output files, P-STAT assumes that this is a BATCH run.

```
Unix:          p-stat    <"input.file"   >"output.file" &
Windows   run  pstat.exe  "input.file"   "output.file"
```

The "&" at the end of the Unix command line causes P-STAT to run as a background job. On a windows machine you can also invoke P-STAT from a DOS command window. If the PSTAT.EXE module is not in your path you will need to provide the full pathname. For example:

```
c:\program files\pstat\pstat.exe  "input.file"  "output.file"
```

This can be changed by editing the system environment variable PATH to point to the folder where the PSTAT.EXE is stored. The quotes are required if the file names contain blanks.

The commands BATCH$ and INTERACTIVE $ are available but are usually unnecessary. When you invoke P-STAT without arguments from your terminal, the window contains the following information

---

P-STAT, version 3.01, rev 2 (Oct 12, 2012)
WHOPPER 2 (6,000 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... 15:58:56 Aug 12, 2012

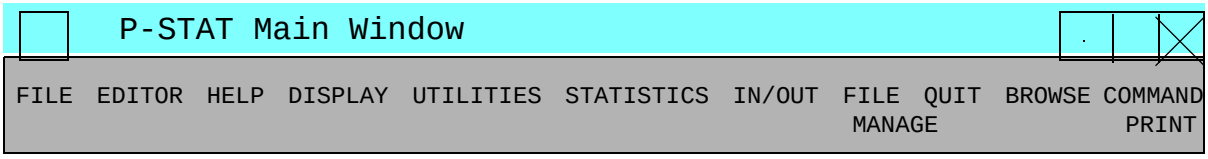
Enter a command:

The line "Enter a command:" indicates that you are running interactively and that P-STAT is ready to execute your commands.

Even if you are running a version of P-STAT with front end menus, you may find that you need to enter a command that is not supported by the windows system. P-STAT expects to find command input in single lines that are **no longer than 80 characters**. A command can be entered on multiple lines (separated by the carriage-return character) until the the end of command is signalled by a single "$". Do not enter the carriage return in the middle of a name or token. If you are entering the command from the P-STAT main window, it is easy to see

if you are approaching the 80 character limit in time to press the enter key and move to the next line. If you are working from a text editor, ***turn off the word wrap feature which automatically moves the cursor to the next line and work from a text window that is set to a width of 80 characters.***

The menu system is designed especially for Intel based PC computers running some form of Windows such as XP or Windows 7 or 8. This has a two line top menu with pull down menus to support the basic commands. Use of these menus is described in the manual "P-STAT: An Introductory Guide for Windows Users".



1.10 Macros

P-STAT macros contain P-STAT instructions or data. They can be used anywhere to hold any part of a P-STAT command stream. However, they usually hold pieces of P-STAT code or command sequences that are frequently used. All macros can have arguments which may or may not have default values. When a macro is used any values that do not have default definitions must be supplied. Macros can call macros which can then call other macros. There are two basic types of macros: block macros and in-stream macros.

Block macros contain one or more P-STAT commands. When the macro is executed the commands in the macro are executed one after the other. Special features of macros that are not available in a normal job stream are:

1. SUBFILES which provide the ability to execute the commands in the macro for each subgroup specified by the SUBFILES command;

2. Logic which permits you not only to skip commands but also to loop back and re-execute earlier commands.

In-stream macros are not complete commands but can be any part of the command except the command name itself. They can appear within the command text, within programming language statements, and within subcommunity or data records. Like block macros, in-stream macros can have key word or positional arguments. The arguments must have values supplied either as defaults in the definition or as argument values when the macro is used.

Many regular P-STAT users use the *command language* once they are past the learning stage. The free-format language is composed of a command name followed by keywords giving optional parameters. The command names and options are mnemonic and therefore easy to remember. Error messages are textual and clearly describe any problems. It is not necessary to know a computer language or much about your local operating system to use P-STAT successfully.

1.11 LEARNING TO USE P-STAT

It is not difficult to learn the basics of P-STAT despite the fact that it is a large system with extensive capabilities. There are two introductory manuals:

1. "P-STAT: Introductory Manual" provides an overview of the P-STAT system and details on the commands and concepts that most users will need.

2. "P-STAT: An Introductory Guide to P-STAT for PC/Windows" introduces the menu system. The menu system is designed to get a user started and perform many basic operations including data entry, simple file management, and some statistics (including TURF). This augments but does not replace the basic introductory manual.

1.12 P-STAT Documentation

The following manuals describing the P-STAT system are currently available and can be downloaded from our web page (www.pstat.com0 in Acrobat format.

1. P-STAT: Introductory Manual (psintro.zip)
2. P-STAT: An Introductory Guide for Windows Users (pcintro.zip)
3. P-STAT: File Management (psmang.zip)
4. P-STAT: A Guide to the P-STAT Programming Language (PPL) (psppl.zip) This includes TEXT-WRITER and MACROS
5. P-STAT: The SURVEY, BALANCE and SAMPLE Commands (pssurv.zip)
6. P-STAT: Plots, Graphs and PostScript Support (psplot.zip)
7. P-STAT: Utility Commands (psutil.zip) . This includes the features of the P-STAT editor
8. P-STAT: Basic Statistics (psstat1.zip)
9. P-STAT: Advanced Statistics (psstat2.zip)
10. P-STAT: Master Index (pstatIX.zip)

1.13 INSTALLATION

The actual process of installation depends on the computer system. Each CD is sent with instructions that are appropriate for that system. In general:

1. PC/Windows: The PSinstall program controls installation and prompts for information about the disk drive and the directory where the module should be placed. PSinstall creates or updates a PSTAT.INI file on the system windows directory. P-STAT reads this file for information such as the location of the P-STAT help file. An auxiliary program PSTATINI can be used to add to or change the information stored in this file.

   The PSinstall program creates a windows group with a P-STAT shortcut that you can click to start a P-STAT run.

2. Unix: modules supplied in tar format on CD. The installation is done from the directory where the module is to reside. Environment variables should be placed in your .profile or .cshrc file.

1.14 Environment Variables

On both Windows and Unix environments there are several P-STAT specific environment variables which make it easier to locate data sets. Most of the information that is contained in these variables can be supplied or changed by P-STAT commands. The use of these commands is covered in full in the chapter “AUTOSAVE: P-STAT System Files”. The only environment variables that have no equivalent command form are PSKEY and PSTART.

1. PSKEY The PSKEY is not required for the basic demo version. The demo version permits all P-STAT commands to be used with a reduced file size. Your PSKEY is supplied when you order a full version of P-STAT. This must be set either in the PSTAT.INI file (PC/Windows) or in one of the .profile, .cshrc or .bashrc Unix files.
2. PSHELP supplies the name and location of the P-STAT help file. This is necessary if the on-line help file is to be used. Since this location is determined at installation time, it makes sense to set it once with an environment variable and forget about it. On the PC, this information is automatically stored in PSTAT.INI by the installation program.

3. PSFILES supplies the drive and path current working directory (folder). This is where P-STAT expects to find or create all files except those explicitly located elsewhere. The assumed location depends on the operating system. It is prudent to issue this command as the first command in your P-STAT run.

   There are three types of files to be located: 1) P-STAT system files (PSAUTO) created by P-STAT commands, 2) temporary work files (PSTEMP) used by P-STAT commands such as SORT, and 3) external data files (PSDATA) including your data input, labels files, command files and files exported to and from other systems.

   PSFILES sets values for PSAUTO, PSDATA and PSTEMP. PSAUTO, PSDATA and PSTEMP, if used, take precedence over the PSFILES setting.

4. PSAUTO provides the full path to a directory where P-STAT system files are located. If PSAUTO is not specified, system files will be written to and read from the current directory. What determines the current directory depends on the operating system.

5. PSTEMP proves the full path to a directory where temporary files are to be placed.

6. PSDATA provides the full path to the directory where other external files are located. This includes raw data files, command files, value label files and files to be exported or imported into spreadsheets or other packages.

7. PSTART provides the full path and the name of a file which contains P-STAT commands to be executed as the run begins. The PSTART file may contain any P-STAT command including PSHELP. A command in the PSTART file takes precedence over the equivalent environment variable.

   The PSTART environment variable is not automatically set by the Windows INSTALL program. If the PSTART environment variable is set and points to an existing file, the commands in that file will always be executed at the beginning of a P-STAT session.

   If there is no PSTART environment variable, P-STAT looks for a file named either PSTART or pstart in the PSDATA directory. If the PSDATA directory has not been specified by either a PSFILES or PSDATA environment variable, P-STAT looks for PSTART or pstart in the current working directory. PSTART is fully described in the manual “P-STAT: Utility Commands”.

1.15 Examples of Three Different File Types

The three different file types are: 1) P-STAT system files created by P-STAT commands; 2) External raw data files including label files and files containing P-STAT commands; 3) temporary files which are used by individual commands to hold intermediate results. If all your files for a project are in the same folder, use the PSFILES command. If your files are in different folders, the following environment variables can be set:

```
1. PSAUTO   'C:\olddata' $
2. PSDATA   'D:\input' $
3. PSTEMP   'C:\tempdir' $
```

The command to create a P-STAT system file named “X” from data stored in a disk file named “ABC”:

```
MAKE X, NV 2, FILE 'ABC' $
```

uses the following files:

1. **X** is the name given by the user for the P-STAT system file (an autosave file) which is created by the MAKE command. It is written to the PSAUTO directory as:

```
C:\olddata\X.PS1
```

Note: P-STAT uses extensions of PS1 and PS2 on system file names. The pathname and the extensions are NEVER used in P-STAT commands. File X.PS1 is always referred to as “X”.

2. **ABC** is an external file being read. When it does not have its own path it defaults to a PSDATA path and is read as:
   ```
   D:\input\ABC
   ```

   Programs such as MAKE also create and then erase several temporary files which have unique names. Given the above PSTEMP, a name for such a file on PC/Windows would be something like
   ```
   C:\tempdir\P_0Z7CR29J2$$$$.TMP
   ```

1.16 MIGRATE

On PC windows P-STAT Version 3 cannot read binary files created by P-STAT Version 2. This does not matter with P-STAT system files which are automatically converted to Version 3. However, P-STAT edit files are NOT automatically converted. The new MIGRATE command is available on the PC only and does the conversion for you.

```
MIGRATE "c:\myfile\test.edt", OUT "c:\newstuff\test.edt" $
```

If you do not know the origin of your P-STAT files, use the FILETYPE command.

```
FILETYPE file.PS1$
```

This examines a file and simply reports if the file is LF90 unformatted, LF95 unformatted, CR-LF formatted, or unknown. LF90 indicates that the file was created by P-STAT version 2.

SUMMARY

Environment variables may be created which tell P-STAT where it should find existing files or write new files. On PC/windows the PSINSTALL program will create the environment variables for you. They are stored in a file named PSTAT.INI which is stored in the directory associated with the system environment variable WINDIR. This is usually C:\WINDOWS or C:\WINNT. If you do not have write access to the windir folder you can set an environment variable named "PSTATDIR" to an alternate folder where you do have write access. If "PSTATDIR" is set, the PSTAT.INI file will be written to that folder. The PSTAT.INI file can be edited in an ordinary text editor or updated with PSTATINI a program that is provided when P-STAT is installed.

PSKEY and PSTART cannot be used as a commands in the P-STAT program. The other environment variables can be run as P-STAT commands. Note: if they are used as commands the argument that follows must be enclosed in quotes. On the PC the command to create or change an environment variable is "SET".

PSKEY key value

PSKEY is required for all except demo installations.

PSTART full path and file name

PSTART provides the complete path and name of a *start-up* file of P-STAT initialization commands.

```
PSTART=d:\joe\psinit.trn $
```

If the PSTART environment variable exists, the contents of the file it cites will be executed at the beginning of any P-STAT session.

PSHELP full path and file name

PSHELP provides the complete path and name of the PSTAT help file. The following assumes that the P-STAT module is on the C drive in the PSTAT directory.

```
SET PSHELP=C:\PSTAT\PSTATHELP.DAT
```

The P-STAT helpfile is also available in HTML and can be viewed from your browser. On the PC, this form of the help file is available from the menus. The html helpfile is located in a folder (helpfiles) under the folder where P-STAT is installed. For example: if P-STAT is installed in /usr/local/bin/pstat the entry to the helpfile is "/usr/local/bin/pstat/helpfiles/index.html".

PSFILES full path

PSFILES provides the complete path to a directory for PSAUTO, PSTEMP and PSDATA if they are not individually specified.

```
SET PSFILES=C:\susie\study32
```

PSAUTO full path

PSAUTO provides the complete path to a directory where P-STAT system files are located and new system files are to be written. It takes precedence over the PSFILES directory.

```
SET PSAUTO=C:\susie\study32\mypsf
```

PSDATA full path

PSDATA provides the complete path to a directory where external data files are located.

```
SET PSDATA=E:\proj32\data
```

PSTEMP full path

PSTEMP provides the complete path to a directory where temporary work files are stored. These files are automatically erased when P-STAT runs are terminated with an END$ command.

```
SET PSTEMP=C:\TEMP
```

BATCH $

Indicates that this is a batch run. The input will be follow in the input file. The output will be written to the output file supplied on the command line.

INTERACTIVE $

Indicates that this is an interactive run. The input will come from the terminal and the output will be written either to the terminal or to a supplied print output file.

MIGRATE

The MIGRATE command is only available on PC/Windows.

MIGRATE, INFILE OldFile.edt, OUTFILE NewFile.edt$

INFILE 'fn'

the input file was written by P-STAT version 2.

OUTFILE 'fn'

and should be written out in the appropriate way for P-STAT version 3. This command is only needed when running PC/Windows.

FILETYPE

The filetype command is only available on PC/Windows.

FILETYPE 'abcd.edt' $

reports the source of the specified file. The possibilities are formatted CR/LF terminated or unformated written by Lahey Fortran 90 or Lahey Fortran 95.

2
Introduction To P-STAT

This chapter covers the following topics:

1. The basic language used in discussing files and data.
2. Initiating a P-STAT session.
3. Creating a P-STAT file and listing that file.
4. Accessing that file in another P-STAT session.

2.1 ORGANIZATION OF INFORMATION

The word “file” refers to a named collection of information. The word “file”, by itself, tells us nothing about the contents. There are just two properties that are common to all files: they have a name and a location. When you are using P-STAT you will encounter several different types of file.

1. Raw data files are the numbers and characters that you wish to process.
2. P-STAT system files are created by P-STAT commands from the raw data.
3. Label files are used to enhance tables and reports by supplying text associated with the data values.
4. Command files contain instructions for P-STAT to execute.
5. Print files contain the tables and reports created by P-STAT commands.

In the examples in this book upper and lower case letters indicate usage.

1. UPPER CASE is used for keywords and command names.
2. Mixed and lower case is used for names where the choice is yours.

In general enter a new name as you would like it to appear in printout. In subsequent use, enter the name in the most convenient way. The only exception is for external files when you are running under Unix/Linux where case matters.

2.2 Raw Data Files

P-STAT can handle both numeric and character data from many different sources. The simplest form is the spreadsheet which is arrayed in rows and columns and stored on a disk as a text file. Data that is organized as a spreadsheet can usually be processed with little effort using the P-STAT TEXTFILE.IN command.

A stream of data taken from a questionnaire may require more detailed information to describe the fields. Data from a questionnaire often needs specific information about each question that has been asked. The MAKE command in P-STAT can easily handle complex arrangements. If the data are taken from a questionnaire on the web, it is often necessary to weed out extraneous characters from the data fields. Utility commands in P-STAT are very useful for this type of “dirty” data.

In an extreme case, where the data must be teased out of the input records, an entire record can be defined as a single large (up to 50,000 characters) character variable. The P-STAT Programming Language (PPL) is then used to locate the relevant information. The availability of many character functions make it possible to manipulate strings and sub-strings. If the data have not yet been keyed into a file, they can be entered directly into P-STAT using the MAKE command.

When the data are stored in the file, P-STAT needs to know where that file is located. Particularly in PC/Windows it is important to use the PSFILES command at the start of the run to provide a path to the current folder/directory. P-STAT is a command driven system. Even when using menus in PC/Windows, your replies are used to create the commands that are executed. The PSFILES command looks like:

```
PSFILES 'C:\projects\sept09' $              or in Unix
PSFILES '/export/home/projects/sept09' $
```

When the PSFILES command is executed, P-STAT knows where it should go to find input data and where to put the results. If you have files that are located in other folders, you can reference them by supplying the full drive/path/filename information in quotes.

2.3 Missing Data

Sometimes information is missing. This can happen in a survey when a person refuses to answer a question or in medical research when a patient misses a visit. Missing data are permitted for both numeric and character variables.

Three different missing values — missing type 1, missing type 2, and missing type 3 — may be defined for each variable. This makes it possible to distinguish, for example, between the person who was not asked the question, the person who refused to answer, and the person who replied, “I don’t know.” Values may be defined as missing either when the file is built or when it is being modified in some way. The P-STAT Programming Language (PPL) provides extensive ways of testing and modifying both good and missing values.

Each of the three types of missing data is converted into a different, very large negative number. These numbers are not likely to occur in normal data. This representation of missing values makes it easy for P-STAT to detect missing values when computations are performed. In P-STAT, if a value in a computation is missing, the result is automatically missing.

2.4 P-STAT System Files

P-STAT commands use the raw data files as input and create output files in a binary format that it can process rapidly. These files cannot be printed directly on the terminal or a printer. Most P-STAT commands either produce new P-STAT system files and/or require one or more P-STAT system files as input.

P-STAT system files store file names, variable names, and a notation of the command which created them, as well as the data values. P-STAT system file names can be up to 16 characters long and can contain only letters, numbers and decimal points. The first character MUST be a letter. Case is unimportant. P-STAT remembers the file name as it is first entered and uses that representation in all reports. Whatever, the name of a P-STAT system file, it is ALWAYS stored on the disk in upper case characters with a 3 character extension. The default extension is “PS” followed by either a 1 or a 2.

```
PSFILES 'C:\projects\sept09' $
TEXTFILE.IN  Test.0909,  FILE 'mydata.txt'  $
```

These two commands use a raw data file with the full name “C:\projects\sept09\mydata.txt” to create a P-STAT system file named“Test.0909”. The TEXTFILE.IN command, without further information, makes 2 assumptions: 1) that the first row of data contains names for the variables; and 2) that the data values are separated by a TAB character. Because of the PSFILES information, the P-STAT system file is stored on disk as:

```
C:\projects\sept09\TEST.0909.PS1
```

Once the file is created you can refer to it as:

```
test.0909   or  TeSt.0909   or    TEST.0909 ......
```

The file extensions PS1 and PS2 are used ONLY by the P-STAT system. When a file is created it gets an extension of “PS1”. When the PS1 file gets modified or a new file with the same name is created, it gets the extension “PS2”. A third use of the file name “test.0909” will have an extension of “PS1”, etc. Thus there is always one level of backup. All system files have an internal counter so that P-STAT can determine whether the .PS1 or

the .PS2 file is more recent. Note: This convenience is a 2-edged sword. It is very useful for test runs but, if you reuse the name of an important file and make a mistake, you run the risk of losing your data. If you do reuse the name of an important data file, pay attention to error messages and end the run before a damaged file replaces your backup.

The basic unit handled by the P-STAT system is a single piece of information, the variable. This may be a *number* or a *character string*. If it is a number, it may be an integer or a real number of any size. All numeric data in P-STAT is carried in double precision for maximum accuracy. If the value is a character string, it may be anywhere from 1 to 50,000 characters in length. If the length is not specified, 40 characters is assumed.

Each variable has a unique name which may be up to 64 characters and may contain only letters, numbers, decimal points, and underscores. The use of a double colon (::) is allowed to divide the variable name into 2 parts: the optional first part (up to 16 characters) is known as the "tag", followed by the double colon (::) and finally additional text for a maximum of 64 characters. Not only must the variable have a unique name, tags, if they are used, must also be unique. If you do not provide a name, P-STAT generates a name in the form of "varN" where N is the variable number.

One result of having variables with 64-character names is the need to be able to refer to them without always having to enter their full names. The order in which the 4 ways of name matching is done in version 3 are:

1. wildcard matching: ?abc or abc? or even ab?c
2. comparing full names,
3. comparing tags

The use of variable names, tags, variable positions and wildcard matches is covered in detail in the manual: "P-STAT: A Guide to the P-STAT Programming Language (PPL)".

2.5 P-STAT Labels Files

Labels files are used to provide value labels for use by commands such as LIST and SURVEY. These labels are stored in text files which are easy to read and easy to edit. They can also be used to provide alternate variable names. File TestLab.txt might contain:

```
/* Labels files may contain comments /*
Var1 'Id number' /
VAR2 'Age Group'  /
VAR3 'How many years of schooling have you completed?'
     'Asked only of those over age 20'
     (1) High School (2) Some College (3) College or more /
var4 'Age of oldest child ' /
var5 'General occupation information' /

/* extended variable names and value labels can be ïntermixed or
entered separately */

var2 var4 (1) Under 15 (2) 15-24 (3) 25-34 (4) 35 and over /
var5 (1) financial (2) medical (3) academic (4) service /
```

Variable names are case independent. A well constructed and organized labels file provides a valuable record of the data in the associated P-STAT system file.

2.6 P-STAT Command Files

P-STAT can be run directly from the P-STAT window on your screen. This is interactive usage. Each command is either built up in the menu system on PC/Windows or entered from the command line at the bottom of the P-STAT window. In interactive usage all printout appears on your terminal unless you direct it elsewhere.In "batch" computing the input is a text file of P-STAT commands and the output is directed to a disk file.

The advantage of interactive computing is that you can see and correct problems, one step at time. You can also change the run as you see the results on the screen. Batch computing is good for jobs that require complex PPL (P-STAT Programming Language) and for jobs that will be processing large volumes of data or require thousands of iterations.

P-STAT commands always begin with a P-STAT command name and always end with a dollar sign ($). They can be freely entered on multiple records. *Turn word-wrap off in your editor if you are creating a P-STAT command file or press the carriage return key at the end of each line.* P-STAT expects 80 character input records.

```
PSFILES 'C:\projects\sept09'
PR 'test.0909.prn' $
TEXTFILE.IN  Test.0909,
            FILE 'mydata.txt'  $
LIST Test.0909, LABELS 'test0909.lab' $
END $
```

2.7 P-STAT Print Files

The output from a P-STAT command is either one or more new P-STAT system files or text. The text may simply be a statement that the command is complete; a report of what was done; or nicely formatted values in a list or a table. In a batch run this printout is written to an output file on disk. In an interactive run this output appears on the screen. In either case you can control the output destination and provide a layout that can be stored on disk and sent to a hard copy printer.

There are several ways to define the attributes of the print file and there is a command to send that file to the real printer. These will be discussed in more detail later in this manual. In the example above the file named "Test.0909" is saved as a text file named "test.0909.prn", It is stored in the folder named in the PSFILES command and can be read, edited or sent to the local printer.

2.8 A SIMPLE RUN

The first step in using P-STAT is getting the P-STAT computer package running. Usually you will run P-STAT from a folder/directory other than the location where P-STAT is installed. You need to know where P-STAT is stored. On Unix/Linux, if that location is in your path, you can run it from any directory by simply entering "p-stat", On PC/Windows you will find in the "START/All Programs" menu an entry for "PSTAT" which contains 2 shortcuts. Select the one labelled "PSTAT". ("PSTATINI" is used to set environment variables.)

If the initial greeting message:

```
P-STAT, version ....
```

does not appear, see the local person who installed P-STAT on your system or refer to the installation instructions that came with the software.

The next step in using P-STAT is to create one or more P-STAT *system files* from your data. It is necessary to write P-STAT instructions in the P-STAT command language. If the MENU system is available for your computer, it can be used to generate the commands for you. The manual "P-STAT: An Introductory Guide for Windows Users" provides documentation on the use of the menu system. The commands that are created in the menu system are identical to those entered from the command line.

The P-STAT command language has its own vocabulary, syntax and punctuation rules. The vocabulary is English-like, with command names that describe the functions, such as MAKE, LIST and END. The syntax and punctuation, while somewhat rigid, are not complex.

In the P-STAT documentation, commands that the user enters and output that P-STAT produces are shown in:

```
monofont.
```

Words that must be entered exactly as shown, such as command names and options, are written in upper case characters.

```
MAKE  LIST  END
```

Words that are your choice, such as file and variable names, are in mixed or lower case:

```
PaceSet.
```

Either case may be used for commands and names input to P-STAT — most people use lower case because it is easier to type. (UNIX users and others with operating systems that differentiate between case in the name of files must use the correct case in referencing files on disk.)

2.9 Command Structure and Punctuation

A P-STAT command begins with a command name that is known to the system, and it always ends with a dollar sign ($). This is a simple command:

```
END  $
```

Its purpose is to end a P-STAT run and close any open files. END must be followed by a "$", which is the punctuation necessary to finish a command. Each command begins on a new line but may continue on subsequent lines. Do NOT continue typing continuously across lines. Press the return key when a line is complete and then continue typing. Remember P-STAT expects the input in **80 character records.**

P-STAT's *command prompt* is:

```
Enter a command:
```

It tells you that P-STAT is ready for you to type a command. Sometimes, when P-STAT is processing a command, the message:

```
HOLDING .
```

appears near the bottom of the terminal screen. That means that P-STAT is holding the rest of its output to give you time to read what is already on the screen. Press the enter or return key to continue seeing the rest of the output or press "Q" to quit seeing output. You cannot enter a command unless the P-STAT command prompt is on the screen.

A command can have three sections. Each section begins on a new record. A command always has the first section and may have sections two and/or three. The sections are:

1. the command name and optional phrases which are separated by commas. The syntax of this section is common to all commands. It ends with a semicolon if another section follows, or with a currency symbol ($) if the command is complete. For example, END $.
2. subcommands, which have a syntax unique to the particular command. This section also ends with a $ if the command is now complete. If the subcommand section is in fact followed by raw data, a semicolon (;); would, in the case of MAKE, announce that the subcommands are done and raw data follows.
3. raw data records ending with a $.

2.10 Making a P-STAT System File From Your Data

There are several commands which can be used to process your data and make a P-STAT system file. The choice depends on where and how the data is stored and on the complexity of the data itself. If the data values have not yet been stored in a computer, the choices are different than if the data are already available in a disk file.

The MAKE command, which is fully described later in this manual can be used to enter the raw, unprocessed data directly.

1. The first step is to provide information about the variables.

2. The second step is to enter the data values

If your data are already stored on the computer, your choice depends on the nature and structure of the data. The simplest case is numeric or character data with delimiters between each data value and between each case. This will often be produced by a spreadsheet such as Excel. The first record may contain labels for each of the variables. The TEXTFILE.IN command is designed to handle this kind of data. TEXTFILE.IN looks at the data to determine the data type for each variable. If the first record, does not have variable names, you must specify 'NO LABELS' . If the variable names are present but do not conform to P-STAT's rules for legal variable names, TEXTFILE.IN does what is necessary to make them legal.

Figure 2.1 Know Your Data

This example is **comma delimited** and **TEXTFILE.IN** will work.
The variables will be named var1 through var5.

```
Allen,Robert,m,,4.5
Johnson,Karen,f,13,3.0
Prince,Dorothy,f,14,3.5

TEXTFILE.IN work1, DELIMITER COMMA, NO LABELS,
            FILE 'tennis.csv' $
```

This **free format** example can be processed by the **MAKE** command

```
Allen   Robert  m - 4.5
Johnson Karen f 13 3.0
Prince Dorothy f 14 3.5

MAKE work2, FILE 'tennis.txt', DELIMITER blank;
Last.name:c First.name:c gender:c age USTA.rating $
```

This example is **fixed format** and **MAKE.FIXED** is appropriate

```
Allen   Robert  m    4.5
Johnson Karen   f 13 3.0
Prince  Dorothy f 14 3.5

 MAKE.FIXED work3,  FILE 'tennis.fix';
   Last.name:c8 First.Name:C9 gender:c1 age 19-20 USTA.rating 22-24
```

The MAKE command is more powerful than TEXTFILE.IN but requires more information from the user. The MAKE.FIXED command is far more complex but it can handle data files that have been collected on-line. When users fill out online questions there is little control over the characters that they type. A special feature of MAKE.FIXED permits the final field to be a character variable of arbitrary length.

Figure 2.1 illustrates the use of the TEXTFILE.IN and the MAKE commands to process a simple data set. All three of these commands are discussed in detail later in this manual and in the manual "P-STAT: File Management".

MAKE can be used on any machine that P-STAT supports and it can be used in both interactive and batch runs. When it is used interactively, the mode is one of query and response rather than full screen. MAKE is more powerful than TEXTFILE.IN and can handle many messy data situations.

MAKE makes a P-STAT *system file* from raw data. The raw data can be read from an existing file or can immediately follow the MAKE command and subcommand information. Once you have a system file, you need not deal with the raw data anymore. System files can be used in all other P-STAT commands, both today and in future P-STAT sessions.

Figure 2.2 Creating a File Interactively From External Data

```
    The prompts, the command and the report

    Enter a command:
>>  PSFILES "C:\projects\tests" $

    Enter a command:
>>  MAKE Tennis,  FILE 'tennis.txt';

    Definitions (i.e., the names and types)
    of the variables are expected now.
    A response of just H or Q, at any time,
    will provide HELP, or will QUIT the command.

    The definitions should end with a dollar.

    Begin entering variable names,
    of  H ( for HELP) or Q (for Quit):

>>  Last.Name:c  First.Name:c  Sex:c  Age  USTA.Rating $

    -----------------MAKE completed-----------------
    | P-STAT file Tennis has been created.          |
    | It has 3 cases and 5 variables.               |
    |                                               |
    | Input file tennis.txt has 96 bytes.           |
    |                                               |
    | Default delimiters (blank, comma) were used.  |
    | 2 end-of-case bytes were used: CR,  LF.       |
    | Time: less than 0.1 second.                   |
    -------------------------------------------------

    Enter a command:
```

The MAKE command is given after the command prompt. If you are entering numeric data and you do not wish to provide your own variable names, you can use the NV (Number of Variables) identifier. NV requires an argument which is the number to variables to be created.

```
MAKE Test, NV 45;
```

This creates a file with 45 variables named var1, var2, ... var45. Usually you will have character variables or want to supply variable names. You can supply the variable names using the identifier VARS followed by variable names and data types or by entering the names and types as subcommands.

Figure 2.2 illustrates how to build and list a P-STAT system file when the data are already stored in an external file. The user entries are in a bold font and indicated by ">>" at the left edge

In Figure 2.2 you can see that the command portion ends not with the dollar sign but with a semi-colon.

```
MAKE Tennis, FILE 'tennis.txt' ;
```

The MAKE command requires information about both the data and the variables. The data information is supplied by using the FILE identifier followed by the name of the file which contains the data. The semicolon (;) ends the command portion of MAKE and indicates that subcommands come next. You must press return after the semicolon. The variable information is supplied using the VARS subcommand.

```
VARS  Last.name:c  First.Name:c  Sex:c  Age  USTA.Rating $
```

Variable names can be continued across several lines. But you must remember to press the “return” or “enter” key before moving to the next line on your terminal. The dollar sign indicates that the variable information is complete. The :c means that the variable is typed character rather than numeric.

Figure 2.3 illustrates how to build a P-STAT system file using the MAKE command and entering the data from the terminal. The command text and subcommands with the variable names may continue on as many lines as necessary. Press “return” (sometimes called “enter”) at the end of each line and continue typing the command. The semicolon (;) ends the *command portion* of MAKE. Because the FILE identifier is not used, the semicolon indicates that both *subcommands and data* follow. You must press return after the semicolon:

This is the command which contains the command name and the name for the P-STAT system file to be created.

```
Enter a command:
MAKE Tennis;
```

The basic elements of this MAKE command are:

1. the command name,
2. the name of the P-STAT system file to be built,
3. the semicolon, which signals the beginning of the subcommands and data, and
4. the dollar sign ($), which ends each P-STAT command. It is entered only after all command text, subcommand text and data records are complete.

When the semicolon is processed, the next prompt is for the subcommands with the variable names:

```
        Definitions (i.e., the names and types)
        of the variables are expected now.
        A response of just H or Q, at any time,
        will provide HELP, or will QUIT the command.

        The definitions should end with a semicolon.

        Begin entering variable names,
>>  Last.Name:C  First.Name:C  Sex:C  Age  USTA.Rating ;
>>  Begin case 1, Last.Name
```

The keyword MAKE is a command name recognized by P-STAT. It must be followed by a legal name for the P-STAT system file that is being built:

```
 MAKE  Tennis;
```

Figure 2.3 MAKE: Entering the Data From the Terminal

```
       Enter a command:
>>     MAKE  Tennis;

          Definitions (i.e., the names and types)
          of the variables are expected now.
          A response of just H or Q, at any time,
          will provide HELP, or will QUIT the command.

          The definitions should end with a semicolon.

          Begin entering variable names,
          or  H (for HELP) or Q (for QUIT):

_
>>      Last_Name:C  First_Name:C  Sex:C  Age  USTA.Rating ;

        begin case 1, Last.Name (c40):
 >>       Allen
        continue with First.Name (c40):
 >>       Robert
        continue with Sex (c40):
 >>       m
        continue with Age:
 >>       -
        continue with USTA.Rating:
 >>       4.5
        begin case 2, Last.Name (c):
 >>       Jones Karen f 13 4.5
       begin case 3, Last.Name (c40):
  >>     Prince Susan f
       continue with Age:
  >>     14 3.5
       begin case 4, Last.Name (c40):
  >>     $

        ------------------------------------------
        | P-STAT file Tennis has been created.   |
        | It has 3 cases and 5 variables.        |
        |                                        |
        | 2 delimiters were used: BLANK and COMMA. |
        | 1 missing value was found.             |
        ------------------------------------------
```

This portion of the command tells P-STAT to call the MAKE program and provides a name, Tennis, for the P-STAT system file. It is wise to use meaningful file names because they are easier to remember and help to document the run. The file name is your choice as long as it conforms to the rules for legal file names in P-STAT. The file name must be no more than 16 characters long, must begin with a letter and must contain only letters, numbers and decimal points.

Commas separate *phrases* in P-STAT commands. The command:

```
MAKE Tennis, FILE 'tennis.txt';
```

is a command with multiple phrases. Each phrase after the command phrase begins with a keyword identifier that is recognized by that command. FILE is a keyword identifier that is recognized by the MAKE command. It tells MAKE that the data are not going to follow but are stored elsewhere. FILE is an identifier which requires a single argument, the full name of the file where the data are stored. The MAKE command text ends with a ";" that indicates that command text is complete and that subcommand or data records follow.

If you make a mistake, such as spelling MAKE incorrectly, you will be placed in P-STAT's *editor*:

```
ERROR... MKAE is not the name of a P-STAT command.

EDITOR entered.
   5...MKAE  Tennis, .....
EDITOR:
```

The simplest thing for a new P-STAT user to do at this point is to enter "Q" to quit the editor.

```
>>  Q

    Leaving the editor.

    Enter a command:
```

Reenter the command after the command prompt appears. Q will always work to get you out of the editor and back to the command prompt. However, it can be frustrating to reenter a command only to make another mistake. It is very easy to make a simple correction and re-execute the command without typing the whole thing a second or even a third time.

```
    EDITOR entered.
       5...MKAE  Tennis, .....
    EDITOR:
>>  C  /MKAE/MAKE
       6...MAKE Tennis, FILE 'tennis.txt';
>>  X
     begin case 1, Last name (C):
```

The two editor instructions that are illustrated here are C (Change) and X (eXecute). Change is followed by two strings: the string with the error and the correct string. Strings are delimited by a single special character such as a slash or a colon at either end. The delimiter character must be a character that is not contained in either string. Once the change has been made the command must be re-executed for the change to take effect. This is done by entering a single X and pressing the return key. Another useful editor instruction is T (Type) which types the entire contents of the command or data record that caused the error.

A complete description of the P-STAT editor may be found in the manual "P-STAT: Utility Commands". In all systems the edit file is deleted at the end of the session unless it is explicitly saved. The chapter on using the editor explains how to save your edit file so that it can be re-used in subsequent sessions.

2.11 Defining Character Variables

In Figures 2.1 and 2.2, five variable names are given: Last_Name:C, First_Name:C, Sex:C, Age and USTA.Rating. Therefore, there must be five data values for each of the cases. The ":C" after the variables Last_Name:C, First_Name:C and Sex:C indicates that these three variables are character variables. Because there are no lengths specified, MAKE assumes that these character variables are no more than 40 characters long. If the last name of one of your tennis players is longer than 40 characters, you must provide an appropriate width:

```
Last_Name:C60
```

This indicates that the last names may be as long as 60 characters and MAKE can process those names accordingly. The specified character length may be up to 50,000 characters. If, as in this same example, you use:

```
Last_Name:C16
```

but one of the player's last names is longer than 16 characters, the value will be truncated to the defined length and the command report will have a line like:

```
One value was truncated.
This occurred in case 13 on variable Last_Name.
```

Figure 2.3 shows variables being entered in three ways. The first case is entered one variable per line. Notice that in interactive computing, P-STAT prompts the user for the proper variable on each line. The "(C):" in the prompt means that this variable is a character variable. All five variables for the second case are entered on the same line. A blank between the values serves as the delimiter. The third case is a combination with the five variables entered on two lines.

After the prompt "begin case 4", a "$" is entered. The "$" indicates that data entry is completed and that P-STAT should build this file. P-STAT reports that the input has been "terminated by a single $" and that three cases and five variables have been processed. Notice that the "$", which ends a P-STAT command, is not entered until *after* all the data records are entered.

The format for entering commands and data has few restrictions about where the values are placed in the input line. Commands must *begin on a new line*, but they may continue on multiple lines. Each data value may be entered on its own line after the prompt, or several values may be entered on each line. The MAKE program keeps track of the variables and cases you are entering and adjusts its prompting accordingly. However, when there are multiple data values on one line, they must be separated by *spaces* or *commas*. It is not necessary to begin each new case on a new line. However, if a new case starts before all of the values for the previous case have been entered, you must either provide the "-" default missing value character or an end of case character ("/").

```
     begin case 5, Last_Name (C) :
>>   Shubert William m 55 / Jones

     continue with First_Name (C):
```

Notice the spaces separating the data values, and the slash indicating the end of the case. This case is missing the final value and MAKE needs to know that the case is really ended so that it can move to the next case. An alternative way would be:

```
>>   Shubert William m 55 - Jones
```

Now the 5th variable is provided for with a symbol that indicates a missing value and "Jones" is properly allocated to the next case.

The MAKE.FIXED command can be used to process *fixed-format* records. Basic information on using the MAKE command is found in the chapter "Introduction to Data Input" later in this manual. The full capabilities of MAKE and MAKE.FIXED are described in the manual "P-STAT:File Management " MAKE is one of the more complex commands in P-STAT because it has all three elements, command text, subcommand text and data records.

2.12 Format of the LIST Command

LIST is the command that displays P-STAT system files. It lists the most recently referenced file, unless it is given a specific file name as its argument. LIST is followed by a "$", indicating that P-STAT should now process the command. (LIST may be shortened to just L without a dollar sign. It is the only P-STAT command that may be abbreviated in this way.) Figure 2.2 illustrates using LIST to display the Tennis file.

LIST automatically sets the proper spacing between the variables so that the listing is easy to read and yet concise. LIST has many optional identifiers that allow the user to control spacing, layout and data format. (See the chapter on LIST later in this manual.) To direct the listing of a file to a printer or diskfile, use the optional PR identifier in the LIST command. Follow PR with a name for the file on disk — put the name between single or double quotes:

```
LIST  Tennis,  PR  'Tennis.lst'  $
```

The listing will be in the disk file named “Tennis.lst”, which you can print after exiting P-STAT. Use your regular operating system instructions to print the file. Most computers also let you use the PRINT command from within P-STAT to print the disk file:

Figure 2.4 Listing a File and Ending the Run

```
      Enter a command:

 >>   LIST  Tennis  $


      Last      First                  USTA
      Name      Name      Sex    Age  Rating

      Allen     Robert    m       15     4.5
      Jones     Karen     f       13     3.0
      Prince    Susan     f       14     3.5

      Enter a command:

 >>   END  $
```

The PRINT command can also be used to print a system file directly.

```
PRINT Tennis $
```

When PRINT is given the name of an existing P-STAT system file it generates a generic LIST command and then sends the LIST output to the printer. In Figure 2.4, END $ is entered in response to the “Enter a command” prompt. END $ is the P-STAT command that gets the user out of P-STAT entirely.

Figure 2.5 Displaying Autosave and External File names

```
      Enter a command:

 >>   FILES $

      --------------------autosave files-------------------------
       name              current                  previous
       Tennis            TENNIS.PS1

      --------------------external files-------------------------
       name              usage         pathname
                         Transfer      start
                         External use  sue/tennis.txt
                         Printer       Tennis.lst
      -----------------------------------------------------------
```

2.13 AUTOSAVE AND EXTERNAL FILES

P-STAT is an autosave system — system files are automatically saved. After you build a system file, it is on your disk and available for use at any time. P-STAT adds the extension of ".PS1" or ".PS2" to the file's name. You must only refer to your file using the name that you gave it when you created it. The FILES command lists all P-STAT system files and external files referenced in the P-STAT session. It shows both the name you gave the file and the full pathname on disk. Figure 2.5 shows the FILES command and the report that it produces.

In the FILES report, autosave files are displayed first; external files are displayed next. The file named "Tennis" that we built earlier is on disk as "TENNIS.PS1". If this file is modified, a *backup* file is created — that is, both *current* and *previous* versions of the file exist. In the report in Figure 2.6, there is only a current version of Tennis. P-STAT keeps track of which is the current and the backup version of each file — you need only refer to the file by its name "Tennis". This is convenient but can be dangerous.

```
MODIFY test [ some modifications ], out test $
---run completes but you find the results are not right---
MODIFY test [ try to correct the error ], out test $
```

Do not use this feature when the input files have important data.

External files are displayed after the autosave files. External files include transfer files such as the initialization file "start", raw data files such as "sue/tennis.txt" and printer files such as "Tennis.lst". Other external files that might appear in the FILES report are labels files and edit files.

When you use P-STAT in a subsequent session, merely reference your system file by name:

```
LIST  Tennis  $
```

P-STAT locates the file on your disk and lists it. If you have forgotten the name of the file, use the SYSTEM command to pass an instruction to your computer's operating system:

```
SYSTEM  'dir'  $
```

2.14 BEGINNING TO USE P-STAT

The explanations in this introductory chapter are not complete, but they are sufficient to allow a new user to begin using P-STAT. There are other ways to MAKE and LIST files, and there are many optional identifiers to control various ways in which these commands operate. Some of the information in this chapter is repeated in the following two chapters both for emphasis and for the benefit of experienced users who may choose to skip this chapter.

Perhaps the best way to learn how to use the P-STAT system is to create small files like the one shown in this chapter. Read the initial introductory chapters in the parts of interest in the manual, and try using the commands with your small files. Once these initial chapters in a part are mastered, move on to another part. It is not necessary to read the manual from beginning to end.

The index in the back of the manual is comprehensive. It references commands and identifiers under many different topic headings, and should enable users to find explanatory text on almost all subjects. The bold entry under a given topic refers the reader to the chapter or main portion of a chapter that most fully describes the topic. Command summaries are also indexed, so that the list of identifiers for a command can be checked for pertinent options.

Of particular interest to new users is P-STAT's *on-line help file*. It contains information on both commands and general topics. The command HELP $ gives help on how to use the help file. To obtain information on a particular topic, enter "HELP Subject $", where subject is either a command name or a topic from the HELP menu.

Other HELP commands and the information they provide are:

```
HELP  COMMANDS  $        list of commands supported by HELP;

HELP  TOPICS  $          list of other topics;
```

```
HELP  EVERYTHING  $
```
list of the entire HELP file;

```
HELP  NEWS  $
```
information on the current release;

```
HELP  AUTOSAVE $
```
information on saving files.

To print HELP text, follow the commands with an output destination:

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

The NEWS feature is important because P-STAT is an evolving system — enhancements are made regularly. They are included in the latest version, and new versions are sent regularly to those P-STAT sites that are under maintenance. The NEWS contains the most current information.

The P-STAT help file is also available for use with browsers such as Firefox or Internet Explorer. The html files are located in the “pstathelp” folder below the directory/folder where P-STAT is installed. Point the browser to that folder and select “index.html” to access the helpfile entries.

SUMMARY

P-STAT COMMANDS

1. P-STAT commands begin with a *command name*. This is a keyword such as MAKE or LIST, which must be spelled correctly. Each command must begin on a *new* input line.
2. P-STAT commands end with a currency symbol ($). The "$" defines the end of *all* command, subcommand and data information for that command.
3. The semicolon (;) is used to indicate that *subcommands* or *data* follow. Since the beginning of the command is defined by the command name and the end of command text is defined by either a semicolon or the currency symbol, the command may begin anywhere on a new input line and may continue across lines with no special convention to indicate continuation.
4. P-STAT commands are composed of *phrases* which are separated from each other by a comma (,) .
5. Each phrase contains either a keyword *command* (the first phrase) or a keyword *identifier* (the other phrases). Some commands and identifiers need one or more *arguments*. Arguments supply the command or identifier with information such as file names and variable names.

P-STAT NAMES

Legal names for P-STAT commands, files and variables must follow certain rules:

1. File names may be no more than 16 characters long. They must begin with a letter and contain only letters, numbers and decimal points.
2. Variable names may be up to 64 characters long. They must begin with a letter can contain only letters, numbers, decimal points and the underscore character. In addition they may contain two parts: tag of 1-16 character followed by a double colon (::). The colon may not be used elsewhere in the name. The tag is followed by descriptive text to complete the variable name.

   Variable names in a given file must be unique. If tags are used, they must also be unique. Thus:

   Q1::Year_of_birth
   Q1::Date_of_birth would be illegal because the tags are not unique.:

   Q1::Year_of_birth
   Q2::Year_of_birth
   Year.of.birth These three are legal even though the text is duplicated.
3. Variables can be referred to by their tag, their text, the full name, or wildcard matching such as "ABC?", "?ABC", or "A?BC". Wildcard matching can be used anywhere that a variable name can be used. In some situations, a wildcard can have several matches, which are all perfectly valid. The expected number can be provided in curly braces, like {4} after the wildcard

P-STAT COMMANDS

TEXTFILE.IN

creates a P-STAT file from an external data file when the values are tab, comma, or blank delimited.

MAKE

creates a P-STAT system file from raw data. It can an handle free-format data stored in an external file or entered directly from the terminal

END

ends a P-STAT session. Deletes any temporary files created during a run.

FILES

lists the P-STAT system files created during the current run and any external files that have been used.

HELP

provides information on how to use the on-line help file.

3
The P-STAT Language

P-STAT instructions are written in the P-STAT command language, which has its own vocabulary, syntax and punctuation rules. The vocabulary is English-like, with command names that describe the function, such as MAKE, LIST, TITLES and END. The syntax and punctuation are straightforward.

This chapter discusses the P-STAT command language. Some of the information repeats that in the earlier chapters. This is both for emphasis and for the more experienced users who may have chosen to skip ahead.

3.1 P-STAT COMMAND SYNTAX

A P-STAT command always begins with a *command name* that is known to P-STAT, and it always ends with a *dollar sign* ($). It has one or more phrases. A phrase is one or more related words or values terminated by punctuation such as a comma, semi-colon, right bracket or dollar sign. Each command begins on a new input line. Two simple commands with just one phrase are:

```
HELP  $    and      END $
```

The HELP command lists the topics and commands for which help text is available. For example: “HELP LIST$” provides help on the LIST command. The END command completes the P-STAT run. The dollar sign ($) is the punctuation that ends a command and indicates that the next phrase begins a new command.

Most commands contain several phrases. Each phrase within a command begins with an *identifier* that is recognized by that command. The command name serves as the identifier of the first phrase. Commas separate phrases. This command has two phrases:

```
LIST  Team,  GAP 3  $
```

The first phrase begins with the command name — LIST. It prints a listing of a file. “Team” is the name of a P-STAT system file to be listed. The second phrase begins with the identifier GAP, which is recognized by the LIST command. It sets the spacing between the columns of variables. Here, the distance between the columns (the gap) is set to three spaces for this listing.

If you are using the P-STAT menu system on PC/windows, the command punctuation is correctly supplied and the names for commands and keyword identifiers are correctly spelled. You can learn the P-STAT syntax by examining the commands that are generated by the menu system. Not everything in P-STAT can be done from within the menus and a knowledge of command syntax may be necessary for problems which require advanced data modification features.

3.2 Legal Names

The names that you supply for your P-STAT system files must begin with a letter and may contain ONLY letters, numbers and decimal points. These file names are limited to 16 characters.

.Legal names for the *variables* in a P-STAT system file must:

- start with a letter,
- be no more than 64 characters long, and
- contain only letters, numbers, decimal points, and the underscore character.
- In addition variable names MAY begin with a 1 to 16 character tag separated from the text by a double colon (::).

The entire length of the tag and text must conform to the 64 character limit. Variable names and variable tags in a given file MUST be unique.

P-STAT **system** variables always begin and end with a decimal point. For example, .DATE.. is the system variable for the current date. **Scratch** variables begin with one or two pound signs. ##Counter is a permanent scratch variable whose value can be passed between commands. Temporary scratch variables begin with a single # and exist only for the duration of the current command. It is up to you, the user, to create your own scratch variables. The values you give them remain unchanged until you explicitly change them. Scratch variables and system variables are limited to a length of 16 characters and can only contain letters, numbers, decimal points and underscore characters.

The keyword TO is a reserved word in P-STAT and may not be used for variable names or file names. It is wise not to use BY, OUT, or any common prepositions as variable or file names because it is possible they may become reserved words in the future. The use of reserved words helps to simplify the command language.

It is also a good idea not to use single letter names for variables, especially the letters H, Q, E and V. In some situations, confusion may exist because these letters have special meanings in P-STAT as

s for HELP, QUIT, EDITOR and VARIABLE.

Names in P-STAT can be any combination of upper case and lower case characters. P-STAT retains the original way the names were entered and uses that representation when printing those names. Subsequent references to a name may be made in uppercase, lowercase, or mixed-case, regardless of how the name was entered originally. A file entered with the name Myfile can be referred to as “myfile”, “MYFILE”, or even as “myFILe”. However, a name must be spelled correctly even though the case may differ.

3.3 Basic Punctuation

The basic elements of punctuation are the *comma* (,), which separates phrases within a command and indicates that another phrase follows; the *semicolon* (;), which means that subcommand or data records follow; and the *dollar sign* or currency symbol ($), which completes the entire command. Figure 3.1 illustrates the first command in a simple run to create a P-STAT system file. Figure 3.2 contains the command needed to correlate the numeric values. Each of the commands in the run ends with a “$”. The end of that run is shown in Figure 3.3. The final command in Figure 3.3 is:

```
END  $
```

END is a single phrase command composed of the command name and a “$”. Its purpose is to end the P-STAT run and to close any open files. The “$” is the punctuation necessary to finish the command. If you terminate a P-STAT run without the “END $” command, you will probably find that some temporary work files are left behind. These files always have a file name that looks like “P_2PMor076.TMP” and may be safely erased. If you prefer to have temporary files stored elsewhere, the PSTEMP command can be used to specify an alternate folder where P-STAT can store these files.

3.4 Identifiers and Arguments

A P-STAT command name phrase may be followed by additional phrases that provide information about specific command options. *Identifiers* begin each phrase and identify which command option is desired. Identifier phrases are separated from each other by commas. For example, in Figure 3.1, the MAKE command name phrase is followed by the identifier SUMMARY. SUMMARY requests that a description file of statistics, summarizing the file being built, also be produced. Identifiers, like command names, must be spelled correctly in order to be recognized by P-STAT.

Identifiers sometimes require *arguments*. Arguments are additional information that a command or identifier needs. In Figure 3.1, the MAKE command name needs a name for the new P-STAT system file. The SUMMARY identifier needs a name for the description file (a special type of P-STAT system file) that is also to be produced. Arguments are often file names. They may also be variable names, numbers, keywords and character strings (such

as 'this is a title'). Keywords, like identifiers, are known to the system and must be correctly spelled. Character strings in quotes contain text or external file names of your choice.

These are some examples of commands and identifiers with arguments:

```
SORT Streams, BY State, OUT Streams $
LIST  Streams,  BY  State,  SKIP 5  $
PERCENTILES  SmithHS,  OUT  SmithPct,  GET  QUARTILES  $
TITLE  T1  CENTER  'Study of Water Resources'  $
```

There are three different types of arguments in the LIST command. “Streams” is the name of the file to list. “State” is the name of the variable by which the Streams files is to be listed. (The file is sorted by State, so it can be listed with the cases in each state grouped together.) “5” is the number of cases to list before skipping a line. The arguments in the PERCENTILES command are file names, except for the argument for GET — it is the keyword “QUARTILES” that tells the kind of percentile that is desired. The three arguments in the TITLES command indicate that this definition is for the first top title, that the title should be centered, and that the text of the title is the character string enclosed in quotes.

Some identifiers do not need arguments. These identifiers tend to be options that are selections. Including them in the command selects them or “turns them on”; excluding them does not select them or “turns them off”. These are some examples of identifiers that do not need arguments:

```
LIST,  TITLES,  COMMAS,  N  $
```

The most recently referenced file is listed when the name of a file is not given after the LIST command. TITLES includes any defined titles above the listing, COMMAS requests that commas be used in large numbers, and N numbers the cases.

Figure 3.1 Creating a P-STAT System File

File Solar.txt as it is stored on disk

```
5.162   .577   .808  11.08  17   1  1  'new machine'
5.086   .569  -  11.18  8.31  1  1    -
(62 more cases of data follow)
```

The P-STAT command to create a System file named 'Solar'.

```
MAKE  Solar,  FILE  'Solar.txt',   SUMMARY Solsummary ;
Eff  FF  Voc  Jsc  Ser  Process  Treatment  Comment:C40
$

------------------MAKE completed-----------------
| P-STAT file solar has been created.           |
| It has 64 cases and 9 variables.              |
| Summary file Solsummary was also created.     |
|                                               |
| Input file solar.txt has 3,904 bytes.         |
|                                               |
| Default delimiters (blank, comma) were used.  |
| One end-of-case byte was used: LF.            |
| Time:  less than 0.1 second.                  |
-------------------------------------------------
```

This command:

```
LIST,  IDEN  $
```

lists all the identifiers that may be used in the LIST command. IDEN is a *general* identifier that may be used in any command to get a list of the identifiers that are available for that command.

3.5 Phrases

P-STAT commands are built from *phrases*, which are separated from each other by *commas*. Each phrase is composed of a command name or an identifier, which may have one or more arguments. If there are arguments for an identifier, they follow that identifier, separated from each other by one or more blanks. The most important element of punctuation is the comma that separates phrases and indicates that another identifier follows.

Identifiers are keywords, that is, words recognized P-STAT. Therefore, their *order does not matter*. Thus, all three of these commands list a file with any defined titles, commas in large numbers and numbered cases.:

```
LIST EngFile, COMMAS,  N,  TITLES $
LIST EngFile, TITLES, COMMAS,  N $
LIST EngFile, N,  TITLES,  COMMAS $
```

Blanks for spacing and readability may (and should) be used freely between words. However, there should not be blanks within a word or command name.

Because P-STAT commands are composed of phrases with simple punctuation rules, they are referred to as *free-format*. Free-format means that they can begin anywhere on the input line. Commands may continue across lines without the need for any continuation conventions because P-STAT command text ends with the $ (or ; if subcommands or data records follow). However, a line should not end with an incomplete word. If an identifier or argument does not fit on the current line, press return and enter it on the next line. In addition, multiple commands cannot be placed on a single input line.

3.6 Subcommands

There are many commands that need more information than can be supplied in command text. For example, MAKE needs input data records, and SURVEY needs one or more table definitions. The *semicolon* separates command information from *subcommand text and data records* that follow. The dollar sign indicates that a command is complete and that there are no more subcommands or data records.

P-STAT makes no distinction between data and subcommands — both are additional information that some commands require. Therefore, the word “subcommands” is often used to refer to both data and subcommand text. Regardless of the content of the subcommands, they follow *after the semicolon on a new line*. Some examples of subcommands are:

```
SAVE.LABELS  'Cars.LAB' ;
  Origin  (1) United States  (2) Europe  (3) Japan /  $

TTEST  Bulbs,  OUT  Bulbs.T ;
  Hardy    Bloom.Time  1 3
  Fragile  Bloom.Time  4 5   $

TEXT ;
  Comments often help document a P-STAT run.
  All text up to the dollar sign is part of the comment.
  $
/* This is another way to insert a comment */
```

```
REGRESSION  Metal,  WEIGHT  Wt.Var ;
 DEPENDENT  Stress,  SUMMARY;
 AGAIN,  INDEPENDENT  Type  TO  Temp,  OUT  MetalReg   $
```

Notice that the content of the subcommands in the prior examples varies from command to command. That is because subcommand information is specific to the particular command. Thus, the format reflects the type of information that is required by the command. The subcommands DEPENDENT and INDEPENDENT may be abbreviated to DEP and IND.

There may be one or many subcommands (there are two subcommands in the REGRESSION command). In all commands, however, *subcommands follow after the initial semicolon* that ends the command portion, and the *dollar sign ends the entire command*. The next token after a dollar sign is expected to be another command name.

3.7 Analysis of a Simple Run

A simple P-STAT session that creates a P-STAT system file, correlates the variables in the file, lists the correlations, and exits P-STAT illustrates the P-STAT command language. It shows the structure of commands and that commands are entered one after another in sequence to perform a series of tasks . The four commands in the run are shown without prompts at the end of the chapter.

Figure 3.1 shows the first command in the run — the MAKE command. MAKE creates P-STAT system files from input data that are arranged so that there is some delimiter (blank, comma, or some other special character defined by the user) between each of the values. The name for the file to be built must be given. The name of the file where the data are stored is required unless they are to be entered from the terminal.

```
MAKE  Solar,  FILE  'Solar.txt',  SUMMARY  Solsummary ;
```

3.8 Missing Data

Sometimes information is missing. This can happen in a survey when a person refuses to answer a question or in medical research when a patient misses a visit. Missing data are permitted for both numeric and character variables.

Three different missing values — missing type 1, missing type 2, and missing type 3 — may be defined for each variable. This makes it possible to distinguish, for example, between the person who was not asked the question, the person who refused to answer, and the person who replied, “I don’t know.” Values may be defined as missing either when the file is built or when it is being modified in some way. The P-STAT Programming Language (PPL) provides extensive ways of testing and modifying both good and missing values.

Each of the three types of missing data is converted into a different, very large negative number. These numbers are not likely to occur in normal data. This representation of missing values makes it easy for P-STAT to detect missing values when computations are performed. In P-STAT, if a value in a computation is missing, the result is automatically missing.

The command portion of MAKE ends with a semicolon, which indicates that the command text is complete and that the rest of the information MAKE needs is to be supplied as subcommands and data. The subcommands must begin on a new line. Once P-STAT has processed the command, MAKE prompts for the variable definitions. Because the data are in an external file the variable definitions end with a “$” indicating that the command is complete.

P-STAT system files are automatically saved when they are created unless the system file name begins with “work”. These saved system files are available for use at any time in subsequent P-STAT sessions.

Figure 3.2 shows the second command in this brief P-STAT run. It is the CORRELATE command, which requests the correlation coefficients between all pairs of variables. Notice that the file produced by the MAKE command, “Solar”, is the input file for CORRELATE. Once a P-STAT system file is built, it may be used in any P-STAT command. This command contains a PPL (P-STAT Programming Language) *clause* in the initial command phrase:

```
CORRELATE  Solar  [ DROP  Comment  Process  Treatment ],
```

The brackets in the command contain a PPL clause. The PPL clause *modifies* the values from the system file as they flow from the file into the CORRELATE command. Here, CORRELATE never sees the three dropped variables. Notice that the PPL clause follows directly after the file to which it applies — there is *no intervening comma*. Commas separate identifier phrases, not PPL clauses. More literally, *commas come before identifiers*.

Figure 3.2 A P-STAT Run: Correlating the System File Values

```
CORRELATE  Solar  [ DROP  Comment  Process  Treatment ],
           OUT  SolarC $

   -------Correlate completed-------
   Input file solar has 64 cases
   and 5 numeric variables.
   There was no missing data.
```

The CORRELATE command continues on a second line — the OUT identifier gives a name for the output file that CORRELATE produces. P-STAT commands can continue on as many lines as necessary. They may break anywhere, except in the middle of a word. P-STAT adjusts its prompts accordingly when the session is interactive. Notice that the CORRELATE command produces a brief report; it does not show the correlations. This is because we provided the name for an output file. If there is no output file, the CORRELATE command prints the correlation coefficients. Some commands produce output that is displayed on the terminal, some produce output that is contained in a P-STAT system file, and some do both.

Figure 3.3 shows the third and fourth commands in this P-STAT run. The LIST command displays the correlations that are in the output file produced by CORRELATE. LIST has three phrases. The END command exits P-STAT.

Figure 3.3 A P-STAT Run: Listing the Correlations and Exiting P-STAT

```
LIST  SolarC,  PLACES 2,  GAP 3  $

                                                  row
  Eff       FF       Voc      Jsc      Ser      label

 1.00     0.99     0.60     0.93    -0.80   Eff
 0.99     1.00     0.56     0.92    -0.80   FF
 0.60     0.56     1.00     0.37    -0.36   Voc
 0.93     0.92     0.37     1.00    -0.86   Jsc
-0.80    -0.80    -0.36    -0.86     1.00   Ser



END  $
```

3.9 COMMAND PROCESSING

P-STAT has an executive program that is responsible for reading the free-format P-STAT commands, storing the information in the command, calling the requested routines, and making the information (identifiers and arguments) found in the command available to those routines. In the case of a command such as this:

```
MAKE Students,  FILE  'Students.txt' ;
```

the information is one argument (the file name "Students") for the command, one identifier (FILE) and one argument (the name of the raw data file "Students.txt") for the identifier FILE.

The executive program contains a list of all the P-STAT command names. When there is a match between the characters read from the terminal and one of the names in its list, the executive program is satisfied. For example, "MAKE" is a name that matches a command in the executive routine, but "MKAE" does not. As a result, a typing error like "MKAE" causes an error condition.

When a command is correctly identified, the executive program calls the proper routine to execute that command. In addition, it passes along the identifiers and their arguments to the routine. In the prior MAKE example, the executive program has stored the argument for MAKE, the fact that the FILE identifier is used and the argument for FILE, so that they are now available for analysis by the MAKE command.

After a command has been processed by the executive program, that is, after the characters from the command name through the ";" or "$" have been read, analyzed and stored, control is passed to the command itself. A command like SURVEY has many identifiers, but only some are required. All identifiers are keywords that must be spelled correctly in order to be recognized by a command. If any identifier is misspelled, there is an immediate error condition. There is also an error condition if a *required* identifier is missing. Once the command has finished, control is returned to the executive program, which looks at the input for the next command.

P-STAT commands are executed in the order in which they are entered. Therefore, the order must follow a logical progression. For instance, a file must be created before it can be analyzed. In an interactive session, the order may well be determined by the results of the previous step. In a batch run, the order must be determined ahead of time. Commands may be entered into an external file using an editor of your choice and executed using the TRANSFER command in either an interactive or batch job.

Figure 3.4 The TRANSFER command

```
The file JobStream.trn contains the following 3 P-STAT commands on
a series of 80 character records.

MAKE  Solar,  FILE  'Solar.txt',   SUMMARY Solsummary ;
      Eff  FF  Voc  Jsc  Ser  Process  Treatment  Comment:C40$
CORRELATE  Solar  [ DROP  Comment  Process  Treatment ],
         OUT  SolarC $
LIST  SolarC,  PLACES 2,  GAP 3  $

 Use the TRANSFER command to execute these three commands.

 TRANSFER JobStream.trn $

 The results will appear on your terminal and when they have
 been successfully executed you can continue to enter more
 commands or you can end the run using the P-STAT command  'END $' .
```

Another mechanism is the macro facility, which is discussed in detail in the manual "P-STAT: A Guide to the P-STAT Programming Language (PPL)". A block macro is a group of P-STAT commands with its own name. An instream macro is text to be used in a P-STAT command, subcommand or even in an instream data record. The entire series of commands or the text is invoked by the use of the macro name. Any string of characters in the macro, such as a file name, can be changed when the macro is called. Macros can be stored in a macro library, which is simply an external text file that contains the macros. You can TRANSFER to it at the start of a run to activate the macros, which makes them ready for use.

3.10 Command Output

P-STAT commands can be divided into three categories based on the type of output they produce:

1. No output;
2. Printed output;
3. P-STAT system files.

Certain commands, such as SCREEN and PRINTER.SETTINGS, produce no output at all. This type of command usually serves to define the environment — set the number of lines on the screen, specify printer attributes, and perform other related tasks.

A second type of command is designed to produce printed output. Commands such as SURVEY and LIST fall into this category. The output that they produce is printed (displayed) on the terminal ***unless*** it is directed to a disk file.

The third type of command is designed to produce new P-STAT system files. MODIFY, SORT and LOOKUP fall into this category. Although each of them also produces a brief report stating what was done, the primary output is one or more new P-STAT system files.

There are a few commands that produce files that are not P-STAT system files. Commands like SPSS.OUT and TEXTFILE.OUT provide an interface to other programs. The SAVE.LABELS command writes a formatted (“readable”) file of variable and value labels on disk. A special type of command is the P-STAT EDITOR command, available only in interactive sessions. The edit file produced by the EDITOR can be saved as an external file. It provides a complete log of the P-STAT session and can be used to re-execute the job stream at a later time, either exactly as is or with changes.

3.11 Repeating a Run

It is often necessary to repeat a command or a series of commands, with minor changes. In P-STAT there are several different mechanisms for doing this. In an interactive session it is easy to enter the editor, locate the commands that need changing, make the necessary changes, and then execute the changed commands.

In an interactive session, the AGAIN command may be used to repeat the prior command exactly as is or with changes. Changes are specified as additional identifiers and arguments that are to be included in repeating the previous command:

```
AGAIN,  GAP 4,  SKIP 2,  PR  'PrtFile'  $
```

This command repeats the prior LIST command, but with 4 spaces between the columns of variables, with a line skipped between every two cases, and with the listing directed to a print file. The identifiers used in the AGAIN command must be acceptable ones for the command that is being repeated. (Unlike other commands, AGAIN is never written in the edit file and *cannot be used in a batch run.*)

3.12 File Modification

The data set in Figure 3.5 illustrates longer variable names some with both initial TAGS and TEXT. This is the data set used in Figures 3.6 , 3.7. and 3.8. In Figure 3.6 you can see that the tag is sufficient to identify the variable in the PPL. The file Students remains unchanged.

Figure 3.5 Data Set With TAGS And TEXT

```
MAKE Students;                        (command text)

 Birthdate Sex Term                   (subcommand section
 Test1::English_Poetry                 provides variable names)
 Test2::Social_Studies
 Test3::Elements_of_Algebra;

 11041994 1 2 68 73 72                (data section)
 01151995 1 1 83 79 91
 06291994 2 3 75  - 80
 08041995 1 2 89 95 97
 10231995 1 2 70 83 75   $
```

The values in P-STAT system files can be modified as they are read by *any* P-STAT command. Note: this does not change the system file itself. The MODIFY command in Figure 3.6 illustrates how this is done. Many modification (PPL) *clauses* may follow a P-STAT system file name. Each group of PPL statements is enclosed in square brackets. In Figure 3.6, the modification clause that generates a new variable equal to the mean scores on a series of tests is:

```
GENERATE GPA = MEAN.GOOD ( Test? )$
```

The new file Students contains all the variables that are in original file Students, plus the new variable named Average. (Notice that the full TAG+TEXT variable names are carried to the output file.) When the *same* file name is used for the output file as in this example:

```
MODIFY  Students [ GENERATE Average = MEAN.GOOD ( Test1 TO Test3 )],
    OUT  Students  $
```

a new file named Students is created. It contains all the variables that were in the input file, plus the new variable. In these examples the use of (Test?) and (Test1 TO Test3) are 2 ways to select all the variables which begin the the letters “Test”). The choice of upper or lower case makes no difference in the ways these functions work. (Data modification is discussed in detail in the manual “P-STAT: A Guide to the P-STAT Programming Language (PPL)”.)

The MEAN.GOOD function is interesting here. What happens when a student has missed a test? In this example, the empty value for the 3rd case on Test2 is represented with a dash and the average is computed using the non-missing data values. All of the arithmetic functions in P-STAT come in 2 forms. The use of the simple function MEAN when there is missing data will cause the result to be missing.

The MODIFY command raises some other questions. Are there now two files named Students? Yes, BUT only the most recent one will be selected in ordinary usage. P-STAT adds the extension of PS1 to the first use of the file. A subsequent modification with STUDENTS as the output file would produce STUDENTS.PS2. If that file is modified, using the same name would produce STUDENTS.PS1. P-STAT can tell which is the newer and, unless you say otherwise, you always get the more recently created file. How, then, would you access the previous version? By using [PREVIOUS] as the initial PPL clause:

```
LIST  Students  [ PREVIOUS ]  $
```

When PREVIOUS is not used, it is assumed that the *current* version of a system file is the one being referenced. CURRENT may be used to explicitly refer to the current version; it is used just like PREVIOUS is used. When a third version of the file Students is produced, the formerly current version becomes the previous version and the formerly previous version is gone. These system files are also called AUTOSAVE files, because they are automatically saved. Note: this feature is a 2-edged sword; very useful when designing a run or trying different

features, but there is a chance for disaster if an error in a run is not detected and the backup that you rely on no longer exists.

Figure 3.6 Use Scratch Variables and Date Formats

```
  GENERATE ##date.fmt:c  =  'yy_mon_dd' $
  GENERATE ##date2.fmt:c =  'Month-dd-yyyy' $

  MODIFY Students
  [ GENERATE GPA = MEAN.GOOD ( Test? ),
    GENERATE Date_of_Birth:c = MONTH.DAY.YEAR ( birthdate ),
    GENERATE Age = EXTRACT.YEARS ( current.date())   -
                   EXTRACT.YEARS ( Date_of_Birth ) ],
   OUT StuMeans $

/* Note: GENERATE can be abreviated to GEN */

  LIST StuMeans
  [ GEN Date_short:c    = FORMAT.DATE ( Date_of_birth, ##date.fmt ),
    GEN Date.expanded:c = FORMAT.DATE ( Date_of_birth, ##date2.fmt ) ]

  [ KEEP Age Date_short Test? GPA Date.expanded ]  $

   END $
```

Age	Date short	Test1:: English Poetry	Test2:: Social Studies	Test3:: Elements of Algebra	GPA	Date expanded
17	94_nov_4	68	73	72	71.00000	November-4-1994
16	95_jan_15	83	79	91	84.33333	January-15-1995
17	94_june_29	75	-	80	77.50000	June-29-1994
16	95_aug_4	89	95	97	93.66667	August-4-1995
16	95_oct_23	70	83	75	76.00000	October-23-1995

P-STAT has many functions and the manual "P-STAT A Guide to the P-STAT Programming Language (PPL) covers the language in its entirety. Figure 3.6 illustrates some simple PPL with a KEEP which selects variables from a P-STAT system file, and an arithmetic function to compute the variable GPA. It also includes some more unusual PPL to illustrate that the language is both powerful and easy to use. There are 2 scratch character variables "##date.fmt" and "##date2.fmt" which are used as templates describing 2 ways that a DATE value might be printed.$

```
GENERATE ##date.fmt:c  =  'yy_mon_dd' $
GENERATE ##date2.fmt:c =  'Month-dd-yyyy' $
```

Dates are interesting because there are so many different ways to arrange the pieces and there are both character and numeric representations to be considered. The ability to define a preferred format for representing a date and to store that format in a scratch variable where it is available to use at any time during a P-STAT run is very

useful. The PPL manual covers all of the P-STAT programming language including a chapter devoted to the numerous DATE functions.

Figure 3.7 An Error in an Interactive Run

```
      MODIFY  Students [ GEN  Average = MEAN (Test1 TO Test3),
         OUT  StuMeans  $

 Error... A dollar has been found ending a record:
             MODIFY Students [ GEN Average = MEAN (Test1 TO Test3), OUT
                   StuMeans $
           The brackets, however, are not balanced.
           The earliest left bracket still open began here:
              [ GEN Average = MEAN (Test1 TO Test3), OUT StuMeans $

      EDITOR entered
       2...MODIFY  Students [ GEN  Average = MEAN (Test1 TO Test3),
              OUT  StuMeans  $
      EDITOR:
      Change /)/) ]
      2...MODIFY  Students [ GEN  Average = MEAN (Test1 TO Test3) ],
              OUT  StuMeans  $

      EDITOR:
      X
       5 cases processed from file Students.


      LIST  StuMeans,  PLACES 2 $

                    Test1::  Test2::
                    English   Social  Test3::
  Age   Sex  Term    Poetry  Studies  Algebra    Average

   16     1     2       68       73       72   71.00000
   17     1     1       83       79       91   84.33333
   17     2     3       75        -       80   76.00000
   16     1     2       89       95       97   93.66667
   18     1     2       70       83       75   76.00000


      LIST  StuMeans,  PLACES 2, SHORT $

      Age   Sex  Term   Test1 Test2 Test3  Average

       16     1     2     68    73     72    71.00
       17     1     1     83    79     91    84.33
       17     2     3     75    73     80    76.00
       16     1     2     89    95     97    93.67
       18     1     2     70    83     75    76.00
```

3.13 ERROR PROCESSING

The sequence that P-STAT follows in processing commands is to:

1. read a command,
2. process that command,
3. read the next command,
4. process that command, and so on.

As a result, P-STAT processes errors *as they occur*. There are different outcomes, however, depending on whether the job is interactive or batch. Figures 3.7 and 3.8 illustrate how P-STAT handles errors in interactive and batch jobs, using the same commands containing the same error.

3.14 Error Processing in Interactive Runs

Figure 3.7 illustrates a short but complete run. (In this manual user supplied variable names and character strings are printed in *mixed* uppercase and lowercase. P-STAT keywords are in all *upper* case.) Short error messages are assumed in interactive execution, although the error setting may be changed at any time in a run by using the ERROR command:

```
ERROR  LONG  $
```

In an interactive run, each command is immediately followed by its output unless an error occurs. When an error is encountered in an interactive session:

1. an error message is printed on the terminal,
2. P-STAT's editor is automatically entered, and
3. control returns to the user.

The editor within P-STAT lets interactive users correct their P-STAT commands, subcommands and data records and then re-execute the command.

In Figure 3.7, there is an error in the MODIFY command because a right bracket is missing. In an interactive run the error can be immediately corrected by making the necessary changes in the editor. The MODIFY command is then re-executed and, only after StuMeans is created, is the LIST command entered. The editor is fully documented in the manual "P-STAT: Utility Commands".

3.15 Error Processing in Batch Runs

When an error is encountered in a batch run, P-STAT tries to continue if it can — that is, if the error does not prevent the next command from being executed. When there are more than five consecutive errors, P-STAT terminates the run at that point. The rest of the commands are not processed. Each time an error is found, the error counter is increased by one. When a good command is processed, the error counter is reset to zero.

The P-STAT command MAXERROR allows the user to change the error limit from five to any positive integer:

```
MAXERROR 3  $
```

In a batch run involving a very large file, you might wish to set MAXERROR to 1, because continuing may be both useless and time-consuming once an error has occurred.

Figure 3.7 illustrates error processing in a batch job. All of the input (the command stream) precedes all of the output. In a batch run, there is no way to recover from many errors. The LIST command that follows the MODIFY cannot be done because the file to be listed is never created. A *fatal* error occurs, and the P-STAT run ends with an error message. Batch error messages are surrounded by a band of EEE's so they stand out in the printed output — that is,

```
ERROR  LONG  $
```

is assumed in batch execution.

Figure 3.8 An Error in a Batch Run

P-STAT FILE:

```
/*Illustrate an Error in a Batch Run'   $ */

BATCH $
MODIFY  Students  [ GEN  Average = MEAN ( Test1 TO Test3),
   OUT  StuMeans  $

LIST    StuMeans  $
```

P-STAT OUTPUT:

```
EEEEEEEEEEEEEEEEEEE
E
E   A dollar has been found ending a record:
E
E     Modify Students[ GEN Average = MEAN (Test1 TO Test3), out
E
E   StuMeans $
E
E   The brackets, however, are not balanced.
E
E   The earliest left bracket still open began here:
E
E      [ GEN Average = MEAN (Test1 TO Test3), out StuMeans $
E
EEEEEEEEEEEEEEEEEEE
```

SUMMARY

P-STAT COMMAND SYNTAX

P-STAT commands begin with a command name. This is a keyword such as MAKE, LIST or SURVEY, that must be spelled correctly.

1. P-STAT commands end with a dollar sign ($). The "$" defines the end of *all* command, subcommand, and data information. The next thing after a dollar sign is assumed to be a P-STAT command.
2. The semicolon (;) indicates that subcommand or data records follow.
3. Each command must begin on a new input record (line). The record is expected to be no longer than 80 characters. Since the beginning of the command is defined by the command name, and the end of command text is defined by either a semicolon or the dollar sign, the command may begin anywhere on a new input record and may be continued across 80 character records with no special convention to indicate continuation.
4. P-STAT commands are composed of *phrases* that are separated from each other by a comma (,). The command name itself often begins a phrase.
5. Each phrase contains either a keyword command (the first phrase) or an identifier. Some command names and identifiers need one or more *arguments*. Arguments supply the identifier with information such as file and variable names.
6. Character string arguments such as comments and titles must be enclosed in quotes.

When an input argument for a command name or an identifier is the name of a P-STAT system file, the name may be followed by data modification (PPL) *clauses*. These clauses must be enclosed in square brackets.

P-STAT NAMES

Legal names for P-STAT **system files**:

1. They must begin with a letter.
2. They may be no more that 16 characters long
3. They may contain only letters, numbers and decimal points.

Legal names for **variables** in a P-STAT system file:

1. They must begin with a letter.
2. They may be no more than 64 characters long including the optional tag.
3. The optional 1 to 16 character tag must be followed by a double colon (::).
4. The TAG and the following TEXT may contain only letters, numbers, decimal points and the underscore character. For example: Q1.A::Date_of_Birth Q1.B::Years_of_Education

   When there are tags and long labels, the wildcard character is often useful. For example:

   MODIFY Test [KEEP Q1?] ....

Legal names for **scratch variables**:

1. They must begin with either # or ##. A single # sign indicates a temporary scratch variable. A double ## sign is used for a permanent scratch variable. Temporary scratch variables exist only for the duration of the current command.
2. They may be no more than 16 characters long
3. They may contain letters, numbers, decimal points or underscore characters.

Variable Name Identifiers:

SHORT

SHORT requests that only the tag portion or the first 16 characters of the variable name be used.

SHORT.TAGS is the equivalent of SHORT.

SHORT.16

SHORT.16 requests that the first 16 characters of the variable name be used in printouts.

SHORT.OLD

SHORT.OLD request that the labels be formatted according to the rules used in P-STAT version 2.

TEXT

TEXT requests that the text portion be used in the printouts. If there is no TAG the full variable name will be used.

FULL

requests that, if possible, the full 64 character variable name be used in printouts and reports.

EXAMPLES

```
1. END $

2. LIST File1, PLACES 2, SKIP $

3. MAKE Students, FILE'stu.dat';
   Last_Name:c Class Grade Gender  $

4. CORRELATE Patients [ IF Doctor = 'Schmidt', RETAIN ],
       OUT PatCor $
```

Example 1 contains a single phrase. Example 2 contains 3 phrases. Example 3 has two phrases and one subcommand. Example 4 has two phrases — they are separated by the comma that precedes OUT. The CORRELATE command name has as its argument the name of the P-STAT system file that is to be input to the command. The argument is followed by a data modification clause. The command name, the file name, and the modifications to that file are all part of the first phrase.

A COMPLETE RUN

```
/* This is a comment.  Comments can appear anywhere in the
   run to document what the run is doing */

MAKE Solar,  SUMMARY Solsummary,  FILE 'Solar.txt';
    Eff  FF  Voc  Jsc  Ser  Process  Treatment  Comment:c40 $

Correlate Solar [ DROP Comment  Process  Treatment ],
```

```
    OUT  SolarC $
LIST SolarC,  PLACES 2,  GAP  3, FULL $
END $
```

4 Introduction to Data Input

There are several commands that can be used to process your raw data and create a P-STAT system file. MAKE, which was briefly illustrated in previous chapters, and TEXTFILE.IN are two of the most important commands for this purpose. This chapter is an introduction to both of them. It also covers the command SPSS.IN which reads an SPSS portable file. MAKE is more powerful than TEXTFILE.IN as it can both read data from an external file and data entered directly from the terminal All three of these commands have a corresponding command to export the data. TEXTFILE.IN, which is a more general name for the TABFILE.IN, command is used to import files produced by commands such as EXCEL where the values in each case are usually delimited by a tab, comma or a blank. The first case often contains the variable labels. SPSS.IN is used specifically to read data produced by SPSS in EXPORT format.

Figure 4.1 TEXTFILE.IN

```
File test.csv contains:

Item 1,Item 2
1,2
2,1

The P-STAT command:

TEXTFILE,IN Test,  FILE "test.csv",  DEL COMMA $

The output report:

--------------TEXTFILE.IN completed---------------
| P-stat file work has been created.              |
| It has 2 cases and 2 variables.                 |
|    2 variables are numeric.                     |
|    0 variables are character.                   |
|                                                 |
| Input file test.csv contains 22 bytes.          |
| Values were delimited by a COMMA.               |
|                                                 |
| Variable labels were read from the input file.  |
| Time: less than 0.1 second.                     |
---------------------------------------------------
```

4.1 TEXTFILE.IN

TEXTFILE.IN, an alias for TABFILE.IN, is often used for importing files which are tab, comma, or blank delimited. On PC/Windows the usual extensions for these files are:

1. tab delimited tab
2. comma delimited .csv
3. blank delimited txt

Figure 4.1 shows a very simple comma delimited file and the command to create a new P-STAT system file.

The DELIMITER (DEL) identifier is not required when tabs are used as the delimiter but if you know the delimiter, then it is best to specify it. Without “DEL COMMA” this TEXTFILE.IN command will find a single character variable with the variable label ‘Item, Item.2” and data values “1,2” and “2,1”.

As you can see from the report in Figure 4.1, the variable labels are read from the input file. If there are no variable labels in the input file the command should read:

```
TEXTFILE,IN Test,  FILE "test.csv",  DEL COMMA, NO LABELS $
```

Without the “NO LABELS” instruction the command will happily create labels from the first row of data. In this case the numbers 1 and 2 would be converted to “V1” and “V2” to make them legal for P-STAT variable names. A case can be incomplete. When the end-of-record occurs earlier than expected for a case, the values it lacks are simply treated as missing.

Usually TEXTFILE.IN reads through the data to determine if a variable is numeric or character. You can use the identifiers CHARACTER and NUMERIC to provide the names of variables that are to be explicitly typed. If the variable names are not supplied, you must use the labels that P-STAT will generate For example:.

```
CHARACTER var1 var9
```

indicates that the first and ninth variables (in a file that has no variable names) are to be considered character even if the data values are all numeric.

4.2 Strings in TEXTFILE.IN

Input values can occur as strings, like "abc". Delimiters (like comma) or an end-of-record indicator (like CR-LF), when found in a string, are treated as ordinary characters, without special meaning. A string is recognized when the initial non-blank character of a value is a quote (") or apostrophe (’). The string ends when the balancing quote or apostrophe is found.

The contents of the string are the characters (if any) between the string boundaries. The contents are not tested for the delimiter or end-of-record characters, and are moved into the P-STAT file as is. If non-blank characters occur after the string and before the delimiter, they are appended to the string contents. Thus, given a comma delimiter,

```
"abc"123 becomes abc123.
```

The IGNORE.STRINGS identifier can be used to turn off the string processing entirely. When HONOR.STRINGS is specified special characters such as the delimiter are not recognized until the final string boundary is processed.

```
STRING.BOUNDARY  apostrophe | quote or quotation.mark| both or either,
```

The default is either. This is used when the first non-blank character in the value is found, to decide if it is the start of a string.

4.3 LABELS in TEXTFILE.IN

The input file will usually have the names of the variables in it's initial record. P-STAT will change these names as needed to make them into legal P-STAT names. There are two optional output files which may be useful in seeing what got changed.

The LABEL.CHANGES identifier, followed by a filename, can be used to write a system file which has, for each variable, the original name and what it was changed to. The EXTENDED.LABELS identifier, followed by the name of an external file, writes a file in P-STAT labels format, with the original name as the extended label.

4.4 SPSS.IN

The SPSS.IN command reads a file that is in SPSS export format and creates a P-STAT system file. SPSS.IN has no identifiers except for:

1. FILE which provides the name of the external file which contains all SPSS information including value labels
2. LABELS which provides a name for a new file of value labels to be created from the SPSS information.
3. SYSTEM.MISSING which defines how SPSS's system missing value should be treated. The argument is either 1, 2, or 3 which indicates which of P-STAT's missing values should be used.

A typical command is:

```
SPSS.IN psfile,  FILE 'from.por',  LABELS 'new.lab' $
```

The SYSTEM.MISSING identifier is not needed if MISSING 3 is appropriate. However, if you have provided values in an SPSS.OUT command for missing data and are using SPSS.IN as a check to see how SPSS will interpret the data the results can be confusing.

4.5 MAKE ESSENTIALS

This section describes how to use the P-STAT MAKE command. It can be run from a command file or from the terminal. When it is run from the terminal there are prompts for the variable information and the data values.

MAKE reads ascii data and creates a new P-STAT system file. The data can be entered on-line, or can be read from an external input file. The names and types of the variables can be provided within the command, placed in subcommands, or read in a separate DEFINITIONS file.

An input file will usually have an end-of-case indicator ending each case; carriage return with line feed is common, as is just a line feed. These are identified by the program. Long, non-standard end-of-case fields are supported. Data values can be numeric, character, or strings that are bounded by a quote or an apostrophe. A bounded string can contain either a numeric or a character value.

On-line input assumes a slash as an end-of-case indicator, and also supports asterisk as a repeat indicator; i.e., 30*1.234 means use 1.234 thirty times. Values can be changed as they are being read.

The data values for MAKE are known as "free format" or "delimited". That means that the values are separated from each other by one or more characters. Blank and comma are the default delimiters when the data are entered from the terminal. When the data are in an external file a comma is the default delimiter. Regardless of where the data are located you can specify one or more alternate delimiter characters if they are needed to uniquely identify the end of a value.

The MAKE command needs to have a name for the new P-STAT system file; the names and data types for the variables and the location of the data records.

Figure 4.2 MAKE: Delimited Data

```
----------The Data----------

          1 0 0 1 1
          1 1 1 0 0
          0 1 1 1 0

----------Six different ways to enter this data----------

 1.       MAKE Items, NV 5;
          1 0 0 1 1
          1 1 1 0 0
          0 1 1 1 0
          $

 2.       MAKE Items, NV 5;
          1 0 0 1 1 1 1 1 0 0 0 1 1 1 0 $

 3.      MAKE Items;
          Test1 to Test3 Q1 Q2;
          1 0 0 1 1 1 1 1 0 0 0 1
          1 1 0 $

          ----------------MAKE completed---------------
          | P-STAT file work has been created.         |
          | It has 3 cases and 5 variables.            |
          |                                            |
          | Two delimiters were used: BLANK and COMMA. |
          ----------------------------------------------

     Three ways to provide the variable names
     when the data are in an external file

 4.       MAKE Items, NV 5, FILE 'data.txt' $

 5.       MAKE Items, TEMPLATE Previous.file,  FILE 'data.txt' $

 6.       MAKE Items, FILE 'data.txt';
          Test1 to Test5 $

          ----------------MAKE completed---------------
          | P-STAT file work has been created.         |
          | It has 3 cases and 5 variables.            |
          |                                            |
          | Input file data.txt has 34 bytes           |
          |                                            |
          | Two delimiters were used: BLANK and COMMA. |
          | Two end-of-case bytes were used: CR, LF.   |
          | Time: less than 0.1 second.                |
          ----------------------------------------------
```

4.6 Variable Names and Data Types

Variable names follow the rules for legal names in P-STAT. The first character must be a letter and the name may contain only letters, numbers, decimal points or underscore characters. Legal names must be from 1 to 64 characters long. A variable name may start with a "tag" of 1 to 16 characters ending with a double colon (::). Both tags and the entire variable name must be unique. A group of variables with the same prefix can be entered by providing the first variable name followed by the word "TO" and then the last variable name. Variable names in a "to" list are generated in ascending numeric order. The following are examples of legal variable names:

```
Age     Gender    Item5 to Item 15   Group.number
Item1::test_score  Item2::test_score
Item3::favorite_novel:c
```

Variables in P-STAT are typed as numeric or character. Numeric is assumed unless character is explicitly specified. A single character value can have from 1 to 50,000 characters. For example:

```
Name:C   Name:c40   Comment:c3000
```

If the variable name is followed by the letter "C" or "c" with no further information, it is assumed to be 40 characters wide.

In Figure 4.2 the variables in the three examples which have the identifier "NV" are given names beginning with the prefix "var" followed by a sequence number. The other three examples have the variable names provided either as subcommands or in a TEMPLATE file. The TEMPLATE file is an existing P-STAT system file which has the appropriate variable names and data types. The TEMPLATE file may have data records or it may have been created just to provide variable information for subsequent commands. There are a total of 5 different ways to specify the variable names:

1. Use the NV identifier in the command
   ```
   MAKE Work,   NV 5;
   ```
2. Use the VARS identifier in the command
   ```
   MAKE Work,  VARS  Id v1 TO v4;
   ```
3. Use the TEMPLATE identifier in the command
   ```
   MAKE Work,  TEMPLATE pstnames;
   ```
4. Use the DEFINITIONS identifier in the command to specify a text file containing the names
   ```
   MAKE Work,  DEF 'varnames.txt';
   ```
5. Provide subcommand records following the command text
   ```
   MAKE work;
   Id v1 to v4;
   ```

There are two ways to tell MAKE where your data are located.

1. Use the identifier FILE in the command
   ```
   MAKE Work,  FILE  'mydata.txt',  NV 5 $
   ```
2. Omit FILE and supply the data records in a separate section following the other information
   ```
   MAKE Work,  NV 5;
   1 3 5 0 1 .....
   $n

   MAKE Work;
   v1 to v5;
   1 3 5 0 1.....
   $
   ```

Notice the punctuation. Keywords such as the command name and the identifiers are separated by the use of a comma (,). The semi-colon (;) is used to define the sections. The dollar sign ($) is the punctuation that tells

P-STAT that the command is complete and that the next token will be another P-STAT command. There are three possible sections in the MAKE command:

1. The MAKE command and the new file name plus any identifiers and their arguments. In Figure 4.2 all six examples have a command section. Examples 4 and 5 have only a command section.
2. The variable names and data types section. This section is omitted if any of the four identifiers; NV, VARS, DEF, or TEMPLATE are used. In Figure 4.2 examples 3 and 6 have a variable names section following the command section..
3. The data section. This section is omitted if the FILE identifier is used. In Figure 4.2 the first 3 examples have an inline data section. In examples 4-6 the data are stored in an external$ file.

Figure 4.3 MAKE: The Delimiter Character(s)

```
     MAKE work, NV 5;

     begin case 1, var1:
     1, 2, 3 4, 5 $


     -----------------MAKE completed-----------------
     | P-STAT file work has been created.           |
     | It has 1 case and 5 variables.               |
     |                                              |
     | Two delimiters were used: BLANK and COMMA.   |
     ------------------------------------------------

      var1  var2  var3  var4  var5

         1     2     3     4     5

------------------------------------------------------------

     MAKE work, NV 5,  DELIMITER ',';

     begin case 1, var1:
     1, 2, 3 4, 5 $


     ------------MAKE completed-------------
     | P-STAT file work has been created.  |
     | It has 1 case and 5 variables.      |
     |                                     |
     | The delimiter character was: COMMA. |
     ---------------------------------------

      var1  var2  var3  var4  var5

         1     2    --     5     -
```

MAKE output always includes a simple report. In the first three examples the data are entered following the command and the MAKE report is short and simple. In the final three examples the data are located in an external file named “data.txt”. The MAKE report when the data come from an external file always has additional information describing what MAKE found in the external file.

4.7 The DATA: Delimiter characters

The default delimiter between data values is a comma (,) or if the data are to be entered from the terminal, a blank. You can supply your choice of delimiter with the DELIMITER (DEL) identifier. If you supply a delimiter, there are three things that can define the area of a value: 1) the defined delimiter; 2) an end-of-case character; and 2) the “$” which ends the command. Blanks are no longer considered a delimiter and are handled as part of the data stream.

The delimiter can be one or more characters. Use of a delimiter is usually unnecessary when you are entering the data from the terminal. It can, however, be very useful when the data are in an external file and particularly if it has come from the web where you have little control over what is there.

Figure 4.3 illustrates the use of a delimiter character and the effect on the data. In the first example DEL is not specified and either or both the comma and the blank serve as delimiter. The result is 5 variables with no missing data. When a delimiter is explicitly specified the use of the blank as a delimiter is dropped. In the second example the delimiter is the comma and

```
1, 2, 3 4, 5   results in the values
1  2  -- 5 -
```

Figure 4.4 MAKE: Multiple Character Delimiter

```
   MAKE work, VARS Id Name:c, DEL 'DEL';

 begin case 1, Name (c40):
 23 del Harrison $

-------------MAKE completed--------------
| P-STAT file work has been created.    |
| It has 1 case and 2 variables.        |
|                                       |
| 3 delimiter bytes were used: D, E, L. |
-----------------------------------------

 LIST work $

     Id   Name

     23   Harrison
```

The first 2 variables in Figure 4.3 are entered cleanly with a comma delimiter. The “3” and “4” are separated only by a blank which is no longer available as the delimiter. Thus the value “3 4” which is not a legal number is set to MISSING 2 for var3. Since the final number “5” is followed by a “$” the case is considered complete: var4 has the value 5 and the final entirely missing var5 is given the value of MISSING 1.

The delimiter is not limited to a single character. The following two examples are perfectly legal:

```
DEL 'del'   or  DEL 'xxx99'
```

Figure 4.5 MAKE: Interactive Prompts and End of Case

```
A Slash is the default end of case character.

     MAKE work, NV 5;

     begin case 1, var1:

     1 1 1 / 3 4 5 6 7 $

     LIST work $

     var1  var2  var3  var4  var5

        1     1     1     -     -
        3     4     5     6     7

In this example the final '$' serves to complete the command
and to end the final case.

     MAKE work;

        Definitions (i.e., the names and types)
        of the variables are expected now.
        A response of just H or Q, at any time,
        will provide HELP, or will QUIT the command.

        The definitions should end with a semicolon.

        Begin entering variable names,
        or (H for HELP) or Q (for QUIT):

     var1 to var5;

     begin case 1, var1:
     1 1 1 3 4 5 6 7 $

     ----------------MAKE completed---------------
     | P-STAT file work has been created.         |
     | It has 2 cases and 5 variables.            |
     |                                            |
     | Two delimiters were used: BLANK and COMMA. |
     ----------------------------------------------

     LIST work $

      var1  var2  var3  var4  var5

         1     1     1     3     4
         5     6     7     -     -
```

as long as the delimiter does not occur outside of quotation marks in the data. Figure 4.4 illustrates the use of a three character delimiter. More complex strings are allowed. These are described in the manual “P-STAT: File Management.”

4.8 The DATA: End of Case Character

If the input data are a stream of numbers and character values, the MAKE command does the best it can in allocating the numbers to the proper variables. You can control this by the use of an end of case (EOC) string. The assumed end of case is the slash (/). The EOC can be a complex string rather than a single character. This is described in the manual “P-STAT: Importing and Exporting Your Data”. You can turn EOC off by using the NO.EOC identifier.

Figure 4.5 illustrates the use of an end of case character and provides an example of missing data. The slash (/) is the default end of case character. If the case does not have all the specified variables, it is padded with missing data. The missing data values which are stored internally as extremely large negative numbers are indicated by the presence of '-' in the LIST command printout.

You can provide an alternate end of case character. You can also specify that there is no end of case character.

```
MAKE work, NV 5, EOC '|' $
```

Figure 4.6 MAKE: Missing Data

```
MAKE work, NV 5, MISSING ’+’;

  begin case 1, var1:
  1 2 3 ++ 5 $

      -----------------MAKE completed-----------------
      | P-STAT file work has been created.           |
      | It has 1 case and 5 variables.               |
      |                                              |
      | Default delimiters (blank, comma) were used. |
      ------------------------------------------------

  Enter a command:
  LIST work $

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


  var1  var2  var3  var4  var5

     1     2     3    --     5
```

```
MAKE work, NV 5, NO EOC;
```

4.9 Missing Data

The assumed missing data character is the dash (-). A single dash is set to MISSING 1. Two consecutive dashes are set to MISSING 2 and 3 dashes to MISSING 3. The MISSING identifier requires a single character:

```
MISSING '+'
```

This sets the missing character to the plus sign and the exact type of missing depends on whether there are 1, 2 or 3 plus signs. Figure 4.6 illustrates the use of the MISSING identifier to change the default missing value.

Notice in Figure 4.6 that the "++" appears in the LIST command as "--". This is because the real values for the 3 types of missing are very large negative numbers. The "++" that you provide is converted by MAKE into such a value. The LIST command only knows that it is printing the very large negative number of MISSING type 2 and represents it as "--". This makes a better looking listing than printing the internal representation.

NO.MISSING is an identifier that specifies that there is no special missing data character.

4.10 OTHER FEATURES

The MAKE command has many other identifiers which are described in full in "P-STAT: Importing and Exporting Your Data". The following is a list and brief description of these identifiers.

1. STRIP, CHANGE and LEFT.JUSTIFY are used to modify the input characters for each variable as they are being read, before the value is moved into the resulting P-STAT system file.

   STRIP removes the quotes or apostrophes from an input bounded string.

   CHANGE phrases modify an input value one value at a time for every variable in every case in the file. There can be many change phrases.

   LEFT.JUSTIFY, if requested, is done after STRIP and CHANGE have finished editing the value.

2. BLANK.DELIMITER when used with DELIMITER causes a blank to also end a value. Lead blanks are ignored when this is in use.

3. NO DELIMITER is used when each input record is a data value and the carriage return or line feed serves as an end of value indicator.

4. DROP provides a way to remove unwanted characters from values that are not bounded strings. The language is the same as that for EOC and DEL, however CRLF is treated differently.

5. FILESIZE can be used to control the number of bytes that will be used from the input file. If positive, use that many bytes. If negative, use all but those.

6. HONOR.STRINGS: if honor.strings is used and the first nonblank of the value is a string boundary, we do not look for the EOC until the string ends.

7. IGNORE.STRINGS: if ignore.strings is used, the string boundary does not determine the end of the value.

8. STRING.BOUNDARY can be set to "APOSTROPHE", "QUOTE" or "BOTH". BOTH is assumed.

9. ONE.PASS applies only to external file input. Usually a first pass looks at the data to determine the largest value for each character variable. If ONE.PASS is used, the program uses the defined size of the values and longer character values are truncated..

SUMMARY

MAKE

```
Make Test, NV 5;
1 1 - 2 1
3 1 2 3 1
$
```

The MAKE command is used to process delimited data either from an external file or from the terminal. It creates a P-STAT system file and optionally a summary file. The command needs to know the name for the file to be created; the variable names and data types; and where the data are located. The variable names and data can be inline or in external files.

The variable names can be entered as subcommands or as part of the command with the VARS identifier.

```
Make Test;                      MAKE Test, VARS Item01 to Item05;
Item01 to Item05;
1 1 - 2 1                        1 1 - 2 1

etc. $                           etc.  $
```

Both the data and the definitions can be in external files:

```
MAKE Test, FILE 'test.txt',  DEF 'testdefs.txt' $
```

Required Identifiers:

MAKE fn

specifies the name for the P-STAT system file to be created.

Required: if the data are not inline:

FILE fn

specifies the name of the file which contains the data values.

Required: if the definitions do not follow the command as subcommands, one and only one of the following.

DEFINITIONS fn

specifies an external file containing the variable names and data types. DEF can also be used.

NV nn

specifies the number of numeric variables in the file. Variable names are generated.

VARS vn vn:c vn ....

is followed by a list of the variable names and data types

TEMPLATE fn

specifies and existing P-STAT system file. The variable names and data types are taken from this file.

Optional Identifiers:

MISSING 'c'

A single character to represent missing data. A single character represents MISSING type 1. A double character represents MISSING type 2. A triple of the character represents MISSING type 3.

DEL 'cs'

One or more characters that delimit a value.

NO.DEL

This specifies that there is no delimiter between values. This can only be used when there is only a single variable and the end of case provides the delimiter.

EOC 'cs'

One or more characters that define the end of a case.

NO.EOC

Tis specifies that there is no end of case character. The case is considered complete when the expected number of values has been processed.

SPSS.IN

```
SPSS.IN pstatfile, FILE “spss.data” $
```

Required Identifiers:

SPSS.IN fn

provides the name for a new P-STAT system file

FILE “fn”

provides that name of a file in SPSS “portable” format.

Optional Identifiers:

LABELS “fn”

provides the name for a P-STAT labels file to be created from the label information in the portable format file.

SYSTEM.MISSING nn

Provides the number (1, 2, or 3) indicating which of P-STAT’s 3 missing values should be used to represent the SPSS system missing value.

TEXTFILE.IN/ TABFILE.IN

```
TEXTFILE.IN pstatfile, FILE "datafile.txt"$
TABFILE.IN  pstatfile, FILE "datafile.txt", DELIMITER COMMA $
```

TEXTFILE.IN and TABFILE.IN are the same command. They are used to read a delimited ASCII data file and create a P-STAT system file.

Required Identifiers:

TEXTFILE.IN fn
TABFILE.IN fn

provides the name for a new P-STAT system file.

FILE "fn"

provides the name of the external file which contains the input data.

LABELS or NO LABELS

specifies whether the first record of the input file contains variable labels or data. LABELS is assumed if neither is specified.

Optional Identifiers:

DELIMITER 'cs' or one of TAB COMMA BLANK

This provides a value delimiter character to be used instead of the default horizontal tab. DEL can be used instead of DELIMITER.

The argument can be a single character in quotes, or the ascii value (0-255) of the character, or the names BLANK, COMMA or TAB.

CHARACTER vn vn to vn

is followed by the names or ranges of variables that are to be typed character in the P-STAT file even though they appear to be numeric.

NUMERIC vn vn to vn

is followed by the names or ranges of variables that are to be typed numeric in the P-STAT file even though they appear to be character.

DECIMAL.COMMA

Use a comma in numbers instead of the decimal point. This is a common practice in many countries.

EXTENDED.LABELS fn

provides a file name for a labels file which associates the original variable text with the name that follows the rules for P-STAT legal variable names

HONOR.STRINGS

When a string is found and HONOR.STRINGS is in effect, any string enders are ignored until the string has ended

IGNORE.STRINGS

is used to turn off string processing entirely.

LABEL.CHANGES fn

creates a new P-STAT system file with one row for each variable containing the variable name which P-STAT supplies and the original name from the input file.

STRING.BOUNDARY APOSTROPHE QUOTE QUOTATION MARK BOTH EITHER

Used to test if the first non-blank character of a value is the beginning of a string.

5
Labels and Titles

Titles and labels are necessary parts of a useful and attractive listing or table This chapter covers the basic labels formats, the commands which save and check the labels, the TITLES command and the general identifiers which specify how 64 character variable names should be displayed.

A LABEL is a character string, like “Mini Van”, to be used in a report instead of a less meaningful value, like 7 or 'MV'. These labels, perhaps many hundreds of them, are stored in one or more external text files. A labels file can also hold extended labels for the names of the variables. With 64 characters available for variable labels, there will be situations where extended variable labels are no longer needed. General identifiers such as: FULL, TEXT, SHORT and SHORT.16 can be used to specify the desired format..

A LABELS file can be created within P-STAT by the SAVE.LABELS command. Alternatively, it can be created and changed by programs like MS WORD, or by editors like NotePad or vi. If it is created in a word processing program such as MS WORD rather than a text editor, it must be explicitly saved as ASCII text before P-STAT can use it. Commands like LIST and SURVEY may read the labels file during a run. The file is not changed by such usage.

A TITLE is one or more lines of text that are written at the top (or perhaps bottom) of each page of a report. Title text is not maintained in an external file. Instead, it is simply defined in a TITLES command, which stores the titles text to be used in the run.

- The SAVE.LABELS command accepts labels for categorical or missing values and extended labels for variable names, saving them in an external file for subsequent use. The LABELS identifier may then be used in commands such as LIST, SURVEY and TEXTWRITER to request that labels in the designated external file be used to replace actual values.
- The CHECK.LABELS command checks an existing labels file for appropriate syntax and punctuation.
- The TITLES command defines multiple top and bottom titles for complete identification of listings, reports and other output. The TITLES identifier may be used in commands such as LIST, SURVEY and PLOT to cause the defined titles to print on each page of output.

The next chapter describes the LIST command and illustrates how reports with titles and labels are produced.

5.1 TAGS AND/OR TEXT

In version 3, variable labels have much more flexibility than in any previous release. and there are some general identifiers that specify how these new labels are to be displayed. The SHOW command is a utility that displays the variables in a P-STAT system file. In Figure 5.1 the SHOW command is used to display the variable names in different formats.

The general identifiers that can be used to specify how to display variable names are:

```
FULL       TEXT       SHORT or SHORT.TAGS
SHORT.16   SHORT.OLD
```

Figure 5.1 SHOW Command With SHORT (TAGS) and TEXT

```
SHOW test $

    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


 SHOW test, SHORT $

   1 N id                    6 N VCR
   2 N Age                   7 N CD
   3 N Sex                   8 N Port.Phone
   4 N Income                9 N Ans.Mach
   5 N Num.TV               10 N Store.1


 SHOW test, TEXT $

    1 N    id
    2 N    Age_of_respondent
    3 N    Gender_of_respondent
    4 N    Income
    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


  SHOW test, SHORT.OLD $

   1 N id                    6 N VCR.Owns.a.VCR.o
   2 N Age.Age.of.respo      7 N CD.Owns.one.or.m
   3 N Sex.Gender.of.re      8 N Port.Phone.Prima
   4 N Income                9 N Ans.Mach.Telepho
   5 N Num.TV.Number.of     10 N Store.1.Where.wa
```

These 5 general identifiers can be used in many of the commands in Version 3. In addition the SURVEY command support these settings at the subcommand level with the SET.LABELS subcommand. There are some statistical commands with matrix output where 64 characters of label for a row or column are not possible and SHORT.TAGS are always used.

5.2 VARIABLE AND VALUE LABELS

Value labels and extended variable labels contribute to the clarity of information in output. *Value* labels are used in listings and surveys in place of numeric or character values; extended *variable* labels are used in place of variable names. Labels of either type are read from external disk files. The SAVE.LABELS command writes label files. Alternatively, an editor or word processing package can be used outside of P-STAT to write labels in a disk file. Regardless of how the labels are written in the file, the format of the labels is the same.

Value labels for numeric variables are supported in the following commands:

```
ANOVA       BOXPLOT     COUNTS         LIST
MAP         PLOT        SPSS.IN        SPSS.OUT
SURVEY      TEXTWRITER  SUBSTITUTE.VL
```

Value labels for character variables are supported in:

```
LIST        MAP         SPSS.IN        SPSS.OUT
TEXTWRITER  SUBSTITUTE.VL
```

Extended variable labels are used by:

```
BOXPLOT     LIST        MAP            PLOT
SPSS.IN     SPSS.OUT    SURVEY         SUBSTITUTE.XL
```

The LABELS identifier requests that value labels replace values in the current command's output, and it gives the name of the label file:

```
SURVEY  MR124,  LABELS 'MR124.lab' ;
```

The USE.XL identifier in the LIST command requests that extended variable labels replace the variable names:

```
LIST    Trout,  LABELS  'TroutLab',  USE.XL  $
```

The SURVEY command automatically uses any extended variable labels in the label file — USE.XL is not necessary. This makes it possible to have a single labels file with both types of labels for surveys, but to use only value labels in listings where there is less room in column headings for extended variable labels. (See the LIST chapter and the first chapter in the manual "P-STAT: The Survey Command" for additional information on labels. Each command uses different features of the labels capability. For example, LIST supports labels for character variables while the SURVEY command does not.

5.3 Value Labels

Value labels are character strings that replace the values of a variable. The format for value labels is that each variable name is followed by a series of *values*, which must be in parentheses, and *labels*, which may be up to 80 characters long including any string boundary characters. The permissible string boundary characters are the quotation mark ("), the apostrophe (') and paired angle brackets(<< >>). The labels for each variable end with a slash (/). A dollar sign ($) indicates the end of all labels:

```
Sex        (1) Male         (2) Female   /
Education  (1) High School  (2) College  /  $
```

The label text need not have string boundary characters unless it contains a left parenthesis, a slash or begins with a string boundary character that is intended to be part of the actual label. Boundary characters are also needed

to supply a blank label or leading blanks in a label. You may not include the chosen delimiter as part of the text. The following examples are all acceptable.

```
Q1  (1) "it's ok" (2) <<it's "so-so">> (3) "it's <= 33" (M1) << >> /
```

Each of the following labels is incorrect and will cause an error message:

```
Q1  (1) 'it's ok' (2) milk and/or cheese  (3) ten (or more)  (M1)    /
```

The fourth erroneous label is the one, after (M1), that is not there at all.

The angle brackets are particularly useful when proofreading labels text. It is easier to see that there is a pair of matching right brackets for each pair of left brackets than it is to see the matching quote or apostrophe. It is also easier for P-STAT to detect and provide error messages for the angle brackets.

Labels can be provided for integers, missing values, real numbers, and for character values. However, many commands only use labels for integers and missing values.

```
Q1 (1) yes (2) no (M1) No Answer (M2) Not Applicable (M3) Don't know
```

M1, M2, and M3 represent the three system missing values.

Character values which are enclosed in parentheses must also be enclosed in quotes or angle brackets and may be no more than 78 characters long:

```
Gender ('m') male ('f') female /
```

Long character strings, whether value labels, character values or extended labels should not be broken across 80 character records. Instead multiple records should be used:

```
Q1 (1)
<<This is the label for value 1 of question Q1.It is a long label!>>
```

The match between the value in the P-STAT system file and the value in the labels file is usually case independent. This can be controlled by adding an "X" before the label:

```
State ( X'NJ') New Jersey ....
```

Only those cases with a value that exactly matches "NJ" get the New Jersey label. The value "nj", for example, will not be a match.

When there are several variables which have identical value labels, the labels need not be re-specified. The variable names may be provided as a list followed by the values:

```
Quest.1   Quest.2   Quest.8  TO  Quest.12   Purchase
(1) Yes    (2) No    (3) Undecided  /
```

The same labels are used for variables Quest.1, Quest.2, Quest.8 through Quest.12 and Purchase. The slash indicates the end of the value label definitions. That set of definitions becomes the most recent set of value labels.

Instead of providing a list of variables before the labels are defined, the most recent set of labels can be reused by including an “@” after the variable name or variable list. To assign the most recent set of labels to a list of variables use this format:

```
Q1A TO Q13B @ /
```

The "@" sign followed by a variable name can be used to indicate that the value labels associated with that previous variable should also be used for the current variable:

```
Quest115 @ Quest1 /
```

requests that the labels associated with variable Quest1 be used for Quest115. The Quest1 labels must have been defined or an error will occur.

Once the labels are written in an external file, they may be referenced with the identifier LABELS, followed by the name of the label file in single or double quotes:

```
LIST  Staff,   LABELS  'LabFile'  $
```

In this example, the file named Staff is listed using the labels that have been written in the external file named LabFile.

Figure 5.2 Labels File: Two Ways of Organizing

Intermixed extended variable labels and value labels

```
Age ( 1 ) 'Under 30' ( 2 ) '30 to^50' ( 3 ) 'Over 50' /
Sex ( 1 ) Male ( 2 ) Female /
Num.TV  <<Number of television sets in your home:>>
    ( 5 ) '5 or more' /
VCR   <<Video Cassette Recorder>>
      <<Which of the following do you own?>>
    ( 0 ) No (1) Yes /
CD    <<Compact Disc Player>>        @ /
Port.Phone   <<Portable Telephone>>  @ /
Ans.Mach     <<Answering Machine>>   @ /
Store.1      <<Type of store item purchased in:>>
    ( 1 ) Dis*count ( 2 ) Depart*ment ( 3 ) Audio, Elec*tronic /
Income.Groups  <<Income of respondent>>
    ( 1 ) Under $20,000 ( 2 ) $20,000 to $40,000 ( 3 ) Over $40,000 /
Own   <<Respondent ownership>>  <<& &>>
      <<of electronic items>>
    ( m1 ) No Reply /
```

Separated extended variable labels and value labels

```
Num.TV       <<Number of television sets in your home:>> /
VCR          <<Video Cassette Recorder>>
             <<Which of the following do you own?>> /
CD           <<Compact Disc Player>>   /
Port.Phone   <<Portable Telephone>>  /
Ans.Mach     <<Answering Machine>>  /
Store.1      <<Type of store item purchased in:>> /
Income.Groups<<Income of respondent>> /
Own          <<Respondent ownership>>  <<& &>>
             <<of electronic items>> /

Age     ( 1 ) Under 30 ( 2 ) 30 to^50 ( 3 ) Over 50 /
Sex     ( 1 ) Male     ( 2 ) Female /
Num.TV  ( 5 ) 5 or more /
VCR   CD   Port.Phone   Ans.Mach
        ( 0 ) No (1) Yes /
Store.1 ( 1 ) Dis*count ( 2 ) Depart*ment
        ( 3 ) Audio, Elec*tronic /
Income.Groups ( 1 ) Under $20,000
        ( 2 ) $20,000 to $40,000 ( 3 ) Over $40,000 /
Own     ( m1 ) No Reply /
```

5.4 Break Characters

Break and non-break characters are a feature of value labels used by the SURVEY command. They help format the column labels which must often be placed in a six column by six row area. The break character is assumed to be the asterisk. It occurs in the middle of a word and is a suggested break point if the word will not fit. The non-break character is usually the caret or not sign. It occurs between words instead of the blank and is a request not to break at this point if possible.

Figure 5.2 has examples of both the break and non-break characters:

```
( 1 ) Dis*count ( 2 ) Depart*ment ( 3 ) Audio, Elec*tronic
```

Without the use of break characters the column headers in a survey could well be:

```
                  Audio,
Disc   Depar    Elect
ount   tment    ronic
```

With the break characters, the column headers are printed as:

```
                  Audio,
  Dis  Depart    Elec
count    ment  tronic
```

The non-break break character is illustrated in:

```
( 1 ) Under 30 ( 2 ) 30 to^50 ( 3 ) Over 50
```

Wherever possible a label is broken at a blank. There is also an attempt to place as much as possible on a single line. Thus the default for the “30 to 50” label is to break after the “to”. The use of the no-break character is to force the break before the “to”. Break characters are stripped out when labels are used by commands such as LIST which do not use them.

5.5 Extended Variable Labels

Extended variable labels are character strings of up to 78 characters that replace the names of variables. Not all of the commands which use value labels also use extended labels, and the actual length that is used depends on the command and the situation.

The format of extended variable labels is simply a string in quotes or angle brackets immediately following one or more variable names. If there are multiple names they will share the same extended label.

```
Var1 Var2 Var3  <<extended variable label>> /
```

A slash indicates the end of the labels for a variable or variable list, and a $ indicates the end of all labels for all variables. Value labels may follow the extended variable label:

```
Variable.Name  <<extended variable label >>
  ( value ) label  ( value )  label ...          /
```

or the extended labels and variable labels may be entered separately:

```
Quest1 "How do you rate your senator's performance?" /
Quest2 "How do you rate your mayor's performance?" /

Quest1 Quest2 (1) excellent (3) mediocre (5) awful /
```

If the *value* labels for a variable are the same as those of the previous variable, you can use the “@” notation to point to the previous variable. The value labels need not be repeated:

```
Income1 <<Head of Household:  Which income category are you in?>>
(1) Under 20,000      (2) 20,000 to 30,000  (3) 30,000 to 40,000
(4) 40,000 to 50,000  (5) 50,000 to 60,000  (6) 60,000 to 70,000
```

```
(7) Over 70,000 /
Gender (1) Male (2) Female /
Income2 <<Spouse:  Which income category are you in? >> @Income1 /
```

There can be, in theory, an unlimited number of extended labels for a single variable. Each label is enclosed in quotes or paired angle brackets. Each label must be 76-78 characters or less. You can put 2 or more extended labels on a single record as long as they fit completely. Any single label should not be broken in the middle. SURVEY and SUBSTITUTE.XL are currently the only commands which makes use of these extra labels. A special feature is the <<&&>> and <<& &>> labels which are used to indicate concatenation. These labels are not printed. They serve as instructions to the command that is using the labels. Commands which do not use the extra extended labels simply ignore them when they appear in the labels file.

```
Q1
<<This is the first line of text for question Q1  The ampersands >>
<<&&>>
<<indicate that the lines should be joined without an added blank>>

<<This is more text for question Q1.  There will be>>
<<& &>>
<<a blank inserted before it is joined with this line>> /
```

5.6 Multiple Labels Files

Any command which uses labels can process multiple labels files. This makes a variety of arrangements possible depending on your needs. A single file might be organized so that each variable in the file is defined in full with its extended label followed by its value labels. A variation on this is the use of the @ varname to avoid the repetition of identical value labels. This is more economical in terms of both human and computer resources, but if the file has thousands of variables, it makes it more difficult to look at the labels for any single variable.

Figure 5.2 illustrates two arrangements of the same information. The first arrangement has each variable followed by its own extended variable names and value labels. The second arrangement has all of the extended variable name first, followed in the same file by all of the value labels.

Another useful organization has the extended labels in one file and the value labels in a second file. If the file is heavily used by both LIST and SURVEY which support different features for the extended labels, you might have two files of extended labels. The order in which they are given to the command determines which label wins when a given variable has labels in more than one file. The rule is that the last one wins.

Another useful organization is to have a labels file for variables that are usually included in your projects. For example, a file of demographic information might have standard labels for occupation, education, etc. If a variable in the labels file is not one of the variables in the P-STAT system file it is ignored. If a standard variable is recoded for a particular project, its new definition would be included in the labels for the new project.

```
List Project9,  LABELS 'demog.lab' 'Project9.lab' $
```

Any variable with value labels in both files will get the value labels from 'Project9.lab'. If the value labels are present in both files and the extended label is only present in the first file, then the extended label is taken from the first file and the value labels from the second file.

5.7 SAVE.LABELS

The SAVE.LABELS command is a utility command which makes an external disk file of value labels and/or extended variable labels. When that file is referenced in a subsequent command, the labels replace values and/or variable names in the output produced by that command. SAVE.LABELS checks the format of the labels and, when PSTAT.FILE is used to specify the corresponding P-STAT system file, reports the number of variables for which labels are provided.

SAVE.LABELS requires a name for the external file in which the labels are to be written. The name is supplied within single or double quotes:

```
SAVE.LABELS 'Psfile.lab';
```

The labels are supplied as subcommands after the command. Either value labels or extended variable labels or both types of labels may be supplied in a single SAVE.LABELS command.

After value and variable labels are written with the SAVE.LABELS command, their usage is requested by including identifiers such as LABELS and USE.XL in appropriate commands:

```
LIST Psfile,   LABELS 'Psfile.lab', USE.XL $
SURVEY Psfile, LABELS 'Psfile.lab';
```

Figure 5.3 Saving and Using a Labels File

```
SAVE.LABELS  'LabFile', PSTAT.FILE Myfile ;

  Sex   <<Sex of Respondent>>
        (1) Male         (2) Female        /
  Education  <<Highest Schooling Completed>>
        (1) High School  (2) College       /
  Income <<Total Family Income>>             /
  Residence
        (1) Apartment (2) Single family (3) duplex    /
  $

THE REPORT WHEN PSTAT.FILE IS USED

  7 records were processed.
  The file will now be checked for errors.
  No errors were found.

  2 extended variable labels and 7 value labels
  were found for the variables in the P-STAT file.


LIST  Census,  LABELS  'Census.lab',  USE.XL  $
```

Figure 5.3 illustrates the SAVE.LABELS command with the PSTAT.FILE identifier and a LIST command which uses the label information. The LABELS identifier: 1) gives the name of the external file of labels, and 2) requests that value labels replace the associated values wherever possible. The USE.XL identifier requests that extended variable labels replace the variable names. Both types of labels may be supplied for some or all of the variables in a P-STAT system file, and for some or all of the values of a variable.

5.8 Checking Labels

After the SAVE.LABELS command writes the labels in the specified external file, the labels are checked for correct format. Missing slashes, unclosed quoted strings, and unknown characters within the parentheses are identified and reported. To turn off the automatic checking of labels, include NO CHECK in the SAVE.LABELS command:

```
SAVE.LABELS  'MeatLamb.lab',  NO CHECK ;
```

CHECK is assumed.

The P-STAT system file that corresponds to the labels file — the one with which the labels are to be used — may be input to SAVE.LABELS using the PSTAT.FILE identifier:

```
SAVE.LABELS  'MeatLamb.lab',  PSTAT.FILE  LambStat ;
```

When this is done, the number of variables for which value and/or variable labels are provided is reported.

You do not need to use the SAVE.LABELS command to enter your labels. Any editor that creates an ASCII file can be used. The one advantage of SAVE.LABELS is that it automatically checks the labels for syntax and balanced parentheses. If you have entered the labels using an external editor, the CHECK.LABELS command may be used to check the format of one or more existing label files. Its arguments are the quoted names of the label files. The corresponding P-STAT system file may be supplied, if desired:

```
CHECK.LABELS  'MeatPork.lab',  PSTAT.FILE  PorkStat  $
```

CHECK.LABELS checks the format of labels and counts the number of variables in common between the system file and label files, the same way that SAVE.LABELS does. You should always check the format of your labels before using them in LIST, SURVEY, etc.

5.9 TITLES

Titles are defined using the P-STAT *command* TITLES:

```
TITLES  T1  'Top Title' ,
        B1  'Bottom Title'  $
```

and they are “turned on” by using the *identifier* TITLES in P-STAT commands that produce output:

```
LIST  BookFile,  TITLES  $
```

TITLES (or TITLE) may be used to define up to nine top titles and three bottom titles for each page of P-STAT output. Individual control is provided for the left, center, and right sections of each title. Titles are printed whenever a P-STAT command that supports titles both 1) incorporates the identifier TITLES and 2) issues a page change.

Top titles are always at the top of the physical page. The placement of bottom titles depends on the relation of the LINES setting to the true page size. The page size of output directed to a printer or disk file is controlled by the LINES setting. This is set to 59 at the start of a P-STAT run but can be changed with the LINES command:

```
LINES 45  $
```

by setting lines in a PRINT.PARAMETERS command, or by using LINES as a general identifier within any P-STAT command:

```
LIST  BookFile,  TITLES,  LINES 45,  PR  'LPT1'  $
```

When LINES is used as a *command*, it remains in effect for *all* subsequent commands whose output is directed to a printer or disk file. When LINES is used in PRINT.PARAMETERS, it affects only the specific print destination. When it is used as an *identifier*, it is in effect *only* for the command in which it is included and overrides any previous settings. The SCREEN *command* sets the number of lines used for output directed to the terminal: Titles are controlled by the LINES setting, not by the SCREEN setting. A long printout directed to the terminal fills the screen and then waits for a carriage return before printing the next screen. Top and bottom titles print as they will when the printout is directed to a device controlled by the LINES setting.

5.10 Defining Titles

In its simplest form, TITLE or TITLES requires a character string containing the title information bounded by a pair of single or double quotes. This defines a single title:

```
TITLE  'P-STAT Initial Data Run'  $
```

It is equivalent to:

```
TITLE  T1  'P-STAT Initial Data Run'  $
```

"T1" specifies that this title is defined for the *first* line of the top of the page. When just TITLE is used, the text is assumed to be defined for the first line of titling. To provide a *second* line of title information, the identifier T2 is required:

```
TITLE  T2  '1-way FREQ –  Numeric Variables'  $
```

If the previous two titles are defined and the TITLES identifier is used in a command, both titles appear on the top two lines of each page of output:

```
                P-STAT Initial Data Run
          1-way FREQ - Numeric Variables
```

A portion of the title information may be replaced or redefined subsequently in a run. The command:

```
TITLE  T2  'Regression Predicting SAT Scores'  $
```

replaces the previously defined second line of titling as follows:

```
                P-STAT Initial Data Run
          Regression Predicting SAT Scores
```

If title two is defined when title one has *not* been defined, the first title prints as a line of blanks. However, if only the first two titles are defined, only two lines of titles print. If title one and title three are defined, three lines print. The second is a line of blanks. An individual title may be explicitly set to blank by using the keyword BLANK or BLANKS:

```
TITLE  T3  BLANK  $
```

This sets the third title to a line of blanks. Any previous contents are replaced with blanks. Note that the keyword BLANK does *not* have quotes around it. If it did, the word BLANK would print in the center of the second line of titling.

Multiple lines of titling may all be supplied in a single TITLE command:

```
TITLES  T1  'Put this title first'  ,
        T2  'Put this title second' ,
        B1  'Put this title at the bottom'  $
```

"T" followed by a number indicates which top title follows. T1 through T9 are allowed. Optional commas may be used to separate the lines of titling. "B" followed by a number indicates which bottom title follows. B1, B2 and B3 can be used. These titles are all centered within the line defined by the current *page width*.

Page width is set by the OUTPUT.WIDTH (OW) command and general identifier. An output width of 80 is assumed for the terminal and 132 for printers and disk files. Use OUTPUT.WIDTH as a command or identifier to reset page width:

```
LIST  BookFile,  TITLES,  OUTPUT.WIDTH 80,  PR 'LPT1'  $
```

(See the chapter in the P-STAT Manual "Utility Commands" for more information on defining page widths for the terminal and for print destinations.)

5.11 Justifying and Centering Titles

The keywords LEFT, CENTER and RIGHT are used to specify where title text is to be placed. When they are not used, the title text is centered. The command:

```
TITLE  'Top Title 1'  $
```

is equivalent to:

```
TITLE  T1  LEFT    '           ' ,
           CENTER  'Top Title 1' ,
           RIGHT   '           '  $
```

The text for a left title is left justified in the line. The text for a right title is right justified in the line. The center title is centered. If the three separate strings are longer than the length of the line, they overlap one another. LEFT may be shortened to L, RIGHT to R, and CENTER to C. Commas are optional between the left, right and center sections of titling.

Any section of any title can be replaced without disturbing other sections previously defined:

```
TITLE  T1  L  'Personnel'  C  'Males',   R  '1984-85'  $
LIST   Staff
  ( IF Sex =  'Male',     RETAIN ),  TITLES  $

TITLE  T1  C  'Females'  $
LIST   Staff
  ( IF Sex =  'Female',   RETAIN ),  TITLES  $
```

Any portion of a line of titling may also be set to blanks:

```
TITLE  T2  C  BLANKS  $
```

5.12 Making Titles with Borders Using FILL

Sections of any title that are not defined are set to blank. These two commands are equivalent:

```
TITLES  T1  C  'Chemistry 101'  $
TITLES  T1  L  BLANK   C  'Chemistry 101'   R  BLANK  $
```

The undefined sections of any title and any space between defined sections may be filled with a specified character, rather than with blanks, to create a border or divider. The identifier FILL is used:

```
TITLES  T1  L  '101'    C  'Chemistry 101'    R  '101'  FILL '=' ,
        T3  L  'Mr. Shaw'   R  'Room 143'  FILL '.'  $
```

Only a *single* character may be designated as the FILL character, and only the line of titling specified previous to the identifier FILL is filled with that character. Commas may be used to separate the left, right and center sections of titling and the FILL character definition:

```
TITLES  T1  L  '101',   C  'Chemistry 101',   R  '101',  FILL '=' ,
        T3  L  'Mr. Shaw',  R  'Room 143',  FILL '.'  $
```

The FILL character for a given line of titling remains as specified, even when the text strings for that line are changed. The FILL character must be explicitly reset to a blank:

```
TITLES  T3  FILL  ' '   $
```

Now the third line of titling just defined does not have any FILL other than blanks.

5.13 Examine Titles With The SHOW Identifier

The identifier SHOW may be incorporated at the *end* of any TITLES command to show all currently defined titles. Figure5.4 contains the definitions for 4 top titles and a bottom title, with the addition of the identifier SHOW:

This command:

```
TITLES,  SHOW  $
```

shows the same defined titles again. The SHOW identifier lets you check how any defined titles look in print. The top and bottom titles are identified separately but there is no printout between them and you cannot see the effects of the LINES setting. The SHOW identifier should be the final identifier in the TITLES command to en-

sure that all the titles are displayed. The page number is zero because no real pages have yet printed in the current command.

Figure 5.4 Defining and Showing TITLES

```
  TITLES  T1  L  '101',   C  'Chemistry 101',   R  '101',  FILL '=' ,
          T3  L  'Mr. Shaw',  R  'Room 143',
          T4  C  'Class List'  FILL '-' ,
          B1     'Page .PAGE.',
          SHOW  $

TOP:
101============================Chemistry 101============================101

Mr. Shaw                                                        Room 143
--------------------------------Class List--------------------------------

BOTTOM:
                                    Page 0
```

5.14 Turning Titles OFF and ON

Defined titles exist, but they appear on output only when the identifier TITLES is used in specific commands that produce output. Defined titles remain in effect until:

1. replacement titles are defined,
2. the titles are explicitly turned off, or
3. the titles are reset.

TITLES may be turned off by using the keyword OFF:

```
TITLES  OFF  $
```

No titles print now, even when the TITLES identifier is used in a LIST or SURVEY command. The titles that were previously defined continue to exist; they are merely turned off. They may be turned back on by using the keyword ON:

```
TITLES  ON  $
```

If the TITLES identifier is used in an appropriate command all existing titles can now print,

The *number* of titles that print can be selectively controlled. Specific lines of titling may be turned off or on:

```
TITLE  T5  OFF  $
```

This turns off all titles from the fifth one through the last. Titles 1 through 4 continue to print.

Defined titles may be “undefined” individually or simultaneously with the keyword RESET:

```
TITLE   T3  RESET  $     or
TITLES  RESET      $
```

Titles that have been reset no longer exist. Thus, they may not be turned back on — they have to be defined again.

5.15 Using System Variables in Titles

There are numerous P-STAT system variables that improve the usefulness of titles. When these variables are encountered in a title, the current system value is substituted. For example:

```
TITLE  T1  L  'File  .FILE.'    R  'Page  .PAGE.' ,
       T2     'Describe the run in center of Title 2' ,
       B1  R  'Note:  This is on the Bottom Right'   $
```

The name of the current P-STAT system file is substituted for .FILE., and the current page number since the command began is substituted for .PAGE.

The system variables that are particularly useful in titles are as follows:

1. .DATE. This is the date set when the current command began, in character form. (It is equivalent to .CDATE., described below.)
2. .TIME. This is the time when the current command began, in character form. (It is equivalent to .CTIME., described below.)
3. .FILE. This is the name of the current file.
4. .PAGE. This is the current page number since the current *command* began.
5. .RPAGE. This is the current page number since the *run* began.

The system variables .DATE. and .TIME. may be prefaced with N, X, R or C:

```
.NDATE.                    .NTIME.
.XDATE.      .NXDATE.      .XTIME.      .NXTIME.

.RDATE.      .NRDATE.      .RTIME.      .NRTIME.
.CDATE.      .NCDATE.      .CTIME.      .NCTIME.
```

The N specifies the *numeric* form of the date or time, rather than the character form. The X specifies the *exact* date or time when the system variable is processed. The R specifies the *run* date or time — when the current run began. The C specifies the *command* date or time — when the current command began. The numeric form of exact, run and command dates or times may also be specified. The dates and times are printed as they are represented in the computer system on which P-STAT is being used.

The general identifier PAGE.NUMBER resets the system variable .PAGE. It has effect only when .PAGE. is used in defining titles and the TITLES identifier is used in a command:

```
LIST  Parts,  TITLES,  PAGE.NUMBER 8  $
```

Normally, .PAGE. is reset at the start of each command so that the first page will be page 1. The .PAGE. value, before the first page change, is therefore zero. Using PAGE.NUMBER 8 will cause the .PAGE. value at the first page change to become 8. If .PAGE. is used in a TITLE definition, it prints with the value specified after PAGE.NUMBER. The identifier PAGE.NUMBER may be used only with the TITLES identifier and in the commands that produce output, such as LIST, SURVEY and PLOT. The identifier RUN.PAGE.NUMBER resets .RPAGE. in a similar manner. NOTE: TITLES can also be used as a general identifier for many commands including LIST, SURVEY and PLOT.

5.16 Using Scratch Variables in Titles

Scratch variables can be used in titles. If the scratch variable is created in a command other than the command which requests the titles, it must be created as a permanent scratch variable. If the scratch variable is created in the command which requests the titles it may be either a temporary or a permanent scratch variable.

```
GENERATE  ##Month:C = 'November'  $
```

```
TITLES T1  'ABC, Inc.  Report for the Month of ##MONTH',
       T2  'District #District.No' $

LIST ABCAll [ IF District = 12, RETAIN ;
              GENERATE #District.NO = District ],
     TITLES $
```

The permanent scratch variable ##Month is created in a PPL command early in the run. The temporary scratch variable #District.No is created as the file is passed to the LIST command. Note that the second pound sign for the permanent scratch variable is required whenever it is used. The titles that are produced with the listing will have “November” substituted for ##Month and “12” substituted for #District.No.

5.17 Fonts for Titles

When PostScript is used for the printout the titles can be given different fonts. Any of 9 different fonts can be selected for the titles. The left, center and right sections of each title line must use the same font. The chapter “SURVEY: Enhancing Table Appearance” has a section on controlling the title fonts. Font support is also described in the manual “P-STAT: Plots, Graphs and PostScript Support”.

SUMMARY

SAVE.LABELS

```
SAVE.LABELS  'LabFile' ;

  Sex        <<Sex of Respondent>>
             (1)  Male         (2)  Female      /

  Education  'Last Schooling Completed'
             (1)  High School  (2)  College     /
  $
```

The SAVE.LABELS command specifies a name for an external file in which labels are to be written. The variable and value labels are provided at the subcommand level. The labels are checked for formatting errors unless NO CHECK is specified. When PSTAT.FILE is used to input the P-STAT system file with which the labels are to be used, the variables in common are noted.

Extended *variable* labels replace variable names in listings and tables. Their format is:

```
Variable.Name  <<Here goes a longer variable label>>
```

V*alue* labels replace numeric and, in some commands, character values. Their format is:

```
Variable.Name  (value) label  (value) label    /
```

Value labels and extended variable labels may be up to 80 characters long including any surrounding quotes. Not all commands that support labels can handle the full 80 characters. Check the documentation for individual commands for details. Each variable name is followed by a series of values and labels with the values placed in parentheses. The value labels for each variable name must end with a slash (/). A "$" signals the end of the labels.

A label file is referenced using the LABELS identifier in commands that produce labeled output, such as LIST and SURVEY:

```
LIST  RFT,  LABELS  'RFT.lab'  $
```

Required:

SAVE.LABELS 'fn'

provides a name in quotes for an external file in which value and variable labels are to be written.

Optional Identifiers:

CHECK

requests that the label file be checked for formatting errors after it is written. CHECK is assumed unless NO CHECK is used.

PSTAT.FILE fn

specifies the P-STAT system file with which the labels are to be used. The SAVE.LABELS report notes the number of variables for which value labels and extended variable labels are provided.

CHECK.LABELS

```
CHECK.LABELS  'LabFile',  PSTAT.FILE  NJdata  $
```

The CHECK.LABELS command checks an existing label file for formatting errors. When PSTAT.FILE is used, the number of variables found in both the P-STAT system file and the labels file are included in the CHECK.LABELS report.

Required:

CHECK.LABELS 'fn'

supplies the quoted name of the external file in which value and variable labels have been written.

Optional Identifiers:

PSTAT.FILE fn

specifies the P-STAT system file with which the labels are to be used. The number of variables for which value labels and extended variable labels are provided is noted.

TITLES

```
TITLE  'Division 12 Report'  $

TITLE  T1  LEFT    'File:  .FILE.'
           CENTER  'Summary Statistics'
           RIGHT   '.DATE.',
       T3  FILL    '-',     SHOW  $


TOP:
File:                    Summary Statistics              Fri Jun 24  1988

------------------------------------------------------------------------
```

The TITLES (or TITLE) *command* defines from one to nine top titles and from one to three bottom titles. Titles may be centered, left or right justified. Titles are used in output produced by commands such as LIST, SURVEY and PLOT, when the *identifier* TITLES is included in those commands. SHOW may be included as the last identifier in the TITLES command to show any defined titles. System variables for the current file name, page number and current date may be used in titles.

Required:

TITLES 'cs'

provides a character string in quotes that contains the text of the title. At least one character string is required. Additional optional identifiers position the string. When none are used, the string is centered and used as the first title.

Optional Identifiers:

B1 to B3

specifies that a bottom title is being defined:

```
TITLE  B1  BLANK  B2  'Figure 2A'  $
```

It is followed by a number between 1 and 3 that indicates which bottom title is being defined.

BLANK

must follow a T or B identifier that points to a specific title line by number. That title line or section of a line is set to blanks. (BLANKS is a synonym.)

CENTER

indicates that the text that follows is to be centered in the title. (C is an abbreviation for CENTER.)

FILL 'c'

specifies a single character to be used to fill the space in the line between defined titles. The FILL specification refers just to the line of titling preceding it. Initially, the fill character is a blank. When no strings are defined for a specific title line (as for T3 in the example at the beginning of the TITLES summary) and FILL is used, the entire title line is filled with the specified character.

LEFT

indicates that the text which follows is to be left justified in the left portion of the title line. (L is an abbreviation for LEFT.)

OFF

turns titles off. OFF may follow a specific title reference such as T4 or B2, indicating that titles from that number on are to be turned off, or it may follow TITLES directly, indicating that *all* titles are to be turned off.

ON

turns on titles which have been turned off. It may be used to apply to a specific title reference or to all titles.

RESET

cancels all defined titles. It may optionally have an argument and cancel the specified title:

```
TITLES  T3  RESET  $
```

RIGHT

indicates that the text which follows is to be right justified in the right portion of the title. (R is an abbreviation for RIGHT.)

SHOW

requests that any defined titles be shown on the terminal or current output device. The titles appear as they will when used in the output of a P-STAT command.

T1 to T9

specifies that top titles are being defined:

```
TITLE  T2  LEFT  'Dated  .DATE.'  $
```

T is followed by a number between 1 and 9 which indicates which top title is being defined.

General Identifiers For Variable Label Formatting

FULL

Use the full (up to 64 character) variable name

TEXT

Use the TEXT if there, else get full variable label.

SHORT.ONLY

This is supported only in the SHOW command. If the variable has no tag it is left blank.

TEXT.ONLY

This is supported only in the SHOW command. If the variable has no text part, it is left blank.

SHORT.TAGS (SHORT)

Use just the tag or, if there is no tag, use the first 16 characters.

SHORT.16

Ignore tags. Take the first 16 characters, trim colons and check for duplicates.

SHORT.OLD

Convert :: to one dot. Convert _ to dot. Then take the initial 16 characters and check for duplicates. This makes labels that are compatible with P-STAT version 2.

6
LIST:
Creating Listings and Reports

The LIST command lists data concisely and attractively, automatically formatting the listing for the current output device — the terminal, a disk file or a printer. Conciseness is important when the output is directed to the terminal because typically the number of columns and lines available for listings is limited. Attractiveness contributes to the readability and clarity of a listing. Thus, the two prime goals of LIST are:

1. a meaningful and attractive layout, and
2. a maximum information in the least amount of space.

These goals are implemented as completely as possible. In situations where they are incompatible, economic use of space takes priority. However, there are a number of LIST command options that may be used to change or improve the appearance of the output when that is more important than conserving space.

LIST requires only the command itself and the name of the file to be listed:

```
LIST  File1  $
```

If the most recently referenced file is the one to be listed, all that is necessary is the command name:

```
LIST  $
```

This may be abbreviated to just “L”. (LIST is the only command that may be shortened to its initial letter.) The file is located and the listing is produced.

6.1 PAGE LAYOUT

The physical placement of the variables and the variable labels on the page affect both the conciseness and clarity of a report. LIST has layout defaults that depend on whether the data for a variable is numeric or character, the relative size of the data values, and the space available on the output destination for the listing. These defaults may be either overridden or controlled by a user to meet his requirements and aesthetics for the listing. The layout of a listing is changed by:

- increasing the spacing between variables,
- expanding, breaking or centering variable names,
- increasing the white space with margins and blank lines,
- supplying titles,
- defining the number of cases and lines per page, and
- transposing (“flipping”) the cases and variables.

6.2 Automatic Spacing of Variables in a Listing

The LIST program makes an initial pass through the entire P-STAT system file that is to be listed in order to determine how much space each variable requires. For numeric variables, the space required depends on the largest absolute value of each variable (9999 requires more space to print than 9), the number of decimal places, and the presence or absence of a sign and a decimal point. (All numbers in a P-STAT file are carried internally as real numbers even if they had no actual decimal places when they were entered.) For character variables, the space depends on the number of characters in each value.

Figure 6.1 shows the LIST command and the listing that it produces of a file of numeric variables. All of the values for both the first and fourth variables are integers, even though they may have originally been entered as fractional numbers (such as “95.00”). It is both a better use of space and more attractive to print numbers that appear to be integers in integer format. The second and third variables in Figure 6.1 both have fractional parts. This fractional information must be preserved as accurately as possible. Notice that both the names and the values of numeric variables are right justified.

Figure 6.1 Integers and Fractional Numbers Affect Spacing

```
LIST  File1  $


VAR1       VAR2        VAR3   VAR4

  95   1030.200    0.054330      1
  65    100.500    0.143143      2
  24      0.050    0.824270      3
  77     -1.000    0.322823      4
  85     10.257   -0.520000      5
  62    500.724    0.760374      6
```

After the LIST program has made its initial pass through the data and determined how many spaces are needed for each variable, two spaces are added for each gap that separates variables. Next, the size of the variable names must be considered. The minimum width that LIST uses for printing a variable is four spaces. Since P-STAT variable names may be up to 64 characters long, they may be formatted in a grid that has up to 6 rows and a dozen spaces. When a name does not fit nicely in a given area, that is, by breaking between lines at a period or underscore character, LIST automatically gives it a wider grid if possible.

As each variable is processed, LIST determines how many variables it has left to fit on the page and how much space is available. If there is space and the variable name being processed will look better with that space, the extra space is allocated. The goal is to try to make the listing look as attractive as possible within the space available. The extra space is not used unless it improves the appearance of the name.

6.3 Controlling Spacing of Variable Names and Values

The GAP, EXPAND, CENTER, PRINT.POSITIONS and DATA.ONLY LIST identifiers and the general identifiers such as TAGS and TEXT control the way the LIST command spaces variables. They change the space between variables, control the expansion and position of variable names, and conserve space by printing only the variable positions and not names. The P-STAT Programming Language (PPL) instruction RENAME changes variable names to shorter, better or “breakable” names.

GAP, followed by a number, specifies the number of spaces to be placed between the data values:

```
LIST  File1,  GAP  4  $
```

The assumed GAP is two. A larger value for GAP both increases the space between values and gives the variable names a wider grid. In other words, the layout of the variable name may use some of the extra gap space.This may result in a more readable listing.

GAP 1 can be used to squeeze more variables on a page. The listing looks readable only when the data are all small integers and the variable labels are very short. Figure 6.2 shows examples of a list with GAP at two different settings. Since there are only four variables and plenty of space on each line, the variable name

"Dependents", which has no natural break, is spread out across the line. Notice also that the names and values of character variables, such as Sex and Status, are *left* justified, whereas those of numeric variables are *right* justified.

Figure 6.2 Changing the Gap Between Variables

```
LIST  Staff,  GAP 1  $


                                  Hourly
 ID  Age Sex      Status          Rate Dependents

101   52 male     full           11.25          3
107   21 male     part-time       4.50          1
113   25 female   full            8.75          0
103   32 female   part-time       5.50          -


LIST Staff,  GAP 3  $


                                        Hourly
ID    Age   Sex        Status            Rate  Dependents

101     52   male       full             11.25           3
107     21   male       part-time         4.50           1
113     25   female     full              8.75           0
103     32   female     part-time         5.50           -
```

When a variable name does not break at a period in the first try and there is space left on the page, the name formatter automatically expands the grid allowed by one space and examines the name again. If it still does not have a natural break, the grid is expanded once more. At this point, however, if there are still quite a few more variables to fit across the page and only a few extra spaces, automatic expansion of the name grid ceases.

Since the name formatter examines both the name size and the space remaining, it is usually the variable names at the left side of the file that are not printed in the most attractive way. It is possible to force the expansion of the name grid by using the identifier EXPAND.

The argument for EXPAND should be a number greater than 2, since two expansions are done automatically. When EXPAND 3 is used, the LIST program attempts three expansions and, possibly, allows three extra positions for the variable name. However, if the expansion does not provide a natural break such as a period, then it is not done. If the EXPAND threshold is reached and the name still does not break at a period, the original space assigned for that variable is used. EXPAND 0 can be used to *turn off* automatic expansion of the variable names. The name "Dependents" in Figure 6.2 would not have been spread out if EXPAND 0 had been used.

Variables may be *renamed* so their names incorporate a period to use as a breaking point, or *variable labels* that incorporate blanks or break points may be supplied as replacements for the variable names. Figure 6.3 illustrates using the PPL instruction RENAME to provide a new name for the variable "Dependents":

```
LIST  Staff  [ RENAME   Dependents  TO  Depen.dents ],
```

The new name contains a period where a break is desired. A period in a variable's name may be dropped in the printout when the next part of the name is on the next line, and including the period would cause expansion.

This name change is in effect only in the listing — the original name remains in the input file. A command that produces an output file must be used to create a permanent change. Variable labels, another way to change variable names, are discussed in the subsequent section entitled “DATA FORMAT”.

Figure 6.3 LIST with Options for More White Space and Titles

```
TITLE  '.DATE.  Listing of .FILE.'  $

LIST  Staff  [ RENAME   Dependents  TO  Depen.dents ],
  MARGIN 6,  SKIP 2,  TITLES  $
```

```
            Wed Jun 15  1997  Listing of Staff

                                          Hourly  Depen
  ID    Age  Sex       Status              Rate  dents

 101     52  male      full               11.25      3
 107     21  male      part-time           4.50      1

 113     25  female    full                8.75      0
 103     32  female    part-time           5.50      -
```

The identifier CENTER may be used in the LIST command to center variable names (or, when supplied, extended variable labels) above their columns. Normally, variable names are right justified above numeric data and left justified above character data. CENTER does not affect the placement of the *values*. To center character values, use the CENTER function in PPL:

```
LIST MailList [ DO #j USING V(1) .ON. ;
     IF V(#j) .CHARACTER. SET V(#j) = CENTER ( V(#j) ) ;
     ENDDO ], CENTER $
```

This command lists the file MailList, centers all character values (the PPL CENTER function) and centers all variable names above their columns (the LIST CENTER identifier). Numeric values are generally not centered. However, they may be centered by first converting them to characters using the CHARACTER function and then centering them using the CENTER function.

The identifier PRINT.POSITIONS requests that the *positions*, and not the names, of variables be printed. It may be used if it is necessary to conserve space in a listing. This may be the case if the file is very large and the number of passes through the file should be minimized. (See the discussion of MAX.PASSES later in this chapter.) The DATA.ONLY identifier specifies that *no* variable names print — just the data is listed. This is useful when the output listing is to be used as input to another software package.

6.4 Controlling the Overall Layout of Listings

Various identifiers control the layout of a listing on a page. MARGIN shifts the listing to the right, SKIP outputs a blank line between cases or groups of cases, and TITLES uses predefined strings to replace default headings. LINES and CASES.PER.PAGE affect the number of cases printed on each page of a listing.

Figure 6.3 illustrates these optional identifiers. MARGIN controls the amount of space in the left margin of the listing. Normally, listings begin printing in the second column (the first column is reserved for carriage control characters.) The argument for MARGIN is the number of spaces the listing is to be indented from the left.

SKIP, short for SKIP.COUNTER, controls the outputting of blank lines. Its argument is the number of cases to print before printing a blank line. In Figure 6.3, a blank line is printed after every two cases. An extra line often improves the readability of a listing, particularly if the spacing correlates with the data in some meaningful way — the extra line separates the cases by sex in this small demonstration file. SKIP 1 produces a blank line after *every* line of output and may be used to double space short listings.

The TITLES *identifier* is used in the LIST command to "turn on" titles defined earlier in the TITLES *command*. Two P-STAT system variables are used in the TITLES command definition. The system variable ".DATE." is the current date in character form. The exact string that prints depends on the particular computer and operating system. ".FILE." is the name of the current input file. (See the chapter on TITLES and LABELS for complete information on providing both top and bottom titles.)

The LIST command normally prints all variables and all cases in the input file, using as many pages as necessary. The number of cases that print on each page depends on the output destination — where the listing is printing, and on the number of lines used by top and bottom titles. If the listing is printing on the terminal without titles, approximately 16 to 20 cases may print; if it is printing in a disk file or on a printer, about 50 to 55 cases may print.

LINES is a *general* identifier — one that may be used in any command — to specify the total number of lines per page. CASES.PER.PAGE is a LIST identifier that specifies the maximum number of cases to print on a page. The number of cases on a page cannot exceed the LINES setting. The default LINES settings are 22 lines for terminals and 59 lines for disk files and printers.

6.5 Transposing a File

A file that does not list concisely on the terminal or elsewhere because it is too wide (has too many variables or very "big" values) may list more attractively after it has been *transposed*. Any P-STAT system file may be transposed — that is, turned on its side so that the cases become variables and the variables become cases.

The TRANSPOSE command is used for files containing only numeric variables and the C.TRANSPOSE command for files containing character and/or numeric variables. C.TRANSPOSE is used to make a more attractive (possibly narrower) listing or to see the effects of some modifications on several complete cases, whereas TRANSPOSE is used primarily to change the arrangement of numeric data.

The C.TRANSPOSE command produces an output file containing *all character* variables. The first variable in the new file contains the names of all the original variables. It is followed by variables corresponding to all the former cases. Figure 6.4 shows a file of character data that does not list compactly and the same file after it is transposed using the C.TRANSPOSE command. The OUT identifier is required. COMMAS may be used to request commas in any of the variables that are numeric in the input file.

```
C.TRANSPOSE Numbers, OUT NumChar, COMMAS $
```

After using C.TRANSPOSE, all variables in the new file are character type — they cannot be used in computations. Therefore, the TRANSPOSE command should be used instead of C.TRANSPOSE when the goal is the rearrangement of data for input to a statistical command. The PPL instructions SPLIT and COLLECT also rearrange data and create numerical indices.

Figure 6.4 Transposing a Wide File

```
      LIST  Synonyms  $

Word                 Syn1              Syn2          Syn3

intake               assimilation      accept        bibulous
misinterpretation    misconception     miscalculate  ambiguous
proof                demonstration     authenticate  attestation
reasoning            ratiocination     consecution   syllogize

Syn4             Syn5

capillarity      adhibit
misconstrue      unclearness
corroboratory    probatory
dianoetic        rationale

      C.TRANSPOSE  Synonyms,  OUT SynTran $

      C.TRANSPOSE completed.
      4 cases processed.

       LIST  $

       CASE          CASE                 CASE             CASE
var    .1            .2                   .3               .4

Word   intake        misinterpretation    proof            reasoning
Syn1   assimilation  misconception        demonstration    ratiocination
Syn2   accept        miscalculate         authenticate     consecution
Syn3   bibulous      ambiguous            attestation      syllogize
Syn4   capillarity   misconstrue          corroboratory    dianoetic
Syn5   adhibit       unclearness          probatory        rationale
```

6.6 DATA FORMAT

The format of the data *values* in a list or report also contributes greatly to the appearance and readability of the listing. The format of the values is changed by:

- the use of value and variable labels,
- the representation for missing data values,
- the number of decimal places and zeros in numeric data,
- the inclusion of commas or dollar signs in large numbers, and
- the folding of character values and value labels.

6.7 Value Labels

Character *labels* may be created and used in place of *numeric or character* data values in listings. Typically, labels are used for categorical data — they identify groups better. For example, the labels “male” and “female” convey information better than do the values “1” and “2” or “m” and “f” for the variable “Sex”. Labels are created using the SAVE.LABELS command or a text editor. Their use in listings is requested by using the LABELS identifier in the LIST command.

Value labels replace the *values* of the variables in listings. (These labels should not be confused with the names of the variables themselves.) When value labels are supplied for a variable, each value of that variable is automatically replaced by the corresponding label (in the listing, not in the P-STAT system file). The use of value labels in a listing often means that additional space is required for each variable, but the listing is generally more attractive and readable.

Value labels are useful for categorical data, that is, data that has one of several distinct values. For instance, values for months of the year might be the integers 1 through 12 and value labels could be used to provide the names of the months:

```
Month  (1) January (2) February (3) March  ...  /
```

Labels may be provided for *any non-missing value* and for each of the three types of *missing values*. They are generally not used for continuous data, such as temperature or weight even though they can be used for fractional values.

The number of labels that can be used in a given LIST command is very large. It depends on the size of P-STAT in use and the size of the labels. In the Whopper/2 version, which is the smallest size available on most computers, there is room for more than 30,000 20 character labels. A given value label may be up to 80 characters long.

Figure 6.5 Listing Numeric Data with Value Labels

Labels file Class5.lab

```
Sex  (1) male (2) female /
Coded.Age  (1) under 25 (2) 25 - 55 /
```

The LIST command

```
LIST  Class5,  LABELS  'Class5.lab',  GAP 4  $
```

Age	Sex	Region	Test Average	Coded Age
13	male	2	1.5	under 25
11	female	3	2.5	under 25
9	male	2	3.9	under 25
25	female	3	2.2	25 - 55
43	male	3	1.9	25 - 55
27	female	2	2.0	25 - 55
54	female	1	3.0	25 - 55
10	female	2	2.4	under 25
21	male	1	3.3	under 25
18	male	1	3.3	under 25

```
LIST command illustrating the LEFT identifier

LIST  Class5,  LABELS  'Class5.lab',  GAP 4,  LEFT  $

                              Test       Coded
Age      Sex        Region    Average    Age

  13     male            2        1.5    under 25
  11     female          3        2.5    under 25
   9     male            2        3.9    under 25
  25     female          3        2.2    25 - 55
  43     male            3        1.9    25 - 55
  27     female          2        2.0    25 - 55
  54     female          1        3.0    25 - 55
  10     female          2        2.4    under 25
  21     male            1        3.3    under 25
  18     male            1        3.3    under 25
```

6.8 Using Value Labels in Listings

After value labels are written in an external file, they may be used in listings. The identifier LABELS is included in the LIST command, followed by the name of the external file containing the labels:

```
LIST  Class5,  LABELS  'Class5.lab'  $
```

Enclose the label file name in quotes. All labels in the label file that refer to variables in the P-STAT system file being listed are used. Excess labels are ignored.

When value labels are used for numeric variables, they are printed using the same rules that are used for printing numbers — they are right-justified in the output field. This is done because value labels are often used for only some of the values for a variable. For example, the variable Children might be coded with 0 through 8 representing exact numbers and 9 representing nine or more. A value label might be desired for only the value 9 to indicate that it means “nine and over”.

If labels exist for most values of variables, the identifier LEFT may be used to left justify the labels in the output field. Any values for which labels have not been supplied are not left justified. The variable names of the variables with value labels are also left justified in their columns. The short list in Figure 6.5 is done both with and without the LEFT identifier.

Value labels in list can be up to 80 characters long. Usually the entire label is used. However, this can be controlled by using MAX.VL and supplying the maximum length to be used. For example:

```
LIST  Employee,  LABELS 'Occup.lab',  MAX.VL 16 $
```

causes the listing to use only the first 16 characters of the value labels.

6.9 USE.XL: Extended Variable Labels

In addition to using labels to replace data *values*, labels may be used to replace *variable names*. Variable labels are called *extended labels* (XL) because they are typically longer than the variable names they replace, although this need not be the case.

Extended variable labels used in the LIST command may be up to 64 characters long. Like other labels in P-STAT, extended variable labels are supplied in an external text file. They may be written directly in that file using an operating system editor or word processor, or by using the P-STAT command SAVE.LABELS.

The identifier USE.XL in the LIST command indicates that any extended variable labels should be used in the listing. The LABELS identifier must be used with USE.XL to supply the name of the label file. When LABELS is used without USE.XL, just the value labels are used in the listing.

Figure 6.6 illustrates both supplying and using extended variable labels. Notice that the extended labels are enclosed in single (or double) *quotes or paired angle brackets*. They come directly after the variable name:

```
Sex  'Sex of Respondent'  (1) Male  (2) Female /
```

and before the value labels.

Figure 6.6 Replacing Variable Names with Extended Labels

```
SAVE.LABELS  'Class5x.lab' ;
 Age       'Uncoded Age'   /
 Sex       'Sex of Respondent'    (1) Male  (2) Female /
 Region    'Geographic Region'    (1) East  (2) West (3) Other /
Coded.Age  'Different Age Groups' (1) Under 25  (2) 25 to 55 /
$

LIST  Class5,  LABELS  'Class5x.lab',  USE.XL  $
```

Uncoded Age	Sex of Respondent	Geographic Region	Test Average	Different Age Groups
13	Male	West	1.5	Under 25
11	Female	Other	2.5	Under 25
9	Male	West	3.9	Under 25
25	Female	Other	2.2	25 To 55
43	Male	Other	1.9	25 To 55
27	Female	West	2.0	25 To 55
54	Female	East	3.0	25 To 55
10	Female	West	2.4	Under 25
21	Male	East	3.3	Under 25
18	Male	East	3.3	Under 25

Extended variable labels may be supplied without also supplying value labels, as is the case with the variable Age in Figure 6.6. USE.XL in the LIST command requests that extended variable labels be used in the listing. Any value labels are also used. When the LABELS identifier is used without USE.XL, only the value labels are used.

6.10 Representing Missing Data

Missing data are normally represented in a listing by "-", "--" or "---" (one, two, or three dashes). The identifier NO DASHES turns off the automatic use of dashes to represent missing data. Missing data then print as "M1", "M2" or "M3".

Value labels may be supplied for each (some or all) of the three types of missing values if other text is desired. Supply the labels the same way value labels are supplied:

```
Status    'Employ*ment Status'
(1) Full  (2)  Part-time  (M1) Unknown  (M2) Refused to Tell /
```

Any values of missing type 1 for Status print as the character string “Unknown”. Any values of missing type 2 print as “Refused to Tell”. Values of missing type 3 print as “---” because no label was supplied for them.

6.11 Specifying Decimal Places for Numeric Data

The number of decimal places that print in fractional numbers and the presence or absence of right-most zeros and commas may be controlled. The PLACES identifiers specify minimum and maximum numbers of decimal places to print, the DOUBLE identifier requests double precision for numbers, TRIM.ZEROS removes zeros from the fractional portion of numbers, and COMMAS requests that large numbers contain commas for readability.

The LIST identifiers MAX.PLACES and MIN.PLACES specify the maximum and minimum number of decimal places to print for numeric values. The identifier PLACES is the same as MAX.PLACES. All three of these options require a single integer between 0 and 9 as an argument. If the variable has no fractional part, it will not print with places even if MAX.PLACES or MIN.PLACES is specified.

The various PLACES options are particularly useful in printing files such as correlation matrices, which contain numbers with long fractional portions. When PLACES 2 is used, for example, the fractional portions of numbers are rounded to two digits. The number 7.5349 prints as 7.53 , and 7.5359 as 7.54.

The PPL *function* PLACES (as distinct from the LIST *identifier* PLACES) may be used to request a specific number of decimal places for selected variables. The function sets the selected variable to the desired number of places *before* the file is passed to the LIST command. Thus, the function PLACES may operate on one particular variable. The subsequent use of the LIST identifiers MIN.PLACES, MAX.PLACES or PLACES may then affect all of the variables in the listing (including the one already modified by the PLACES function):

```
LIST  TTests
  [ SET  T.Prob  =  PLACES ( T.Prob, 2 ) ],
    MAX.PLACES 3  $
```

The identifier DOUBLE requests that the maximum possible number of places be printed for all numeric variables, thereby making use of the fact that numbers are carried internally in the computer in double precision. The number of digits actually printed depends on the particular computer and the range of the values encompassed by each variable. LIST will normally print a value of 123.456654321 as 123.4567; using DOUBLE we get the full 123.456654321.

The identifier TRIM.ZEROS removes zeros from the right side of the *fractional* portion of numbers. Columns of numbers do not appear right aligned, but numbers with many decimal places are easier to see. The identifier COMMAS requests that the *whole number* portion of all numeric variables be listed with commas every three digits (counting left from the decimal point). COMMAS improves the readability of large numbers.

6.12 Representing Dollar Amounts

Values which are dollar amounts can be listed using the DOLLAR identifier. The DOLLAR identifier is followed by the list of variables which should be printed with a “dollar” format. The dollar format prints the value to two decimal places, inserts commas as appropriate and places the “$” immediately to the left of the first non-zero digit. See Figure 6.13 for an example of the DOLLAR format.

```
LIST   Total,  DOLLAR  Sales  Costs  Profit $
```

6.13 Flagging Values of Special Interest

The FLAG identifier is used to flag values of special interest. The FLAG identifier is followed by pairs of variables. The first variable of the pair is printed normally. The second variable of the pair is abutted directly to the right of the first and has no column label. The second variable is typically a short character value containing a percent sign or an asterisk. The following example appends a percent sign to each non-missing values of the variable named Percentages:

```
LIST   Results  [ GENERATE Pct:c1 = '%';
                  IF  Percentages MISSING, SET Pct = ' ' ],
       FLAG  Percentages   Pct $
```

In the following example HIGH is created as a character variable with a width of two. It is set to blank, one asterisk or two asterisks depending on the value of the variables Blood.Pressure and Temperature. A second character variable LOW is either a blank or an asterisk.

```
LIST  Patients [ GEN  High:C2 = ' ',  GEN  LOW:C1 = ' ' ;
                  IF  Blood.Pressure LT 60, SET LOW = '*' ;
                  IF  Temperature GT 101, SET High = '*';
                  IF  Temperature GT 103, SET High = '**' ],
       FLAG  Temperature High  Blood.Pressure  LOW $
```

The use of Flag causes the variable HIGH to be positioned immediately to the right of Temperature and LOW to be positioned to the right of Blood.Pressure. The column labels for HIGH and LOW are not printed. Figure 6.13 contains an example of flagged output.

6.14 Folding Listings

Files with *character* variables that have long strings for their values often do not list concisely. In Figure 6.7, the file named "Address" takes multiple screens or pages to list. If there were many cases in the file, it would not be possible to see a complete case until the left portion of all the cases had listed first. The FOLD identifier folds or breaks the character variables or value labels in a listing. Long character strings then print on several lines. This often makes it possible to list a wide file on the terminal with all the variables for each case showing contiguously. The identifiers N or SKIP are often used with FOLD to highlight the start of each case with a number or blank line. This listing is not very attractive, but it succeeds in getting all values of a case on the screen at the same time.

Figure 6.7 shows the listing of a file that prints in two portions. Not all the variables for a single case fit on a single line. The folded listing of the same file follows. FOLD 7 specifies a maximum of 7 characters per line for each character variable. GAP 1 requests only one space between columns to further conserve room, and SKIP 1 skips one line between cases to differentiate between them.

The FOLD identifier may have one or two optional arguments. The larger of the arguments gives the maximum length of strings that should print on one line, and the smaller argument gives the minimum length. When just one argument is given, it is the maximum length and the minimum length is assumed to be one-half the maximum. When no arguments are provided, the minimum and maximum are assumed to be 16 and 32.

For example, a value of 48 characters is to be folded when FOLD 20 is used. The minimum chunk is therefore 10, and the maximum is 20. To determine the size of the first chunk, a break character (blank, colon, semicolon) is sought in characters 11 to 20. Suppose characters 14 and 18 are blanks. The rightmost one is used, and the first chunk is characters 1-18. If no break characters are found the chunk is characters 1-20. The same logic is used on the remainder of the value for the next line. Note, if FOLD 20 20 is used, each chunk until the last is 20 characters, i.e., no breaks are sought.

Using the FOLD identifier in LIST causes long character strings to print on multiple lines so that more variables fit in a listing. Alternatively, a P-STAT system file may be transposed before it is listed. (See the earlier section on the C.TRANSPOSE command.) In addition, a large output width may be specified for listings.

The general identifier OW (OUTPUT.WIDTH) can be included in the LIST command to set the number of columns to be used for listings. Its maximum argument may range between 600 and 4000, depending on the size of P-STAT on your computer. Not all printers are able to print lines that wide. OW does not provide that ability for your printer — it merely makes it possible for P-STAT to take advantage of it, if it exists.

Figure 6.7 Folding Character Values in Wide Listings

```
     LIST  Address  $

Last
Name      First Name    Street                              City

Fisher    Gorden        56 E. Main St.                      Portsmouth
Perez     Ronaldo       Maryland Ave.                       Washington
Wella     Raquel        1233 S. University Ave.             Ann Arbor


                  Home          Work
State  Zip        Telephone     Telephone     SS Number

RI     02871      401 683-4566  401 683-3321  123569553
DC     20540      202 287-6338  202 281-5677  134679556
MI     48104      616 445-1245  517 344-6284  274937225


     LIST  Address,  FOLD 7,  GAP 1,  SKIP 1  $

Last    First                                      Home      Work      SS
Name    Name    Street  City    State Zip          Telephone Telephone Number

Fisher  Gordon  56 E.   Portsmo RI    02871        401 683   401 683   1235695
                Main    uth                        -4566     -3321     53
                St.

Perez   Ronaldo Marylan Washing DC    20540        202 287   202 281   1346795
                d Ave.  ton                        -6338     -5677     56

Wella   Raquel  1233    Ann Arb MI    48104        616 445   517 344   2749372
                S. Univ or                         -1245     -6284     25
                ersity
                Ave.
```

6.15 LISTING FILES WITH SUBGROUPS

When the data in files are arranged or sorted into subgroups, there are many options available in the LIST command that both:

- display the structure of the subgroups attractively, and
- provide statistical information summarizing each subgroup.

6.16 Headings and Breaks for Subgroups

A P-STAT system file can be said to have subgroups on a certain variable if all cases with the same value on that variable are together in the file. The cases in a P-STAT system file may be arranged into *subgroups* because the data was collected and entered that way, or they may be sorted to create subgroups. Figure 6.8 illustrates sorting

a file. The file is sorted by the values of the variable Sex. Two subgroups are created. Files may be sorted by from one to fifteen numeric and character variables — the number of subgroups that is created depends on the number of values of each of the BY variables.

Files that contain subgroups may be listed with labels, headings and additional white space emphasizing the subgroups. LIST is used with the BY identifier:

```
LIST  Class5,  BY  Sex  $
```

Its argument is the name of the variable or variables whose values define the groups. In the listing, the groups are labeled with *dashed headings*. When the LABELS identifier is also used to give the name of a labels file, labels are used in the headings. If the files had been ordered both by Sex and by Age groups within Sex, "BY Sex Age" could have been used in the LIST command, and each Sex/Age subgroup would be labeled. Headings print when groups change or a page change occurs.

Figure 6.8 Listing Cases by Subgroups with Numbering

```
SORT  Class5,  BY  Sex,  OUT  Class5  $

        ...SORT COMPLETED...
     10 CASES WERE IN THE FILE FILE.

 LIST  Class5,  BY  Sex,  N,  BY.N,  GAP 3,  LABELS 'Class5.lab'  $

-----------------Sex:  Male-----------------

     GRP                        Test
       N    Age    Region    Average    Coded Age

 1     1     13         2        1.5    Under 25
 2     2      9         2        3.9    Under 25
 3     3     43         3        1.9     25 - 55
 4     4     21         1        3.3    Under 25
 5     5     18         1        3.3    Under 25

-----------------Sex:  Female---------------
    GRP                        Test
       N    Age    Region    Average    Coded Age

 6     1     11         3        2.5    Under 25
 7     2     25         3        2.2     25 - 55
 8     3     27         2        2.0     25 - 55
 9     4     54         1        3.0     25 - 55
10     5     10         2        2.4    Under 25
```

In Figure 6.8, the cases are numbered to show their order in the file and in the subgroup. The BY.N and N identifiers are used to request sequential numbering . BY.N gives the number of the case within its group as defined by the BY option. N gives the case number of each case in the total file.

When a file is sorted by a series of variables, it often makes an attractive report to have the major breaks defined by the headings provided by the BY identifier, and the minor breaks defined by extra blank lines. The identifier SKIP.VAR can be used to specify one or more variables to control spacing, thereby providing minor breaks. Whenever the value of a SKIP.VAR variable changes, a blank line is printed:

```
LIST  Patients,  BY  Treatment.Group,
  SKIP.VAR  Patient.Number  $
```

A change in the BY variable, Treatment.Group is a major break. A change in the SKIP.VAR variable, Patient.Number, is a minor break producing a blank line whenever that variable changes. This is a reasonable way to list the file if the file is in Patient.Number within Treatment.Group sort order.

Figure 6.9 LIST with SKIP.OMIT

```
LIST  Cereals  $

                                                Percent
 Category  Manufacturer  Cereal                  Sugar

        1            10  Munchola                    0
        1            12  Some Bran                   0
        1            14  Grape Berries               0
        1            18  Wheat Nuggies               0
        2            20  Raising Bran               10
        2            23  Corn Plakes                15
        2            25  Honeynut Crunch            25
        2            28  Grain Krispies             27


LIST  Cereals,  SKIP.OMIT  Category  $

                                Percent
 Manufacturer  Cereal            Sugar

           10  Munchola                0
           12  Some Bran               0
           14  Grape Berries           0
           18  Wheat Nuggies           0

           20  Raising Bran           10
           23  Corn Plakes            15
           25  Honeynut Crunch        25
           28  Grain Krispies         27
```

SKIP.VAR skips a line when the *value* of a particular variable changes. It may be followed by up to five variable names. The blank line or break produced by SKIP.VAR makes it easier to see subgroups. If there are no subgroups in the file, an extra line appears every time two consecutive values of the SKIP.VAR variable differ — that is, randomly. SKIP (SKIP.COUNTER), discussed earlier, differs because it skips a line after a specified *number of cases* have printed.

SKIP.OMIT functions just like SKIP.VAR, but the variable controlling the skipping is *omitted* from the output. Thus, even though the variable does not appear in the output, breaks in the output make it easy to see that the values of that variable have changed.

This identifier is useful when you have variables like those in the Cereals file in Figure 6.9. Here, the values for Category range from 1 to 2 and it is unnecessary to print them in the output. By using SKIP.OMIT, Category is eliminated from the output and a blank line is printed when its values change. As a result, a listing is produced in which a blank line appears after the first four cereals print.

Paging may also be controlled by the values of a variable. NEWPAGE is followed by a variable whose values control page changes. When the value of this variable changes, a new page of printout begins. In Figure 6.12, a page change occurs whenever the value of Sex changes. (The page change is indicated by the heading containing the file name, although for illustrative purposes the two pages are contained in one figure.)

Figure 6.10 STUB to Blank Out Repetitive Values

```
TITLE  'Means, Standard Deviations and Counts',  B1  'Page .PAGE.'  $

 LIST   SurvStat,  PLACES 2,  SKIP 3,
      STUB  Sex  Age  statistic,
      TITLES,    PAGE.NUMBER 5  $
```

Means, Standard Deviations and Counts

Sex	Age	statistic	Education	Occupation	Marital Status	Hrs Last Week
female	25 to 34	means	2.35	2.22	1.23	34.73
		st.dev	0.54	0.68	0.42	11.09
		counts	66.00	63.00	66.00	26.00
	35 to 44	means	2.16	2.18	1.10	35.82
		st.dev	0.49	0.70	0.31	11.25
		counts	58.00	49.00	58.00	17.00
male	25 to 34	means	2.53	2.16	1.24	44.42
		st.dev	0.56	0.60	0.43	15.97
		counts	38.00	37.00	38.00	31.00
	35 to 44	means	2.24	2.18	1.18	45.41
		st.dev	0.70	0.72	0.39	11.55
		counts	34.00	34.00	34.00	32.00

Page 5

6.17 Stubbing Hierarchical Variables

Variables with a *hierarchical* relationship, that is, variables sorted into ordered groups, lend themselves to "stubbing." Lengthy listings also benefit from stubbing. Stubbing uses the STUB identifier, which:

1. positions the variables given as its arguments on the left side of a listing,
2. repeats the variables on subsequent pages if the listing is a multi-page one,
3. blanks out repetitive values of the stub variables.

The LIST program prints as many variables as it can for all the cases in the file and then prints the next group of variables for all the cases in the file. It continues in this fashion until all variables have been printed. It is often useful, when printing continues on a second page, to repeat one or more variables (such as ID) so that it is easy to identify the cases that are being printed.

With wide files, the STUB identifier is used primarily to request that certain variables print on the left of the page and that the variables repeat on each page of the printout:

```
LIST  Patients,  SKIP.VAR  Patient.Id,  STUB  Patient.Id  $
```

In this example, Patient.Id is used both to control spacing and as a stub variable to be repeated on each page of printout.

Figure 6.10 illustrates using STUB primarily to blank out repetitive values. There is only a single page of printout so the STUB option is not necessary to repeat the variables on each page; column selection could have been used to place them on the left. However, when STUB is used, the stubbed variable value is printed only when its value changes or when a page change occurs. This makes the pattern of repetitions more obvious and highlights the organization of the listing. Note that STUB variables are automatically placed on the left side of the page in the *order* specified in the command.

Figure 6.10 also illustrates the use of the general identifier PAGE.NUMBER. PAGE.NUMBER sets .PAGE. so that page numbering in the command will start with that number. If .PAGE. is used in any title definitions and the TITLES identifier is used in LIST, the page number set using PAGE.NUMBER prints in the title. Normally .PAGE. is reset to 1 at the start of each *command*. RUN.PAGE.NUMBER may be used to reset .RPAGE., which is normally reset to 1 at the start of a P-STAT *run* or session.

Figure 6.11 STUB with FILL of Repetitive Values

```
LIST  SurvStat,
      STUB  Sex Age Statistic,
      FILL '.',
      SKIP.VAR  Age,
      PLACES 2  $
```

Sex	Age	statistic	Education	Occupation	Marital Status	Hrs Last Week
female	25 to 34	means	2.35	2.22	1.23	34.73
.	.	st.dev	0.54	0.68	0.42	11.09
.	.	counts	66.00	63.00	66.00	26.00
.	35 to 44	means	2.16	2.18	1.10	35.82
.	.	st.dev	0.49	0.70	0.31	11.25
.	.	counts	58.00	49.00	58.00	17.00
male	25 to 34	means	2.53	2.16	1.24	44.42
.	.	st.dev	0.56	0.60	0.43	15.97
.	.	counts	38.00	37.00	38.00	31.00
.	35 to 44	means	2.24	2.18	1.18	45.41
.	.	st.dev	0.70	0.72	0.39	11.55
.	.	counts	34.00	34.00	34.00	32.00

If you would like repetitive values of the STUB variables to print, the identifier FILL may be used:

```
LIST  SurvStat,  STUB  Sex Age Statistic,  FILL, ....
```

A value such as “female” will not be blanked out, but will print in each case. If FILL is used with a character argument, that character is used to fill in or replace the repetitive values. This may sometimes be preferable to the blanking out of the values. Figure 6.11 illustrates the use of FILL with the decimal point.

When stub variables are long character variables or numeric variables with long value labels, they are automatically folded when the FOLD identifier is used. The use of NO FOLD.STUB can be used to change this. The following examples illustrate first a folded stub value and then a stub with NO FOLD.STUB in effect.

```
LIST  Offices,  STUB City,  FOLD 8 $                    produces

New York   Smith
New Orle   Jones
ans

LIST  Offices,  STUB City,  FOLD 8,  NO FOLD.STUB $    produces

New York      Smith
New Orleans   Jones
```

6.18 Using STUB to Override BY Headings

The dashed headings that are produced when BY is used are less satisfactory for *small* subgroups. The BY variable headings are normally formatted as a major break — they span the page. When each subgroup has a large number of members, this is attractive. However, when each subgroup has only a few members, the printout may look cut up. In addition, when the subgroups have only a few members, it can take many lines to present a small amount of information.

The STUB identifier may be used *without arguments*, when BY is also used, to format the listing more suitably for small subgroups.

```
LIST  Class5, BY Sex,  STUB $
```

Subgroup statistics, such as TOTALS and MEANS, are calculated as usual. However, the BY variable headings print using the format for STUB variables — that is, the BY variables are stubbed and there is no dashed heading.

The BY and STUB options may be modified by the identifiers and arguments IDENTIFY ALL and NO IDENTIFY. They affect the headings that are printed. When a BY variable changes (or a page change occurs), the BY heading is printed for the variable that *changed.* If two BY variables change simultaneously, both BY headings print. Normally BY variables that have not changed do not have a line in the heading. IDENTIFY ALL causes *all* BY headings to print when any BY variable changes. NO IDENTIFY *turns off* all BY headings. When IDENTIFY is used with the combination of BY and STUB, the variables print in the STUB position and are identified with BY headings also.

When a BY group is so large that it crosses page boundaries, the identifier RE.IDENTIFY can be used. This causes the BY headings to be printed at the top of any new page.

6.19 SUMMARY STATISTICS

The LIST command can produce totals, means, high and low values, and percentages that summarize an entire file or subgroups within the file. When statistics summarizing the whole file are desired, they are requested using the appropriate identifier: MEANS, TOTALS, HIGHS, LOWS or PERCENTAGES. When statistics summarizing subgroups are desired, the variables that define the subgroups are specified with BY and the particular statistics are requested.

Figure 6.12 Statistics on Subgroups — with Headings

```
LIST   SurvSort,  BY  Sex  Age,
STUB   Education  Occupation,

MEANS  Children  Siblings  Hrs.Last.Week,
NEWPAGE  Sex  $


FILE SurvSort

------------------- Sex: female   ---------------------------
------------------- Age: 25 to 34 ---------------------------
                                                        Hrs
                                                       Last
Education     Occupation    Children     Siblings      Week

grade school  professional         0            5        40
                                   5           12         -
high school   white collar         1            2         -
                                   1            2         -
                                   4            7        40
                                   3            5        17
                                   4            2         -
              professional         4            1         -
                                   0            1         -
                                   0           10        40
 Age                           -------      -------   -------
Mean                             2.20         4.70     34.25


------------------- Age: 35 to 44 ---------------------------
                                                        Hrs
                                                       Last
Education     Occupation    Children     Siblings      Week

grade school  professional         3           12         -
high school   white collar         3            2        28
                                   5            4         -
                                   4            3         -
                                   4            4         -
                                   2            2         -
                                   3            6         -
                                   3            1         -
              professional         7            3         -
                                   2            7         -
 Age                           -------      -------   -------
Mean                             3.60         4.40     28.00

 Sex                           -------      -------   -------
Mean                             2.90         4.55     33.00
```

```
 FILE SurvSort

------------------- Sex: male     --------------------------
------------------- Age: 25 to 34 --------------------------
                                                        Hrs
                                                       Last
Education     Occupation    Children     Siblings      Week

high school   blue collar          1            1        40
              white collar         1            1        40
                                   2            2        43
                                   3            5         -
              professional         3            8        70
                                   2           11         -
                                   0           15        15
                                   0            3         -
college       blue collar          0            2        25
              white collar         0            1         -
 Age                            -------      -------  -------
Mean                              1.20         4.90    38.83

------------------- Age: 35 to 44 --------------------------
                                                        Hrs
                                                       Last
Education     Occupation    Children     Siblings      Week

grade school  professional         2            3        36
                                   5            9        40
high school   white collar         3            3        40
                                   3            5        48
                                   3            2         -
                                   1            5        40
                                   5            4        40
              professional         2            1        40
                                   1            0        40
                                   3            3        40
 Age                            -------      -------  -------
Mean                              2.80         3.50    40.44

 Sex                            -------      -------  -------
Mean                              2.00         4.20    39.80

Grand                           =======      =======  =======
 Mean                             2.45         4.37    38.10
```

Typically, BY and several statistical identifiers are used both:

1. to request statistics by subgroups, and
2. to arrange the list with dashed headings.

Summary statistics print at the end of each subgroup or at the end of the file — before the next dashed heading — except for percentages, which print by each case. When STUB is also used, the BY variable is stubbed and there are no dashed headings. When NO IDENTIFY is used with BY and summary statistics, subgroups are not explicitly identified, but the printing of the statistics creates a break.

6.20 Means, Totals, Lows and Highs

Figure 6.12 illustrates producing a listing with means for subgroups. The variables following BY, Sex and Age, define the groups. The file must be *sorted* by these same variables in the same order to create the subgroups before it can be listed by subgroups. The identifier MEANS is followed by the variables for which means are desired. When the identifier MEANS is used *without* arguments, means are produced for *all* the numeric variables that are not BY or STUB variables. Means can be produced on a maximum of 60 variables at a time.

Since there are two sort variables in Figure 6.12, the mean is printed when there is a change in Sex and again when there is a change in Age. When Sex and Age both change, the means for Females in that age group and the age group means are both printed. When the end of the file is reached, a grand mean is also computed and printed. Figure 6.13 shows a listing with statistics, but without dashed headings. This is accomplished by using the STUB identifier without arguments.

Figure 6.13 Statistics on Subgroups — Dollar, Commas and Flag

```
TITLES  'Department Totals for 1989',
    B1  '* Sales income below target value' $


LIST  Sales [ GEN Worry:c = ' ' ;
              IF Sales  LT  1000000,  SET Worry = '*' ],

BY  Department,

COMMAS,   DOLLAR  Sales,   FLAG  Sales  Worry,

STUB,     TOTALS  Volume  Sales, TITLES,
FILL '"'  $
```

Department Totals For 1989

Department	Region	Volume	Sales
Software	North	1,375	$1,538,234.75
"	South	1,430	$1,426,900.50
"	East	1,550	$1,850,930.00
"	West	1,100	$952,677.02*
Department Total		5,455	$5,768,742.27
Hardware	North	250	$5,335,498.50
"	South	190	$4,250,344.00
"	East	350	$8,550,604.00
"	West	180	$2,503,340.00
Department Total		970	$20,639,786.50
Grand Total		6,425	$26,408,528.77

*** Sales income below target for year**

Other summary statistics that can be produced by the LIST command are TOTALS, LOWS and HIGHS. If these identifiers are not followed by variable lists, all the numeric variables not specified as BY or STUB variables are used. However, there is a limit of 60 variables for each statistic.

Any mixture of the four types of summary statistics can be used. If more than one statistic is requested in a LIST, the results are printed in the order in which they appear in the command. In this example, the order is totals, means, lows and then highs:

```
LIST    BY  Household,
        TOTALS  Income,
        MEANS  Age,
        LOWS  Age Income,
        HIGHS   Age  Income  $
```

NO CASES specifies that no cases are to print. If the identifiers HIGHS, LOWS, MEANS or TOTALS are also specified, the listing contains *only* those summary statistics without any cases. In this example, NO CASES causes only the variable names and the totals of all numeric variables to print:

```
LIST  Diet,  TOTALS,  NO CASES  $
```

6.21 PERCENTAGES

The identifier PERCENTAGES produces an extra column with percents of the total count accounted for by each variable. *PERCENTAGES may only be used with the identifier TOTALS*. Percents are based on the totals for the entire file when BY is not used. When BY variables are specified, the percents are based on the totals for the defined subgroups.

CASE.PERCENTAGES produces an extra column with the percents of the *case* total accounted for by each value in the case — that is, the percentages are "row" percents, rather than "column" percents. The TOTALS identifier must be used with two or more variables when CASE.PERCENTAGES is specified.

CASE.PERCENTAGES and PERCENTAGES cannot be used together. Neither PERCENTAGES nor CASE.PERCENTAGES can be used with the FLAG identifier.

6.22 GENERAL CONTROL

Several identifiers control various parameters of the listing procedure itself, rather than the format of the list or report. The identifiers OW, PR and VERBOSITY are general identifiers that may be used by themselves as commands, or as identifiers within many different commands. MAX.PASSES is a LIST identifier that limits the number of passes LIST will make through a file.

The PR *identifier* changes the output print destination for the duration of the LIST command. This command directs the listing to the file on disk referenced by the name "Printer.txt":

```
LIST  Students,  PR  'Printer.txt'  $
```

A name for a disk file, to be printed later, may be given. The output destination is supplied between single or double quotes, and it may contain any symbols that the computer operating system uses to reference disks and directories.

When PR is used by itself as a *command* rather than as an identifier, all future output is directed to the specified destination:

```
PR  'Sue/Diskfile'  $
```

If PR is used as a command *without an argument*, all future output is directed to the primary output device. This is typically the terminal in an interactive session and a disk file in a batch session.

```
LIST Students, PLACES 2, GAP 1, PR 'Students.txt'
PRINT 'Students.txt' $
```

The PRINT command can be used to send a print destination file to the primary printer for your computer. On Unix it may be necessary to specify print instructions. For example:

```
PRINT 'Students.txt', CONTROLS 'lp -dlw' $
```

The OW (OUTPUT.WIDTH) identifier, discussed in the earlier section on FOLD, can be included in the LIST command to set the number of columns to be used for listings to special widths. Its maximum argument ranges from 600 to 4000, depending on the size of P-STAT in use. This increased output width overrides that assumed by print destinations (80 for terminals and 132 for disk files and printers) or defined by a PP (PRINT.PARAMETERS) command. However, if the actual device (disk file or printer cannot support that width, the lines in the listing may wrap around or be truncated.

VERBOSITY controls the extent of information present in error messages and in listing headings. The argument for VERBOSITY, which may be shortened to V, is an integer from 1 to 4. Arguments between 1 and 3 request that the headings for small files not be printed. An argument of 4 requests that headings always be printed.

MAX.PASSES controls the maximum number of passes that are made through the system file in listing it. Normally, as many passes are made as are necessary to list all of the variables. The width of the output device, the number of variables, and the room it physically takes to list a variable's values and labels determine the number of variables that can be listed in a single pass, and thus the number of passes required. One grouping of variables that fills the output width is considered "one pass" through the file. The number of cases in the file may require several pages of listing, but that is not relevant to the number of passes.

If the number of passes is to be limited with MAX.PASSES, the variables should generally be selected (using the PPL instructions KEEP and DROP) in order of their relative importance. After the specified maximum number of passes is made, the listing stops regardless of whether or not all the variables have been listed.

SUMMARY

LIST

```
L
LIST      $
LIST  XX,  GAP 3  $

LIST  XX,  LABELS  'XX.Lab'  $
```

The LIST command lists any P-STAT system file. L by itself (no $) or LIST $ list the most *recently referenced* file. When no optional identifiers are used, the listing is automatically formatted to fit the output device (terminal, diskfile or printer) to which it is directed. (See the C.TRANSPOSE command at the end of the summary for information about rotating a file so that it may be listed in a concise form.)

Required:

LIST fn

specifies the file to be listed. The file name must be provided to select a particular file. LIST $ or just L will list the most recently referenced file.

Optional Identifiers: Page Layout

CASES.PER.PAGE nn

specifies the maximum number of cases to print on a page. The number of cases specified cannot exceed the LINES setting.

CENTER

requests that the variable names or extended variable labels be centered over each column. The values or value labels in each column are not centered. When CENTER is not used, the variable names or extended labels are flush left over character data and flush right over numeric data.

DATA.ONLY

requests that the file be listed without any heading information and without variable names. This is usually used when the data are to be sent to an external file for use by some other program.

EXPAND nn

indicates the number of extra expansions of the print positions that may be used by the formatting routines to make the variable names readable. When EXPAND is not used, two expansions are automatically tried. EXPAND 0 turns off the automatic expansion of variable names.

GAP nn

gives the number of spaces to be placed between the variables (columns). The assumed GAP is 2.

MARGIN nn

gives the number of spaces to indent the left of the listing (the left margin). This can be used to center a printout.

N

causes the printing of the case number within a file.

PRINT.POSITIONS

requests that only the *positions* of the variables in the file be printed, not the variable names.

SKIP.COUNTER nn

indicates the number of lines to print before a blank line is left for spacing. SKIP.COUNTER 1 double spaces a listing. SKIP is an abbreviation for SKIP.COUNTER.

TITLES

requests that all defined top and bottom titles print before and after the listing. Titles are defined using the TITLES command (see the TITLES and LABELS chapter).

Optional Identifiers: Data Format

COMMAS

requests that the whole number portion of numeric variables be listed with commas every three digits (counting left from the decimal point).

DASHES

prints dashes: "-" for missing type 1, "- -" for missing type 2, and "- - -" for missing type 3. This is assumed. NO DASHES specifies that dashes are not to be printed. M1, M2 and M3 are then printed for the three types of missing. Labels may be supplied for missing values if other character strings are desired.

DOLLAR vn vn

prints the list of variables cited with exactly two decimal places. The value is preceded by a dollar sign.

DOUBLE

requests that the maximum number of places be printed for all variables, thereby taking advantage of the fact that numbers are carried internally in the computer in double precision. The number of digits actually printed depends on the particular computer and the range of the values encompassed by each variable.

FLAG vn vn

is followed by pairs of variables. The first variable of a pair is printed normally. The second variable is abutted directly to the right of the first and has no column label.

FOLD nn nn

requests the "folding" or breaking of the character variables in the listing. Folding results in long character strings printing on several lines. The identifier N is often used with FOLD to highlight the start of each case with a number.

The larger of the optional arguments gives the maximum length of strings that should print on one line, and the smaller argument gives the minimum length. When just one argument is given, it is the maximum length and the minimum length is assumed to be one-half the maximum. When no arguments are provided, the maximum is assumed to be 32. Breaks occur at a blank, if possible. For FOLD 32, a blank in characters 17-32 causes a break; if none, 32 characters are used.

LABELS 'fn'

provides the name of an external file containing value labels, and requests that those labels replace numeric and character values in the listing. Label files are written using the SAVE.LABELS command or an operating system editor. The format of labels is the same regardless of how they are written:

```
Sex  'Sex of Respondent'  (1) male  (2) female  /
```

See the USE.XL identifier also.

LEFT

causes any value labels and their variable names to be left justified.

MAX.PLACES nn

indicates the maximum number of decimal places to be used in listing numeric data.

MIN.PLACES nn

indicates the *minimum* number of decimal places to be used in printing numeric data *that have fractional parts*. A number may have more digits in its decimal portion than show in the default listing. If not, additional zeros are added to the decimal portion until the requested minimum number of places (up to nine) is achieved.

MAX.VL nn

sets the maximum number of characters in a value label. MAX.VL 16 would only use the initial 16 characters. The default is to use the entire value label.

PLACES nn

indicates the maximum number of decimal places to be used. It is equivalent to MAX.PLACES.

TRIM.ZEROS

causes trailing zeros to be removed from the right side of the decimal portion of numeric values.

USE.XL

indicates that *extended variable labels* are to be used. These labels replace variable names — they may be up to forty characters long. The LABELS identifier must be used with USE.XL to give the name of the label file that contains the extended labels and, possibly, value labels. See the format of extended labels above, under the LABELS identifier. When USE.XL is *not* used, but LABELS is, only *value* labels are used in listings (not *variable* labels).

Optional Identifiers: With Subgroups

BY vn

specifies one to ten numeric and character variables that define subgroups. The file must already be sorted on these variables. Whenever a group changes, there is a major break in the printing and the new group is identified with a dashed heading. When STUB is also used (without arguments), there are no dashed headings and the BY variables are stubbed to identify the subgroups.

BY.N

requests that a within-group sequence number be printed. This may be used only when BY is also used.

FILL

requests that STUB values be repeated even when they are the same as the previous ones. FILL followed by a character in quotes causes that character to be used in the STUB field instead of the repetitive variable value:

```
LIST  Cats,  STUB  Breed,  FILL '.'  $
```

IDENTIFY

causes the BY heading to print when a BY variable changes. This is assumed. IDENTIFY ALL causes the headings for all BY variables to print any time that any BY variable changes. NO IDENTIFY turns off all BY headings.

NEWPAGE vn

specifies a variable that, when its value changes, causes a page change in the printout. NEWPAGE is often used with BY.

RE.IDENTIFY

causes the current BY labelling to be repeated whenever a new page begins for that BY group.

SKIP.OMIT vn

causes a variable to be omitted from the output and a blank line to be printed when the value of that variable changes. Thus, even though the variable does not appear in the output, its values control the placement of blank lines in the listing.

SKIP.VAR vn vn

specifies one or more variables that, when one of their values changes, indicates a minor break. A blank line is printed to separate groups.

STUB vn

designates the names of variables to appear on the left side of each page of printout. STUB variables are repeated on each page of a multi-page printout. Repetitive STUB values are blanked out unless the identifier FILL is used.

Optional Identifiers: Summary Statistics

CASES

requests that all cases from the input file be printed. This is assumed. NO CASES may be specified with the identifiers HIGHS, LOWS, MEANS or TOTALS to print just the variable names and the corresponding summary statistics.

CASE.PERCENTAGES

produces an extra column with the percents of the *case* total accounted for by each value in the case — that is, the percentages are “row” percents, rather than “column” percents. The TOTALS identifier must be used when CASE.PERCENTAGES is specified.

HIGHS vn vn

specifies a variable or list of variables for which high values are to be computed. If no variables are named, all the numeric variables not cited as BY or STUB variables are used. Statistics summarize the whole file, unless BY is also used — then, statistics summarize the subgroups defined by the BY identifier as well as the entire file.

LOWS vn vn

specifies a variable or list of variables for which low values are to be computed. If no variables are named, all the numeric variables not cited as BY or STUB variables are used.

MEANS vn vn

specifies a variable or list of variables for which mean scores are to be computed. If no variables are named, all the numeric variables not cited as BY or STUB variables are used.

PERCENTAGES

produces an extra column with the percents of the total count accounted for by each value. These are "column" percents. The TOTALS identifier must be used when percentages are requested, and percentages are given only for the variables for which TOTALS are calculated. If BY variables are specified, the percents are based on the totals for the innermost subgroups.

TOTALS vn vn

specifies a variable or list of variables for which totals are to be computed. If no variables are named, all the numeric variables not cited as BY or STUB variables are used.

There may be up to 60 variables for which HIGHS, LOWS, MEANS and/or TOTALS are produced in the Whopper/1 size of P-STAT. This is the size that is commonly distributed.

Optional Identifiers: General Control

MAX.PASSES nn

controls the maximum number of passes that are made through the data file in listing it. Normally, as many passes are made as are necessary to list all of the variables. If the number of passes is limited with MAX.PASSES, the variables should generally be selected (using PPL) in order of their relative importance. After the specified maximum number of passes is made, the listing stops regardless of whether or not all the variables have been listed.

OW nn

specifies the number of columns to be used for listings. It is short for OUTPUT.WIDTH. The maximum number of columns that may be specified is from 600 to 4000, depending on the size of P-STAT in use. When OW is not used, an output width of 80 is assumed for terminals and 132 for disk files and printers. When OW is used, it overrides these assumptions. The print destination (terminal, disk file or printer) should be capable of supporting this increased width.

PAGE.NUMBER nn

sets the page number, .PAGE., to the specified number. If .PAGE. is used in any title definitions and the TITLES identifier is used, the page number set using PAGE.NUMBER is used to number the initial page, and so on. Normally, .PAGE. is reset to 1 at the start of each command.

PR 'fn'

specifies a print destination. The listing is directed to this destination (a disk file or a printer) instead of to the terminal. PR, without an argument, directs listings back to the primary output device (the terminal).

RUN.PAGE.NUMBER nn

sets the run page number, .RPAGE., to the specified number. Normally,

.RPAGE. is reset to 1 at the start of a P-STAT session. See PAGE.NUMBER also.

VERBOSITY nn

gives an integer value between 1 and 4 that designates a verbosity setting. Settings between 1 and 3 cause headings not to be printed when the file is very small. If VERBOSITY 4 is used, headings are always printed. V is an abbreviation of VERBOSITY.

Optional Identifiers: Variable Names

FULL

Use the full 64 character variable labels.

TEXT

Use just the text of the variable labels.. If no text, use the full label

SHORT.TAGS

Use just the tags. If no tags, use the first 16 characters

SHORT.16

Use the first 16 characters, trim colons from any tags and check for duplicates.

SHORT.OLD

Convert to the earlier 16 character label format and check for duplicates.

C.TRANSPOSE

```
C.TRANSPOSE  Waste,  OUT  WasteTR  $
```

The C.TRANSPOSE command "flips" a file on its side. This often changes the aspect ratio of a file from wide to narrow, permitting a more attractive concise listing. All variables in the transposed file are *character* type. The TRANSPOSE command should be used for numerical transpositions.

Required:

C.TRANSPOSE fn

specifies the name of the file to be transposed.

Required Identifiers:

OUT fn

gives a new name for the transposed file.

Optional Identifiers:

COMMAS

places commas within all of the formerly numeric variables.

7
COUNTS: Frequencies and Descriptive Statistics

The COUNTS command produces *frequency distributions* and *percentages* for all unique values of all character and numeric variables in te input file. There may be an unlimited number of different values of each variable. Character variables may be any length. Cumulative counts and percentages are calculated, as are various means and measures of variation. If weighting is requested, both unweighted and weighted frequencies and statistics are computed. Multiple response or related variables may be combined and their values tallied as if they were single variables. Percentages of combined variables can be based on good or total responses or respondents. Value labels may be supplied. A single BY variable can be used to get subgroup results.

Note: COUNTS output in version 3 has one major difference from COUNTS in version 2. There are now 3 choices for the order of the variables when the BY identifier is used and the assumed order is different from the assumed order in version 2. A new identifier, FORMAT, specifies which of the 3 different layouts you prefer. A second, minor change: the identifier RESET replaces NONE (which continues to work) because it better identifies what is happening.

The default sections of statistics produced by COUNTS include:

- VALUES: frequencies and percentages of all unique values.
- MISSINGS: frequencies and percentages of three types of missing values.
- NS: number of different values found in the variable, number of cases with a non-missing value, and overall number of cases. Also, for combined variables, the number of non-missing responses, the number of possible responses, and the number of variables in the combination.
- STATS: mean, median, mode, number of cases at the mode, number of tied modal values, variance, standard deviation, standard error of the mean, low and high 95% confidence intervals of the mean, low, high, range, interquartile range, sum, and sum of squares.

Additional summary statistics may be requested as sections. These statistics include the following:

- HARMONICS: harmonic and geometric means, mean of positive values.
- MOMENTS: skewness, kurtosis, standard errors of each, coefficient of variation, corrected sum of squares, 2nd, 3rd and 4th moments.
- PERCENTILES: 1st, 5th, 10th, 25th, 50th, 75th, 90th, 95th and 99th percentiles.
- NORMALS: Shapiro-Wilk and Anderson-Darling tests of normality.

Each of these sections may be selected or omitted as a section. BASIC, a subsection of STATS, can be used to obtain just the mean, standard deviation, low value and high value. Individual selection of statistics from within a section is also supported. If you wish to omit some of the items in the default sections, the identifier RESET, cleans the slate. Thus “COUNT work, RESET, BASIC” produces an output which has just the basic statistics for all the variables in the file.

The output from the COUNTS command is a P-STAT system file, which is automatically listed with appropriate format options. The output may be directed to a print file or to an on-line printer. The frequency distributions and percentages may be in ascending or descending order on either the values or frequencies. The result output columns usually include counts, cumulative counts, percents and cumulative percents with four ad-

ditional result columns if weighting is requested. However, you may choose which of these 4-8 result columns you wish to see.

7.1 BASIC USAGE

Typical usage of the COUNTS command requires only that an input file be specified — a name for the output file that is produced is optional and need not be supplied:

```
COUNTS  PaceSet  $
```

Figure 7.1 Frequencies and Statistics Produced by COUNTS (Portion)

```
COUNTS  PaceSet  [DROP  ID  Income],  OUT  PaceSetC  $


-----COUNTS of file PaceSet completed-----
80 cases and 12 variables were processed.

                                          cum            cum
variable  value            count or stat  count    pct     pct

Age       1                 28.             28   35.     35.
          2                 27.             55   33.75   68.75
          3                 23.             78   28.75   97.5
                             -               -    -       -
          missing 1          2.             80    2.5    100.
          missing 2          0.              0    0.     100.
          missing 3          0.              0    0.     100.
                             -               -    -       -
          number of values   3.              -    -       -
          good  n           78.              -    -       -
          total n*          80.              -    -       -
                             -               -    -       -
          mean               1.935897436     -    -       -
          median             2.              -    -       -
          mode               1.              -    -       -
          cases  at mode    28.              -    -       -
          number of modes    1.              -    -       -
          variance           0.658175158     -    -       -
          std deviation      0.811279951     -    -       -
          se of mean         0.091859366     -    -       -
          low 95 conf mean   1.752982071     -    -       -
          up  95 conf mean   2.118812801     -    -       -
          low                1.              -    -       -
          high               3.              -    -       -
          range              2.              -    -       -
          q3-q1              2.              -    -       -
          sum              151.              -    -       -
          sum of squares   343.              -    -       -
```

Frequencies are tallied for all unique values of all variables in the input file. Use the P-STAT Programming Language (PPL) instructions KEEP or DROP to select or omit variables such as “ID” or “Case.Number”, which are typically consecutive unique values whose frequencies are not of interest. Figure 7.1 shows the COUNTS

command and a portion of the output it produces — the brief report, and the statistics for one of the variables, “Age.”

The output file produced by COUNTS contains frequencies and percentages for all unique values and all of the three types of missing values. The number of different values, the total number of cases and the number of good (non-missing) values are also given. An asterisk indicates the value that is the base or denominator of the percentages — by default, it is the total number of cases. (This may be changed; see the section on “Changing the Percentage Base.”)

Additional statistics that summarize the *numeric* data values are also included in the output. The mean is the arithmetic mean of all observed values for the particular variable. The median is the middle value, and the mode is the most common value. The number of cases at the mode is given, as is the number of modes. When there is more than one mode, there are several most common values and the sample is multi-modal.

The variance, standard deviation, low, high, range and interquartile range (q3-q1) are measures of the dispersion of values about the mean. The standard error of the mean gives the sample-to-sample fluctuation of the mean. It is used in calculating the 95% confidence intervals about the mean. The sum of the values and the sum of the squared values (sometimes called the “uncorrected” sum of squares) are also given; they may be of use in calculations of other statistics, such as variants of the variance and standard deviation.

7.2 Listing the Output File

The COUNTS output file is a temporary file with a name beginning with the prefix “WORK”, unless the OUT identifier is used to supply a name for a regular autosave (“permanent”) file — see Figure 7.1. (Temporary files disappear when the END command is processed and the P-STAT run ends.) The output file is automatically listed in an appropriate format as the COUNTS command completes. Using NO LIST, however, prevents the automatic listing.

P-STAT generates and executes a LIST command to produce the automatic listing, which is a *stubbed* listing:

```
LIST  OutFile,  STUB  variable value,  SKIP.VAR  variable,
      DOUBLE,   FOLD  26,   TRIM.ZEROS  $
```

“OutFile” is either a name beginning with “WORK” or the name supplied in the COUNTS command after the OUT identifier. The COUNTS output file has variables literally named “variable” and “value”. “variable” has the name of the current variable for which frequencies and other statistics are computed. “value” contains either the values being aggregated or the names of the statistics (like “mean”). They are stubbed so that they re-appear in successive passes of the printout, which occurs when weighting is requested and the output destination allows for only 80 characters (the terminal, for example). In addition, the stub of “variable” prevents many repetitions of the name.

SKIP.VAR skips a line when the variable name changes — that is, between variables. DOUBLE requests full double precision for all numbers. FOLD 26 breaks long character strings so that only 26 characters print per line; additional characters continue on the next line. TRIM.ZEROS removes trailing zeros from the right side of the decimal portion of numbers.

An alternate, slightly narrower format for the listing is produced when the LIST.BY identifier is used in COUNTS. The LIST command that is generated by COUNTS then contains “BY variable, STUB value” instead of “STUB variable value”, and omits the “SKIP.VAR variable” phrase. Each variable is identified by a heading across the page.

The output file produced by COUNTS when OUT is used is a regular P-STAT autosave file. It can be listed at any time, just as any autosave file can, by using the LIST *command*. The identifiers shown in the previous example may be used, or any identifiers normally permitted in LIST may be used. PPL may be included in a LIST command or in any other command to select, order or modify the file.

7.3 Printing the Output File with Titles

The listing of the COUNTS output file prints on the *default print destination*, typically the terminal. The name of a disk file may be supplied as an alternate destination. The PR identifier gives the name of a *disk file* that can be printed at a future time. PR, TITLES, and other general identifiers may be supplied directly as identifiers to the COUNTS command.

```
COUNTS  PaceSet  [ DROP  ID  Income ],   OUT PaceSetC,
    PR '/usr/tom/PaceSet.lst' $
```

The name of the alternate print destination is within quotes. Any string recognized by the host computer may be given after PR. The name may include slashes or colons — whatever is recognized by your system as a legal disk file name.

The TITLES *identifier* is used to turn on any titles previously defined with the TITLES *command*:

```
COUNTS  F243,    OUT  F243C,  LIST.BY,
        TITLES,  PR 'F243.prn' $
PRINT 'F243.prn'$
```

This command produces a listing with headings (LIST.BY) and titles. The PRINT command sends the disk file to the current system printer. On Unix it may be necessary to specify print instructions. For example:

```
PRINT 'F243.prn', CONTROLS 'lp -dlw' $
```

The TITLES command must be used *prior* to COUNTS to specify the title strings. Up to nine top and three bottom titles may be supplied:

```
TITLES  T1  LEFT    'May 13, 1988',    RIGHT  'Research Dept.',
        T3  CENTER  'Frequency Distributions',
        B1  CENTER  '.PAGE.'   $
```

Scratch variables or system variables, such as ".PAGE.", may be used in title strings. The page number is substituted for .PAGE. in the bottom title. (See the chapter on TITLES for more information.) After defining titles, the TITLES identifier may be included in COUNTS to print the titles above the listing of the output file. Figure 7.5 illustrates using TITLES.

Any identifiers that are specific to the LIST command may be supplied as part of an optional *quoted* character string argument to the identifier USE — the argument is used in the generated LIST command. The entire argument following USE is enclosed in quotes. The identifiers are separated by commas and included in the quoted argument. Supplied identifiers are an *addition* to the command that is generated by COUNTS. Identifiers that conflict with those in the generated LIST command should not be used. Instead, use a separate LIST command to list the COUNTS output file, as shown in Figure 7.2 .

```
COUNTS   f243,   USE   "MAX.PLACES 4,   GAP  1"   $
```

7.4 COUNTS Output and Value Labels

The listing of the COUNTS output normally contains the values for numeric data unless a labels file is supplied. The LABELS file is used to supply text which is substituted in the listing for the corresponding data values.

```
COUNTS  PaceSet  [ DROP  ID  Income ],  LABELS 'PaceSet.Lab' $
```

Figure 7.5 illustrates the use of LABELS in the COUNTS command. The COUNTS command uses up to 32 characters of the value label text but does not make use of any extended labels.

7.5 Weighting the Frequencies

The counts, percentages and statistics calculated by the COUNTS command may be weighted. The name of the variable whose values are case weights is supplied with the WEIGHT identifier:

```
COUNTS  PaceSet, Labels  'PaceSet.Lab',   WEIGHT  W.Factor, ....
```

Normally, each case “counts as one.” When a weight variable is supplied, each case “counts” as much as its weight. The weight variable may be generated as the file is input to COUNTS:

```
COUNTS PaceSet   [GEN W.Factor = 1     ;
    IF  Age = 1,  SET  W.Factor =  .929;
    IF  Age = 2,  SET  W.Factor =  .964;
    IF  Age = 3,  SET  W.Factor = 1.130 ],
  OUT PaceSetC,
  WEIGHT  W.Factor, ....
```

Figure 7.2 Weighted COUNTS Output File Displayed with LIST (Portion)

```
            LIST  PaceSetC,  BY  Variable,  GAP 1,  PLACES 3,
q3,
                  TRIM.ZEROS   $


--------------------------- variable: Age -------------------------------

                    count   cum           cum w.count   w.cum         w.cum
value             or stat count    pct    pct or stat   count  w.pct    pct

Under 30           28.       28  35.    35.    26.012  26.012  32.5   32.5
30 to 50           27.       55  33.75  68.75  26.028  52.04   32.52  65.03
Over 50            23.       78  28.75  97.5   25.99   78.03   32.48  97.5
                    -         -   -      -      -       -       -      -
missing 1           2.       80   2.5  100.     2.     80.03    2.5  100.
missing 2           0.       80   0.   100.     0.     80.03    0.   100.
missing 3           0.       80   0.   100.     0.     80.03    0.   100.
                    -         -   -      -      -       -       -      -
number of values    3.        -   -      -      3.      -       -      -
good  n            78.        -   -      -     78.03    -       -      -
total n*           80.        -   -      -     80.03    -       -      -
                    -         -   -      -      -       -       -      -
mean                1.936     -   -      -      2.      -       -      -
median              2.        -   -      -      2.      -       -      -
mode                1.        -   -      -      2.      -       -      -
cases  at mode     28.        -   -      -     26.028   -       -      -
number of modes     1.        -   -      -      1.      -       -      -
variance            0.658     -   -      -      0.675   -       -      -
std deviation       0.811     -   -      -      0.822   -       -      -
se of mean          0.092     -   -      -      0.093   -       -      -
low 95 conf mean    1.753     -   -      -      1.815   -       -      -
up  95 conf mean    2.119     -   -      -      2.185   -       -      -
low                 1.        -   -      -      1.      -       -      -
high                3.        -   -      -      3.      -       -      -
range               2.        -   -      -      2.      -       -      -
q3-q1               2.        -   -      -      2.      -       -      -
sum               151.        -   -      -    156.038   -       -      -
sum of squares    343.        -   -      -    364.034   -       -      -
```

Often, weights are calculated to maintain the same total number of cases:

$$.929\,(28) + .964\,(27) + 1.130\,(23) = 26.012 + 26.028 + 25.990 = 78.03$$

Now each age group contains roughly 26 cases, and yet the total of 78 non-missing cases remains the same. (See the counts in Figure 7.1.) When the WEIGHT identifier is used in the COUNTS command, the output file contains *both* weighted and unweighted counts and percentages. The weight variable is included in the file, but it is not weighted. A negative or missing weight variable is treated like a weight of zero.

In Figure 7.2, a portion of the output file is printed using the LIST *command*. The command includes "BY Variable" to position "variable" as a heading across the top of the listing. "GAP 1", "PLACES 3" and "TRIM.ZEROS" are also used. These identifiers produce a narrow listing with trailing zeros removed from the right of the decimal point. A very similar listing could be produced by including LIST.BY and USE "GAP 1, PLACES 3" in the COUNTS command.

7.6 Respecting Case in Character Values

The COUNTS command produces frequencies, percentages and applicable statistics for character as well as numeric variables. Unique variables are *not* differentiated by case — "New Jersey" is not different from "NEW JERSEY". (However, it is different from "N.J.", which is different from "NJ".) Using the EXACT identifier in COUNTS causes case to be respected. Uppercase and lowercase representations of the same character strings are then considered different or unique, and their frequencies are tallied separately.

Leading or embedded *blanks* cause character strings to be considered different — "New Jersey" is different from "New Jersey" regardless of case. (Trailing blanks are ignored.) The PPL character function COMPRESS can be used to get rid of all leading, trailing and embedded blanks except for one blank between words:

```
COUNTS Addresses
[ DO #j USING V(1) .ON. ;
     IF V(#j) .CHARACTER. SET V(#j) = COMPRESS ( V(#j), 1 ) ;
     ENDDO ], OUT ....
```

In this example, a DO loop is used to reference all character variables in the file and set their values to compressed strings with only *single* embedded blanks. To get rid of *all* leading and embedded blanks, omit the second argument to COMPRESS (i.e., omit the ", 1").

7.7 COMBINING MULTIPLE RESPONSE VARIABLES

Multiple response variables or related variables may be combined and their frequencies tallied together as if they were one variable. Multiple response variables ("questions" in market research terminology) arise when more than one response ("answer") is possible. The responses are stored in multiple variables. For example, in response to the question, "What magazines do you subscribe to?", there may be from zero to ten (or as many as permitted) answers. The multiple variables may be combined so that the frequencies for the magazines reflect all respondents receiving them, and not just the respondents who named them first or second, and so on.

Related variables are simply those that logically might be tallied together — perhaps, because their values use the same scales or range of values. For example, the values of "Occupation of Head of Household" and "Occupation of Spouse" could be tallied together to show occupations in general.

7.8 Combining Adjacent Variables

The COMBINE identifier is included in the COUNTS command to give for each group: 1) a name for the group in quotes — up to 32 characters, 2) the *name* of the initial variable that begins the group of variables to tally together, and 3) the *number* of variables in the group:

```
COUNTS  PaceSet, COMBINE 'Stores' Store.1 4   'Electronics' VCR 4,
        OUT  PaceSetC  $
```

In this example, there are two groups of variables to combine — the group showing stores where electronic items were purchased (Store.1 to Store.4) and the group showing ownership of four electronic items (VCR, CD, Port.Phone and Ans.Mach). The values in each group of variables are combined and tallied as if they were the values of a single variable. Any other variables in the input file are treated as individual variables.

Figure 7.3 Frequencies and Statistics for a Multiple Response Grouping

```
      COUNTS  PaceSet    [ DROP   ID   Income;
          KEEP  Store?     VCR  CD  Port.Phone  Ans.Mach   .OTHERS. ],
          COMBINE 'Stores'  Store.1 4  'Electronics' VCR 4,
         OUT  PaceSetC  $

-----COUNTS of file PaceSet completed-----
80 cases and 12 variables were processed.
```

variable	value	count or stat	cum count	pct	cum pct
Stores	1	39.	39	48.75	-
	2	58.	97	72.5	-
	3	48.	145	60.	-
		-	-	-	-
	missing 1	175.	320	-	-
	missing 2	0.	320	-	-
	missing 3	0.	320	-	-
		-	-	-	-
	number of values	3.	-	-	-
	good n	69.	-	-	-
	total n*	80.	-	-	-
	responses	145.	-	-	-
	possible resp	320.	-	-	-
	vars combined	4.	-	-	-
		-	-	-	-
	mean	2.062068966	-	-	-
	median	2.	-	-	-
	mode	2.	-	-	-
	cases at mode	58.	-	-	-
	number of modes	1.	-	-	-
	variance	0.600287356	-	-	-
	std deviation	0.774782135	-	-	-
	se of mean	0.064342154	-	-	-
	low 95 conf mean	1.934891868	-	-	-
	up 95 conf mean	2.189246063	-	-	-
	low	1.	-	-	-
	high	3.	-	-	-
	range	2.	-	-	-
	q3-q1	2.	-	-	-
	sum	299.	-	-	-
	sum of squares	703.	-	-	-

When using COMBINE, each group of variables to be combined must be *together* in the input file. If necessary, use the KEEP instruction to group the variables correctly:

```
COUNTS  PaceSet
   [ KEEP  Store.?   VCR  CD  Port.Phone  Ans.Mach   .OTHERS.], COMBINE
'Stores' Store.1 4  'Electronics' VCR 4,  OUT  PaceSetC
```

The four store variables are positioned first in the file, the four electronic item variables next, and the remainder of the variables afterwards. The wildcard character "?" references variables that have a common prefix or suffix. The P-STAT system variable ".OTHERS." refers to all variables in the input file not explicitly mentioned after KEEP. It positions all these other variables *after* the multiple response ones. If necessary, precede the KEEP instruction with a DROP instruction if an ID or Case.Number variable should not be tallied.

A group name, the name of the first variable in the group and the number of total variables to be combined follow the COMBINE identifier. In the prior example, the first four variables are combined into one group and the next four variables are combined into another group. The first group will be labelled "Stores", the second group "Electronics". There may be many groupings — they are always assumed to reference sequential variables starting with the named variables. Figure 7.3 shows a portion of the output produced by COUNTS when COMBINE is used. (Cumulative percentages are not computed unless percents are based on responses instead of on respondents. See the section on changing the percentage base.)

7.9 Combining Non-adjacent Variables

If the variables to be combined are not grouped together, the MR.GROUPS identifier may be used. MR.GROUPS takes the form:

```
MR.GROUPS 'label' list-of-vars 'label' list-of-vars ...
```

where 'label' supplies a name for the combined group to identify it in the output, and list-of-vars defines a group of multiple response type variables.

The variables do not have to be adjacent. Phrases like store1 TO store6 can be used. MR.GROUPS will cause such a group of variables to be tabulated as if they were a single variable.

The label is required. Characters other than letters and digits may be used. The label can have as many as 32 characters.

7.10 Changing the Percentage Base

The base for the frequency percentages may be changed for single and multiple response variables, but is more often of concern with multiple response variables. By default, all percentages (both unweighted and weighted for single and multiple response variables) are based on "total n", the total number of cases. The asterisk in the COUNTS output indicates this.

The output produced for combined variables is slightly different from that produced for individual or single response variables. Three additional statistics are included:

```
responses     possible resp     vars combined
```

These follow the counts for "good n" and "total n" — see Figure 7.3.

"Responses" is the number of *good* (non-missing) responses counted. "Possible resp" is the *total* number of responses possible — that is, it is the number of cases times the number of variables combined. In Figure 7.3, there are 80 cases and 4 combined variables, or a total of 320 possible responses if everyone answered with the maximum number of responses. The 145 good responses and the 175 missing responses ("missing 1") yield the total of 320 possible responses. "Vars combined" is just the number of variables in the grouping.

"Good n" is the number of *cases* that have *at least one non-missing response*. "Total n" is the total number of cases, including those without any good responses at all. (Here, 80 - 69 or 11 cases had no good responses for any of the Store variables.) This parallels the situation in single response variables — "good n" is the number of cases with non-missing values and "total n" is the total number of cases.

The BASE identifier specifies an alternate percentage base. One of these arguments follows BASE:

```
GOOD          TOTAL          RESPONSES          POSSIBLE          LOCAL
```

BASE TOTAL is the default unless BY is used with interleaved output, described later in this chapter. Percentages are based on the total number of cases when the percentage base is BASE TOTAL. Another option is BASE GOOD, which bases percentages on the number of good cases. BASE RESPONSES bases percentages of *combined* (multiple response) variables on the number of good responses. For noncombined (single response) variables, this is the same as BASE GOOD. BASE POSSIBLE bases percentages of combined variables on the total number of responses possible. For noncombined variables, this is the same as BASE TOTAL.

LOCAL percentaging is the default when using BY with interleaved output. LOCAL can only be used with this form of output, which is described later in the chapter.

7.11 Selecting the Results Columns of the Output

The COUNTS output file normally contain four columns of results unless the percents are omitted because NO VALUES has been used. More columns are added if WEIGHT is used. The RESULTS identifier may be used to select which of these columns are to be placed in the output file.

```
COUNTS  PaceSet,  RESULTS  1  3  $
```

creates an output file with four columns: the variable name, the value, the count, and the percent. The cumulative count and the cumulative percent are omitted.

Figure 7.4 COUNTS Output with LABELS, Result Selection, and VALUES

```
COUNT  PaceSet [ KEEP Age Sex Income.Groups ],
        LABELS 'PaceSet.Lab',
        RESET, VALUES,  RESULTS  1 3  $


COUNTS of file PaceSet    PAGE 1

                                  count
                                    or
variable       value               stat     pct

Age            Under 30              28   35.
               30 to 50              27   33.75
               Over 50               23   28.75

Sex            Male                  39   48.75
               Female                40   50.

Income.Groups  Under $20,000          2    2.5
               $20,000 to $40,000    50   62.5
               $40,000 and Up        20   25.
```

```
COUNTS PaceSet, WEIGHT W.Factor,  RESULTS 1 5 $
```

creates an output file with four columns: the variable name, the value, the count and the weighted count. The variable name and the value are always in the output file. If BY is used, the BY value will also be present. Only the columns which contain values and percents that can be selected.

1. selects count or statistic

2. selects cumulative count
3. selects percent
4. selects cumulative percent
5. selects weighted count or statistic
6. selects weighted cumulative count
7. selects weighted percent
8. selects weighted cumulative percent

Figure 7.4 illustrates COUNTS output with results selection, labelling for the categorical values and no summary statistics. Use of RESET to turn off all printout, followed by VALUES to request the individual values rather than any summary statistics is described in the following section.

7.12 SELECTIONS OF STATISTICS

The COUNTS command produces the default sections of statistics shown in the past figures. These sections are: VALUES, MISSINGS, NS and STATS. Additional statistics may be requested — these sections are: HARMONICS, MOMENTS, PERCENTILES and NORMALS. Alternatively, all or some of the statistics may be omitted from the COUNTS output.

7.13 Default Sections of Statistics

VALUES refers to the frequencies and percentages of unique values, the initial section of the COUNTS output. MISSINGS refers the counts of any of the three types of missing values observed in the data values, the second section of the default output. NS provides the numbers of unique values, good cases and total cases. This section also includes the numbers of responses, possible responses and combined variables when COMBINE or MR.GROUPS is used. STATS is the section of summary statistics — the mean, median, mode, variance, standard deviation, standard error of the mean, 95% confidence intervals, low, high, range, interquartile range, sum and sum of squares.

Any of these sections of COUNTS output may be suppressed by including NO and the particular section in the COUNTS command:

```
COUNTS  Taxes  [ KEEP  District ],  NO STATS, ....
```

In this example, the section of summary statistics is suppressed from the output. If just a reduced selection of summary statistics is desired, use BASIC in the command:

```
COUNTS  Taxes  [ KEEP  District ],  BASIC, ....
```

Only the mean, standard deviation, low and high are included in the statistics section when BASIC is used.

Alternatively, RESET may be used in the COUNTS command to turn off the list of output options. It is then followed by the names of the desired sections:

```
COUNTS  PS134  [ DROP  Student.No ],  RESET,  VALUES,  MISSINGS, ..
```

Here, the output file contains just the frequencies and percentages of unique values and missing values. If no selections follow RESET, an error results.

7.14 Optional Sections of Statistics

HARMONICS, MOMENTS, PERCENTILES and NORMALS are optional sections of statistics that may be requested in COUNTS. Include the identifier or identifiers of desired sections in the command:

```
COUNTS  VirusII,  HARMONICS,  PERCENTILES, ....
```

These additional sections follow the default statistics. Including ALL in the COUNTS command results in all optional sections of statistics; the specific identifiers do not need to be used. ALL may be followed by NO and the name of a specific section to exclude it from the output.

Figure 7.5 shows all sections produced by COUNTS except for the VALUES section. This is often an appropriate option for continuous variables. Notice that the listing shows full double precision for all numbers.

Figure 7.5 COUNTS Output File with All Statistics Except Values

```
      TITLES  T1  'File PaceSet:  Summary Statistics for Income'  $

      COUNTS  PaceSet  [ KEEP  Income ],  ALL,  NO VALUES,
              TITLES,  OUT  IncomeCt $
```

File PaceSet: Summary Statistics for Income

variable	value	count or stat
Income	missing 1	0.
	missing 2	5.
	missing 3	3.
		-
	number of values	65.
	good n	72.
	total n*	80.
		-
	mean	34398.0556
	median	32510.
	mode	25900.
	cases at mode	2.
	number of modes	7.
	variance	87125742.6448
	std deviation	9334.1171
	se of mean	1100.0363
	low 95 conf mean	32204.6456
	up 95 conf mean	36591.4655
	low	18000.
	high	54600.
	range	36600.
	q3-q1	15025.
	sum	2476660.
	sum of squares	91378216000.
		-
	harmonic mean	32000.8737
	geometric mean	33178.502
	mean pos values	34398.0556
		-

```
skewness                          0.4668
kurtosis                         -0.6833
se of skewness                    0.2829
se of kurtosis                    0.5588
coef variation                   27.1356
corrected ssq              6186016000.
second moment                85915662.8858
third  moment              363921513429.7415
fourth moment          0.16837430752159d+17
                                      -
percentile 01                    18000.
percentile 05                    21260.
percentile 10                    23752.
percentile 25                    26725.
percentile 50                    32510.
percentile 75                    41750.
percentile 90                    48490.
percentile 95                    52340.
percentile 99                    54600.
                                      -
shapiro-wilk  W                   0.9434
sw:prob norm <W                   0.0054
anderson-darl A2                  1.0016
ad:prob norm >A2                  0.012
```

Scientific notation is used to preserve the precision of extremely large or small numbers — for example, the fourth moment.

The HARMONICS section contains the harmonic mean, the geometric mean and the mean of the positive values. The *harmonic mean* is the number of good values divided by the sum of the reciprocals of the values. It is appropriate when the data values are different costs or rates per given item or distance. An example is a constant amount of money invested each month to purchase a specific stock whose price varied from month to month. The harmonic mean of the prices gives the average price of the stock per share.

The *geometric mean* is the nth root of the product of n numbers, where n is the number of good values. (It is calculated as the antilog of the arithmetic mean of the logs of the number.) The geometric mean is appropriate when averaging ratios, rates of change, index numbers, and so on. An example is the average of percentages of increase in production from quarter to quarter.

The *mean of positive values* is just the arithmetic average of the positive values (zero and negative values are excluded). An example is the size of the average loan to bank customers, where zero represents no loan and should not deflate the average loan amount.

The MOMENTS section contains measures of skewness and kurtosis, their standard errors, the coefficient of variation, the corrected sum of squares and the second through the fourth moments. *Skewness* refers to the symmetry of a frequency distribution. It is zero when the distribution is symmetrical. When it is negative, the distribution is skewed to the left — there is a long tail toward the smaller values. When it is positive, it is skewed to the right — the long tail is toward the larger values.

Kurtosis refers to the flatness of the distribution in relation to the normal distribution. It is zero when the distribution is neither flat nor peaked. Negative values of kurtosis indicate a flatter distribution with lighter tails than the normal distribution; positive values indicate a more peaked distribution with heavier tails. The *standard errors* of skewness and kurtosis give the sample-to-sample variation of these measures. The ratios of skewness and kurtosis to their standard errors may be used as estimates of normality.

The *coefficient of variation* gives the percentage of variation in the sample. It is the standard deviation expressed as a percentage of the mean. The *corrected sum of squares* is the sum of the squared deviations of each value from the mean value. (The "uncorrected" sum of the squares given in the STATS section of the COUNTS output is the sum of the squared values, uncorrected for the mean value.) The second, third and fourth *moments* about the mean are analogues to the mechanical moments of forces operating on a lever at various distances from the fulcrum. They are used in computing skewness and kurtosis.

The PERCENTILES section contains the 1st, 5th, 10th, 25th, 50th, 75th, 90th, 95th, and 99th percentiles. The 50th percentile is the median or middle value. Similarly, the 25th percentile is the value one quarter of the way through the ordered data values, and so on. When WEIGHT is used in the COUNTS command, both unweighted and weighted percentiles are produced. Use the PERCENTILES *command* to compute any arbitrary percentiles or if interpolated percentiles are desired.

The NORMALS section contains several *tests of normality* of the data distribution. The Shapiro-Wilk W and the Anderson-Darling A^2 statistics and their associated probabilities are given. In Figure 7.5, the probability of obtaining a W statistic less than .9434 is .0054. The assumption of normality is therefore rejected at the .01 level of significance. (Values of W greater than .9 are expected in normal distributions — the actual expected values increase with the size of the sample.) The Shapiro-Wilk test is calculated when the size of the sample is between 3 and 2000. (Royston, 1982, 1988)

The Anderson-Darling test is most appropriate for large samples greater than 2000. In Figure 7.5, the probability of obtaining an A^2 statistic greater than 1.0016 is .012. The assumption of normality is therefore rejected at the .05 level of significance. A^2 ranges from 0 to 2 approximately. (Pettitt) The NORMAL tests require that all the values of a variable fit in memory at the same time. See the section on "Ordering the Frequencies".

Figure 7.6 A Table of MEANS produced by the COUNTS command

```
 COUNT  PaceSet [ DROP  ID  ],  RESET,  STATS 1,  NO LIST,
        OUT  PacesetM  $

 LIST   PacesetM,  STUB Value  $


    FILE PacesetM
COUNTS of file PaceSet    PAGE 1

value    variable          count or stat

mean     Age                  1.935897436
         Sex                  1.506329114
         Income           34398.055555556
         Num.TV               3.1
         VCR                  0.5625
         CD                   0.35
         Port.Phone           0.4375
         Ans.Mach             0.4625
         Store.1              2.066666667
         Store.2              2.142857143
         Store.3              2.085714286
         Store.4              1.972972973
         Income.Groups        2.25
```

7.15 Selecting Individual Statistics

Row selection of individual statistics within a statistics section is done by following the section name with the number of the rows within that section that are to be produced. A table identifying the line numbers in each section is found in the SUMMARY at the end of the chapter.

```
COUNT PaceSet,  RESET,  PERCENTILES 2 8  $
```

produces just the 5th and 95th percentiles for all the variables in the file.

```
COUNT   PaceSet,   MISSING  1   $
```

produces the default statistics except that the MISSING section includes only missing type 1.

```
COUNT  PaceSet,  RESET,   STATS  1 7 $
```

produces just the means and standard deviations for all the variables in the file.

The correct number for each statistic within its section can be determined by consulting the table in the SUMMARY at the end of the chapter.

7.16 Computing Percentiles

There are three methods available for the computation of percentiles: NPLUS1, FIRST and GROUPED. The methods are used in both the percentiles section and in computing the median which is just a special case of percentiles.

1. METHOD.DEFAULT requests the default, which is to use the NPLUS1 method if the data permit, and if not, to use the FIRST method.

   The FIRST method is needed (1) when a weighted percentile is requested and one or more of the weights are not integers, or (2) when the number of cases exceeds 2.147 billion.

2. METHOD.NPLUS1 requests the NPLUS1 method. The result is set to missing if there are fractional weights, or if the N exceeds 2.1437 billion.

   Suppose a median is wanted when N=200, and the cases are ordered from x(1) to x(200). We take N+1 times P, the sought percentile, which would be 200+1 times .5, or 100.5 . The 100.5 is split into IP, the integer part (100), and FP, the fractional part (.5). The median is:

   ```
   (1-fp) * x(ip) + fp * x(ip+1),
   ```

   where we use x(1) for x(0), and x(n) for case x(n+1).

3. METHOD.FIRST requests the FIRST method, which returns the value of the first case at or beyond ordered position N*P.

4. METHOD.GROUPED requests the GROUPED method, which is intended for grouped data, ie, where each integer represents a range. For example, 1 means a sale price of $100,000 to $120,000, 2 means a sale price of $120,000 to $135,000, and so forth, and you want to know how far into the range the median may be.

   Given values 1,2,3,4,5 with frequencies of 10,30,30,20 and 10 respectively, the median is 3.3333, because the 50% point is one third of the way through the 30 threes. The 99% result would be 5.9 .

   The assumes that the distribution is even within each range. A missing result occurs if the data are not all integers.

7.17 OUTPUT FILE FORMAT OPTIONS

Various options affect the way the results produced by the COUNTS command are written in the *output file*. The contents of the output file, once that file is produced, remain constant. Options in the USE argument or in the LIST command can vary the *display* of the file, but not change what it contains. Output file format options include:

1. the presence or absence of break lines
2. the maximum number of unique values included
3. the number of decimal places in the percentages
4. the order of the frequencies of unique values
5. printing all possible integer values in a range

7.18 Break Lines, Values and Percentage Places

"Break" lines are inserted in the COUNTS output file between the various sections of statistics to improve readability. The lines contain blanks and missing values. Figure 7.5 shows the break lines between the various sections. Using NO BREAK in the COUNTS command suppresses these lines from the output file. If LIST is being used to display a COUNTS output file that contains break lines, a PPL instruction can be included in the LIST command to remove them:

```
LIST  IncomeCt [ IF  Value  = ' ',  DELETE ],  STUB Variable  $
```

The MAX identifier gives the maximum number of unique values that should be included in the output file. Frequencies are computed for all unique values, but values beyond the maximum are lumped together in one category. When MAX is not used, frequencies for all unique values are included in the output file. It may be useful to use MAX when the approximate number of unique values is not known. For example, the *continuous* variable "Income" could have as many distinct values as there are cases in an input file. In Figure 7.5, NO VALUES is included in the COUNTS command to avoid an output file with a large number of cases giving frequencies. Alternatively, MAX could be used:

```
COUNTS  PaceSet  [ KEEP  Income ],  MAX 12, ....
```

The output file produced by this command would contain frequencies for the first 12 unique values; the frequencies for the remainder of the values would be totalled into a single category called "remaining values."

By default, there are two decimal places in the percentages and cumulative percentages in the COUNTS output file. The identifier PCT.PLACES may be included in the COUNTS command to request a different number of places. Integers from zero through nine are valid arguments. If an existing output file from COUNTS has more decimal places than you want in a listing, the identifier PLACES may be included in the LIST command to specify the desired number of places for all numeric variables.

7.19 Ordering the Frequencies

The VALUES section in the COUNTS output file contains the frequencies and cumulative frequencies of all unique values. This section is in *ascending order* of the *values*. The DOWN identifier may be used in the COUNTS command to reverse this and position the values in descending order. UP is assumed when DOWN is not specified.

The VALUES section may be ordered by the observed *frequencies* or observed *weighted frequencies* of the unique values. The FREQUENCIES or W.FREQUENCIES identifier is used in COUNTS to request this. DOWN or *descending order* is assumed when the VALUES section is ordered by frequencies. UP may be specified if ascending order is desired. The most and least common values are obvious when frequency ordering is used. FREQ and W.FREQ are synonyms for FREQUENCIES and W.FREQUENCIES, respectively.

Figure 7.7 COUNTS: BY Variable, 3 FORMATS, and TAGS

```
COUNT PaceSet [ KEEP Age, Sex ], LABELS 'PaceSet.lab', RESET,  NS 2, STATS 1,
BY AGE,     TAGS $

       Age         variable  value    count or stat

       Under 30    Sex       good  n   28.
                             mean       1.285714286

       30 to50     Sex       good  n   27.
                             mean       1.592592593

       Over 50     Sex       good  n   22.
                             mean       1.681818182

       all cases   Sex       good  n   79.
                             mean       1.506329114


             FORMAT 2, TAGS $

       variable  Age         value    count or stat

       Sex       Under 30    good  n   28.
                             mean       1.285714286

                 30 to 50    good  n   27.
                             mean       1.592592593

                 Over 50     good  n   22.
                             mean       1.681818182

                 all cases   good  n   79.
                             mean       1.506329114


              FORMAT 3, TAGS $

        variable  value    Age         count or stat

        Sex       good  n  Under 30     28.
                           30 to50      27.
                           Over 50      22.
                           all cases    79.

                  mean     Under 30      1.285714286
                           30 to50       1.592592593
                           Over 50       1.681818182
                           all cases     1.506329114
```

There are *no* size constraints for the COUNTS command *unless* (1) FREQUENCIES or W.FREQUENCIES is specified, or (2) the NORMALS section is requested. Typically, as many passes as necessary are made through the input data file to compute the COUNTS statistics. The number of passes depends on the number of variables in the input file and the number of unique values each has. When frequency ordering is requested, a single variable's values must fit in memory. COUNTS must compute the frequencies for all values to be able to order them by frequency — thus, the number of unique values is limited.

The maximum number of *unweighted numeric* values that may be ordered by frequency for one variable in the Whopper/1 size is 25,000.

If WEIGHT is used in the COUNTS command to request weighted counts and percentages, this maximum number of values is reduced by 33%. A character variable of length 16 reduces the number of values by approximately 50%. Use the PPL instructions KEEP and DROP in the COUNTS command when FREQ or W.FREQ are used to avoid computing frequencies and statistics on variables not of interest — no point in aborting the COUNTS command because of an identification variable or a character variable that you forgot was in the file.

The use of VERBOSE causes all statistics lines to print even if they have a missing value. This will happen when there are too few good values to compute a statistic. If VERBOSE is not used, only statistics which have a non-missing value are placed in the output file.

7.20 BY GROUPS

A single character or numeric BY variable may be supplied. When BY is used the output contains counts and statistics for the entire file as well as counts and statistics for each subgroup defined by the BY variable. The file does *NOT* need to be sorted on the BY variable.

Figure 7.7 illustrates the use of COUNTS with a BY group variable. The variable Age in the PaceSet file has three values. Therefore, using variable AGE as the BY variable causes four analyses for each of the other variables in the file; three for each of the values on AGE and one for all cases. There are 3 possible arrangements of the variables in the output report. The assumed output has values of the BY variable as the outermost level. This can be explicitly requested by using "FORMAT 1". Each BY variable value is followed by the current analysis variable and its values. This layout, which is now assumed, was not available in P-STAT Version 2.

"FORMAT 2" produces a layout that was the assumed layout in P-STAT Version 2. "FORMAT 3" produces the layout that was produced in version 2 by using "NO INTERLEAVE". When there is no interleaving, all the statistics for one group are followed by all the statistics for the next group.

The default percentaging when BY and INTERLEAVE are in use is LOCAL based percentage. The base is the count in the "all cases" line. Explicit use of an alternate base such as BASE TOTAL, causes across value within by-group percentaging to be used, even though the output is interleaved. When NO INTERLEAVE is specified, LOCAL percentaging is not used and the assumed base for percentaging is the total count.

ALL.CASES LAST is assumed when using BY. It causes the "all cases" line to follow the individual by-group lines. ALL.CASES FIRST causes it to precede the by-group lines for a given value or statistic. NO ALL.CASES causes the "all cases" lines to be omitted. ALL.CASES can also be used with non-interleaved BY output.

REFERENCES

1. Pettitt, A.N. (1977). "Testing the Normality of Several Independent Samples Using the Anderson-Darling Statistic, *Applied Statistics*, 156-161.

2. Royston, J.P. (1982). "An Extension of Shapiro and Wilk's W Test for Normality to Large Samples," *Applied Statistics*, 31, 115-124.

3. Royston, J.P. (1988). "Correcting the Shapiro-Wilk W for Ties," Journal of Statist. Comput. Simul., 31, 237-249.

4. Stephens, M.A. (1974). "EDF Statistics for Goodness of Fit and Some Comparisons," *Journal of the American Statistical Association*, 69, 730-737.

SUMMARY

COUNTS

```
COUNTS  Region1  $

COUNTS  PaceSet  [ DROP  ID  Income ],  OUT  PaceSetC,
        TITLES,  PR  'usr/tom/PaceSet.lst'  $

COUNTS  Players
        [ KEEP  Q12.1 Q12.2 Q12.3 Q12.4 Q12.5  Q25.?  .OTHERS. ],

        COMBINE  'Q12.group' Q12.1  5   'Q25.group'  Q25.1  4,
        PCT.PLACES 1,   LIST.BY,  RESET, STATS,  OUT  Players2  $
```

Required:

COUNTS fn

provides the name of the P-STAT system file that is input to the COUNTS command. Frequencies and percentages (individual, cumulative and weighted, if applicable) are calculated for all *unique* good and missing values of *all* variables (numeric and character) in the input file. In addition, summary statistics are computed for each numeric variable. Unless other sections are explicitly requested, the COUNTS output file includes:

VALUES:
frequencies and percentages of all unique values

MISSINGS:
frequencies and percentages of three types of missing values

NS:

number of values	good n	total n	

STATS:

mean	median		
mode	cases at mode	number of modes	
variance	std deviation	se of mean	
low 95 conf mean	up 95 conf mean		
low	high	range	q3-q1
sum	sum of squares		

Optional identifiers either suppress or include various sections of statistics.

The COMBINE identifier may be used to group multiple response or other variables together — frequencies, percentages and statistics are computed for their *combined* values. The optional OUT identifier supplies a name for an autosave (“permanent”) output file of results. The output file is automatically listed with appropriate format options, unless NO LIST is requested. The listing may be directed to a disk file or printer by including appropriate identifiers in an argument that is used in the LIST command. (COUNT is a synonym for COUNTS.)

Optional Identifiers:

COMBINE “name” vn nn “name” vn nn ....

requests that the group of variables starting with *vn* and including a total of *nn* variables be combined into “one” variable, that the second group of variables starting with the *second vn* and including a total of *nn* variables be combined into another “single” variable, and so on. Frequencies, percentages and summary statistics are calculated for the *combined* values of each group of variables. The variables are typically *multiple response* variables, but any reasonable combinations are valid. However, numeric and character variables may not be combined.

A group name, in quotes, is used to identify the combined group. It can be up to 32 characters long,

```
COUNTS  Players
 (KEEP  Q12.1 Q12.2 Q12.3 Q12.4 Q12.5  Q25.?  .OTHERS.),

  COMBINE   'Question 12'  Q12.1  5
            'Question 25'  Q25.1  4,  ....
```

If necessary, the KEEP instruction is used to position the variables to be combined so that they are *together* in the input file. In this example, the five multiple response variables Q12.1 to Q12.5 are positioned together as the first group of variables to be combined, and the four variables beginning with “Q25.” are positioned as the next group of variables to be combined. (The “?” is the

character — any variable names beginning with “Q25.” will match. The P-STAT system variable “.OTHERS.” refers to all other variables in the input file not already selected.) Other variables in the input file are treated individually.

When COMBINE is used, the NS section of output includes:

number of values	good n	total n*
responses	possible resp	vars combined

The “good n” and “total n” are counts of *respondents*; the “responses” and “possible resp” are counts of actual and possible *responses*. An asterisk follows “total n”, the value that is the default *base of the percentages*. See the BASE identifier for information on specifying an alternate base. Also see the MR.GROUPS identifier for combining non-adjacent variables.

EXACT

causes *case* to be respected in determining unique values of character variables. Normally, “JONES” is not considered different from “Jones” — an occurrence of each string yields a frequency of two for the value “JONES” (or whichever string comes first). Using the EXACT identifier results in a frequency of one for each of the values.

LABELS ’fn’

provides the name of an external file containing value labels to be used in place of categorical data values. See the Chapter on Titles, Labels for details on usage.

LIST

requests that the output file produced by COUNTS be listed on the current output device (typically the terminal). This is assumed unless NO LIST is used:

```
COUNTS  Forest12,  OUT  Counts12,  NO LIST  $
```

Typically, the output file is automatically displayed, appropriately formatted with the variables whose values are counted *stubbed*. Stubbing positions the variables on the left and on each page of multi-page listings, and it blanks out repetitions of the variable names. LIST is a synonym for LIST.STUB — the

default stubbed listing. The LIST.BY identifier may be used if a narrower listing with *headings* is desired.

An alternate print destination can be supplied. Other, LIST-specific identifiers may be part of a character string argument to USE. The USE argument is used in the LIST command generated by COUNTS. (See the USE identifier for specifics.)

LIST.BY

requests that the output file produced by COUNTS be listed with the variable names as *headings* running across the output. This reduces the width of the default COUNTS listing somewhat.

MR.GROUPS 'cs' vn vn vn 'cs' vn vn

If the variables to be combined are not grouped together, the MR.GROUPS identifier may be used as an alternative to the COMBINE identifier.

```
MR.GROUPS 'label' list-of-vars 'label' list-of-vars ...
```

OUT fn

supplies a name for the output file that contains the frequencies, percentages and summary statistics. It is automatically listed as COUNTS completes. The output file is a regular P-STAT autosave file that is available for future use as input to P-STAT commands. When OUT is not used, the COUNTS results are put in a temporary file whose name begins with "WORK" — it disappears when the P-STAT run ends.

PR 'fn'

specifies a print destination. The listing is directed to this destination (a disk file or a printer) instead of to the terminal. PR, without an argument, directs listings back to the primary output device (the terminal).

SHOW.COMMAND

causes the generated LIST command to be printed.

TITLES

titles requests that all top and bottom titles be printed when the OUT file is listed. Titles are defined using the TITLES command (see the TITLES and LABELS chapter).

USE 'cs'

gives identifiers and their arguments that should be used in the LIST command generated by COUNTS. Any other identifiers that are normally permitted in the LIST command (see the LIST chapter) may be part of the character string argument to USE:

```
COUNTS  Region1,  USE  "PLACES  3,  GAP  1",  OUT ....
```

The entire character string argument following USE *must be enclosed in quotes* — use double quotes to enclose single ones, or vice versa. The COUNTS output file could also be listed using the LIST *command* and any of its options at any time.

WEIGHT vn

gives the name of the variable whose values are case weights. Instead of each case "counting as one," it counts as specified by the weight variable. A negative or missing weight value is treated like a weight of zero.

Optional Identifiers: Statistics

ALL

requests that *all* possible statistics produced by COUNTS be calculated and included in the output file. In addition to the statistics typically computed (shown at the start of the summary section on COUNTS), the output file includes these statistics for numeric variables:

HARMONICS:			
harmonic mean	geometric mean	mean positive values	
MOMENTS:			
skewness	kurtosis	se of skewness	se of kurtosis
coef variation	corrected ssq		
second moment	third moment	fourth moment	
PERCENTILES:			
percentile 01	percentile 05	percentile 10	percentile 25
percentile 50	percentile 75	percentile 90	percentile 95
percentile 99			
NORMALS:			
shapiro-wilk W	sw:prob norm < W	anderson-darl A2	ad:prob norm > A2

The ALL identifier may be used in the COUNTS command with NO followed by a specific section of the statistical output:

```
COUNTS  States,  ALL,  NO HARMONICS,  NO PERCENTILES, ....
```

This suppresses the harmonics and percentiles sections of the statistical output.

BASE arg

specifies an alternate base for percentages. The argument that may follow BASE is one of these:

```
GOOD          TOTAL          RESPONSES      POSSIBLE      LOCAL
```

The default (unless BY and INTERLEAVE are used) is TOTAL — that is, the count of *total cases*, including those with missing values, is the denominator in the percentage calculations. (An asterisk in the output indicates the percentage base.) When BASE GOOD is specified, the count of cases with *non-missing* values of the particular variable is the denominator. If COMBINE or MR.GROUPS is also used, the count of cases with *at least one non-missing* value of the combined variables is the denominator. LOCAL is the default when BY and INTERLEAVE are used, and is not valid otherwise.

RESPONSES and POSSIBLE are valid arguments when COMBINE or MR.GROUPS is also used. When BASE RESPONSES is used, the count of *non-missing responses* is the denominator for combined variables and the count of *non-missing cases* is the denominator for other variables. When BASE POSSIBLE is used, the count of *possible responses* (cases times combined variables) is the denominator for combined variables and the count of *total cases* is the denominator for other variables.

BASIC

requests that the four basic statistics — the mean, the standard deviation and the low and high values — be computed.

The other statistics usually computed in the STATS section (the median, mode, variance, etc.) are not included in the BASIC output. BASIC is the same as using STATS 1 7 11 12.

HARMONICS nn nn

without arguments requests that the entire harmonics section of the statistical output be included in the output file. This section includes the harmonic mean, the geometric mean, and the arithmetic mean of

only the positive values. The optional numeric arguments specify which of the three statistics are to be used. For example HARMONICS 3 produces just the mean of the positive values.

METHOD.DEFAULT

requests the default, which is to use the NPLUS1 method if the data permit and if not, to use the FIRST method.

METHOD.NPLUS1

requests the NPLUS1 method. The result is set to missing if there are fractional weights or if the N exceeds 2.1437 billion.

METHOD.FIRST

requests the FIRST method, which returns the value of the first case at or beyond ordered position N*P.

METHOD.GROUPED

requests the GROUPED method, which is intended for grouped data, i.e., where each integer represents a range.

MISSINGS nn nn

without arguments requests that counts of the three types of missing values be included in the COUNTS output. This is assumed, unless NO MISSINGS is used to suppress this portion of output. The optional numeric arguments specify which of the MISSINGS are to be included.

MOMENTS nn nn

without arguments requests that the moments section be in included in the output file. This section includes measures of skewness and kurtosis, their standard errors, the coefficient of variation, the corrected sum of square, and the second through fourth moments. The optional numeric arguments specify which of the nine statistics from this section are to be included. MOMENTS 7 8 9 produces just the 2nd, 3rd, and 4th moments.

RESET

turns off the default output assumed by the COUNTS command. RESET should be followed by the specific sections of output desired:

```
COUNTS  States,  RESET,  VALUES, ....
```

This requests that only the section giving the frequencies for unique values be included in the output file. The MISSINGS section, for example, is not included.

NORMALS nn nn

without arguments requests that the Shapiro-Wilk and Anderson-Darling tests of the normality of the distribution and their probabilities be included in the COUNTS output. The optional numeric arguments can be used to select any of the four rows in this section. Use of NORMALS requires that all values of a variable fit in memory at the same time.

NS nn nn

without arguments requests that the counts (NS) section — the count of different values, the good n and the total n — be included in the output file. This is assumed, unless NO NS is used in the COUNTS command. The optional numeric arguments can be used to select any of the three rows in this section. Three additional rows are produced for combined variables.

PERCENTILES nn nn

without arguments requests that the entire percentiles section be included in the output produced by

COUNTS. The 1st, 5th, 10th, 25th, 50th, 75th, 90th, 95th and 99th percentiles are calculated. (Use the PERCENTILES *command* to request any arbitrary percentiles.) The optional numeric arguments can be used to select any of the nine rows in this section.

VERBOSE

requests that requested statistics be included in the output file even when they are missing because they cannot be computed.

RESULTS nn nn

specifies which of the counts and percents columns are to be included in the output. RESULTS 1 prints just the counts. RESULTS 1 5 prints counts and weighted counts, etc.

Optional Identifiers: BY Groups

BY vn

specifies a single numeric or character variable which defines subgroups. Counts and statistics are produced for the entire file and for each subgroup.

ALL.CASES arg

followed by FIRST or LAST controls the placement of the "all cases" group in the BY output. FIRST is assumed if nothing is specified. NO ALL.CASES can be used to omit the "all cases" group from the output file.

INTERLEAVE

is assumed for BY output. In interleaved format each value or statistic is listed with all by-groups before the next value or level is listed.

NO INTERLEAVE

causes the output to have all values and statistics for the first BY level followed by all values and statistics for the next BY level, and so on.

BASE LOCAL

is assumed for interleaved BY output. It shows the percent that each BY level is of the cases with that value. One of the other bases such as BASE TOTAL can be selected for use with interleaved BY output. BASE LOCAL can only be used with interleaved BY output.

FORMAT 1 2 3

The assumed output has values of the BY variable as the outermost level. This can be explicitly requested by using "FORMAT 1". Each BY variable value is followed by the current analysis variable and its values. This layout, which is now assumed, was not available in P-STAT Version 2. "FORMAT 3" produces the layout that was produced in version 2 by using "NO INTERLEAVE".

TABLE OF CODES

NS

1. number of values
2. good n
3. total n
4. responses
5. possible resp
6. vars combined

MISSING

1. missing 1
2 missing 2
3. missing 3

NORMALS

1. shapiro-wilk W
2. sw:prob norm <W
3. anderson-darl
4. ad:prob norm >A2

STATS

1. median
2. mean
3. mode
4. cases at mode
5. number of modes
6. variance
7. std deviation
8. se of mean
9. low 95 conf mean
10. up 95 conf mean
11. low
12. high
13. range
14. q3-q1
15. sum
16. sum of squares

HARMONICS

1. harmonic mean
2. geometric mean
3. mean pos values

MOMENTS

1. skewness
2. kurtosis
3. se of skewness
4. se of kurtosis
5. coef variation
6. corrected ssq
7. second moment
8. third moment
9. fourth moment

PERCENTILES

1. percentile 01
2. percentile 05
3. percentile 10
4. percentile 25
5. percentile 50
6. percentile 75
7. percentile 90
8. percentile 95
9. percentile 99

8
TURF

This beginning of this chapter covers basic usage of the TURF command. TURF stands for Total Unduplicated Reach and Frequency. The rest of the chapter covers the STEP identifier to implement the greedy method when the number of combinations makes direct computation impractical. It also covers other less usual features, methods, timings and limitations. TURF is frequently used in market research applications.

8.1 TURF COMMAND

Let us consider the following example: a file has the responses of 1000 cases on 40 items. Each item is a TV program which the respondent did or did not watch. You would like to know which group of 8 programs was reached by the largest number of respondents. A respondent is "reached" by a given group of programs if they watched at least one of the programs in the group.

TURF, to find the best group, evaluates each of the 76.9 million combinations of 8 items taken from a pool of 40, and writes a file identifying the 100 best combinations. This takes about a minute on a 2.4GHz pc. Weighting can be applied to the cases, or to the response values, or to the items. The "being reached" criterion of one response can be increased.

Adding options affects speed. The above one-minute run takes 12 minutes if case weights are used, and would take about 30 minutes if the other options are used.

8.2 Reach Versus Frequency

Reach and frequency are different measurements in TURF. The REACH score for a combination is the number of cases that have at least N positive responses on the variables in that combination. N is the reach threshold in use, which has a default setting of one.

The FREQ score for a combination is, for the reached cases, normally the total number of positive responses on the variables in the combination. The response.weights option causes the FREQ score to be the sum of the positive responses.

If a respondent indeed watched all eight programs in a group, the frequency count for that group would be increased by 8, but the reach count would only be increased by 1. If the input were hours watched rather than just watched or not watched, the response.weights option would sum the response values and thereby measure the impact of the combination.

8.3 Features of the TURF Command

1. Allows up to 210 items (i.e., variables).
2. Allows combinations of items up to size 60.
3. Allows several combination sizes to be done in one run. either independently or in a stepwise manner.
4. Allows tens of thousands of cases.
5. Allows weighting of cases.
6. Allows weighting of items.
7. Allows weighting of responses; this allows the intensity of a response to be utilized.

8. Allows setting a reach threshold of more than one.
9. Allows forcing designated items into every combination.
10. Allows limits on how many of a set of items can be placed in the combinations to be analyzed.
11. Writes a result / file containing the best combinations. The items within each combination are ordered by their importance.
12. Writes a template file for use by the TURF.SCORES command. It contains the reach score for each case. TURF.SCORES can be used to learn more about the people who were reached. It is documented in the manual “P-STAT: Basic Statistics”.
13. Takes 3.1 seconds for 1,000 cases on 40 items, 6 at a time on a 2.4 GHz PC. This evaluates 3.83 million groups.
14. Runs so rapidly that billions of combinations can be done.
15. Takes 18.5 minutes for 1,000 cases on 100 items, 6 at a time on a 2.4 GHz PC. This evaluates 1.192 billion groups. Note: using all options would take about 9 hours.
16. On the Windows/PC shows the percent of combinations already processed in a progress window.
17. Writes a detailed report when the command finishes.

8.4 OVERVIEW

Figure 8.1 A Typical Turf Report

```
----------TURF analysis for file tin completed-----------
| OPTIONS: none                                          |
|                                                        |
|       29 items were used in the analysis.              |
|      550 cases were read and used.                     |
|      213 cases had at least one positive response.     |
|                                                        |
| SIZE   4 evaluated 23,751 combinations:                |
|      212 was the best REACH, found in 2 combinations.  |
|      628 was the best FREQ in those 2 combinations.    |
|      729 was the best FREQ in any size 4 combination.  |
|                                                        |
| The FREQ score for a combination is the count          |
| of the non-zero responses for that combination,        |
| summed over the reached cases.                         |
|                                                        |
| REACH.RESULTS file work1 has the 100                   |
| combinations with the highest reach scores.            |
| The items are ordered by their REACH contribution.     |
|                                                        |
| Two reach.stats lines were included:                   |
|   1 cumulative reach.                                  |
|   2 unique reach provided by each item.                |
|                                                        |
| Time: .1 seconds.                                      |
----------------------------------------------------------
```

Figure 8.2 A Typical REACH RESULTS file

```
TURF results ordered by REACH, from input file tin using 29,4

size
and              pct                 item  item  item  item
rank  reach  reached  freq  stats    .1    .2    .3    .4

4_1     212   38.545   628          VAR5  VAR4  VAR3  VAR23
                           cum.r    186   208   210   212
                           unique   16    8     2     2

4_2     212   38.545   602          VAR5  VAR4  VAR3  VAR28
                           cum.r    186   208   210   212
                           unique   18    8     2     2

4_3     211   38.364   660          VAR5  VAR4  VAR3  VAR13
                           cum.r    186   208   210   211
                           unique   19    8     2     1
```

The default is to write the 100 combinations that have the highest reach value. If some are tied, they are ordered by their freq value. In Figure 8.2, just the first three are shown.

The best combination contains VAR5, VAR4, VAR3 and VAR23. VAR5 is first because it was the best single item of the four. It reached 186 cases by itself; none of the others did as well.

VAR4 is shown next because VAR5 with VAR4 reached more cases (208) than VAR5 + VAR3, or VAR5 + VAR23. And so on. This is shown on the "cum.r" line, which stands for cumulative reach. The default is to include this extra line.

The next line, "unique", was requested in this TURF run. It shows, for each item, how much reach would be lost if that item were dropped from the combination. VAR5 reached 16 cases that the other three items in the first combination did not.

Figure 8.3 TURF Report From a Longer Run

```
---------TURF analysis for file work2 completed----------
| OPTIONS: none                                          |
|                                                        |
|      100 items were used in the analysis.              |
|    1,000 cases were read and used.                     |
|      973 cases had at least one positive response.     |
|                                                        |
| SIZE   6 evaluated 1,192,052,400 combinations:         |
|      941 was the best REACH, found in 1 combination.   |
|    1,956 was the FREQ value in that combination.       |
|    1,983 was the best FREQ in any size 6 combination.  |
|                                                        |
| The FREQ score for a combination is the count          |
| of the non-zero responses for that combination,        |
| summed over the reached cases.                         |
```

```
|                                                          |
| REACH.RESULTS file work101 has the 100                   |
| combinations with the highest reach scores.              |
| The items are ordered by their REACH contribution.       |
| Cumulative reach is shown.                               |
|                                                          |
| Time: 18 minutes, 35.5 seconds.                          |
------------------------------------------------------------
```

8.5 TURF EXAMPLES

Figure 8.4 A TURF Run Using Defaults

```
------------------file ddd---------------

case.id  www      v1    v2    v3    v4    v5

   9001    1       1     1     0     0     0
   9002    1       1     1     0     0     0
   9003    1       1     1     0     0     0
   9004    1       1     1     0     0     0
   9005    1       1     1     0     0     0

   9006    1       1     1     0     0     0
   9007    1       0     1     0     0     0
   9008    1       0     0     1     0     0
   9009    1       0     0     1     0     0
   9010    1       0     0     1     0     0

   9011    1       0     0     1     0     0
   9012    1       0     0     0     1     0
   9013    1       0     0     0     1     0
   9014    1       0     0     0     1     0
   9015    3       0     0     0     0     2
   9016    3       0     0     0     0     2

   TURF ddd  [DROP case.id  www],   SIZE 3,  REACH.RESULTS rrr $
   LIST rrr $
```

The dataset in Figure 8.4 has a case identifier (case.id), a case weight (www), and the responses to five variables. The responses are zeros, ones and two twos. This data set is used to show various TURF options. The twos on the final 2 cases are treated differently from ones *only* when the RESPONSE.WEIGHTS option is in use.

This command:

```
TURF ddd  [DROP case.id  www],   SIZE 3,  REACH.RESULTS rrr $
```

uses PPL (the P-STAT Programming Language) to drop the first two variables, leaving five items for the turf analysis. Using [Keep v1 to v5] would have done the same thing.

SIZE 3 tells the command to look at the items in groups of 3. REACH.RESULTS RRR creates a file named RRR that identifies the best groups. The TURF command prints an extensive report detailing the options that were used, the steps that were taken and the output files that were created.

Figure 8.5 TURF Report

```
----------TURF analysis for file ddd completed-----------
| OPTIONS: none                                          |
|                                                        |
|        5 items were used in the analysis.              |
|                                                        |
|       16 cases were read and used.                     |
|       16 cases had at least one positive response,     |
|          making that the maximum possible reach.       |
|                                                        |
| SIZE   3 evaluated 10 combinations:                    |
|       14 was the best REACH, found in 1 combination.   |
|       14 was the FREQ value in that combination.       |
|       17 was the best FREQ in any size 3 combination.  |
|                                                        |
| The FREQ score for a combination is the count          |
| of the non-zero responses for that combination,        |
| summed over the reached cases.                         |
|                                                        |
| REACH.RESULTS file rrr has the 10                      |
| combinations with the highest reach scores.            |
| The items are ordered by their REACH contribution.     |
| Cumulative reach is shown.                             |
|                                                        |
| Time: .1 seconds.                                      |
----------------------------------------------------------
```

In this simple example ten groups are tested: v1-v2-v3, v1-v2-v4, etc. through v3-v4-v5. We are looking for the group that has at least one positive response for the largest number of cases. In other words, which group of three items REACH the most cases? The best combination uses variables v2, v3 and v4, which reach 14 of the 16 cases. The FREQ score for that combination is also 14.

Variable v1 is not included in the result because it adds nothing once v2 is selected. Variables v1, v2 and v3 have the highest FREQUENCY of response, but do not reach as many different cases as the group of v2-v3-v4. The twos in cases 9015 and 9016 on v5 are treated as ones.

The second command in Figure 8.6 is “LIST rrr”. Figure 8.6 shows the contents of File rrr as it is displayed by the LIST command. The items are ordered by the extent of their contribution to the total reach. The second line contains the actual amount of the contribution for each variable. The REACH.RESULTS file is discussed in detail later in this chapter.

Example 2 (Figure 8.7) shows case weighting. In Figure 8.7 variable WWW is not dropped because it is needed in the command. We are seeking the group of three variables that has the largest WEIGHTED number of reached cases. Cases 9015 and 9016 are the only cases with a case-weight of other than one, so they are the ones affected by case-weights in this example. The TURF output for this command shows that the CASE.WEIGHTS option was selected.

Figure 8.6 The REACH.RESULTS file

```
     ====== The LIST command ======

     LIST rrr $


     ====== The Listing produced ======

     FILE rrr

 TURF results ordered by REACH, from input file ddd using 5,3

size
 and              pct                item  item  item
 rank  reach  reached  freq  stats  .1    .2    .3

 3_1      14    87.50    14         v2    v3    v4
                            cum.r  7     11    14

 3_2      13    81.25    13         v2    v3    v5
                            cum.r  7     11    13

 3_3      13    81.25    13         v1    v3    v4
                            cum.r  6     10    13

 3_4      12    75.00    12         v1    v3    v5
                            cum.r  6     10    12

 3_5      12    75.00    12         v2    v4    v5
                            cum.r  7     10    12

 3_6      11    68.75    17         v2    v3    v1
                            cum.r  7     11    11

 3_7      11    68.75    11         v1    v4    v5
                            cum.r  6     9     11

 3_8      10    62.50    16         v2    v4    v1
                            cum.r  7     10    10

 3_9       9    56.25    15         v2    v5    v1
                            cum.r  7     9     9

 3_10      9    56.25     9         v3    v4    v5
                            cum.r  4     7     9
```

As you can see in Figure 8.7, the best group contains variables v2, v3 and v5, which reach 17 weighted cases. Variable v2 has the greatest reach with 7 cases. V3 adds 4 more cases. (Both v2 and v3 had unit weights.) Adding variable v4 would provide 3 more, but adding variable v5 increases the weighted reach count by 6, since each of

those two cases has a case-weight of three on variable WWW. Thus the final order for the best combination based on contribution is v2, v5, and v3 for a weighted reach of 17. The FREQ score for that combination is also also17.

Figure 8.7 A TURF Run Using Case Weighting

```
          TURF ddd [ DROP case.id ],
                SIZE 3,  REACH.RESULTS rrr,
                CASE.WEIGHTS www $


          LIST rrr [ CASES 1-2 ] $


 TURF results ordered by REACH, from input file ddd using 5,3

 size
 and              pct                item  item  item
 rank  reach  reached  freq  stats  .1    .2    .3

 3_1      17       85    17         v2    v5    v3
                             cum.r  7     13    17
```

Example 3 (Figure 8.8) shows response weighting. In Figures 8.4 and 8.7, the responses to the items were treated in a zero versus nonzero manner. Using RESPONSE.WEIGHTS causes the actual response values to contribute to the reach scores. In addition, using REACH.THRESHOLD 2 causes a case to be reached only when its reach score for a given group is 2 or more. Figure 8.8 also shows the first case in the REACH.RESULTS file.

Figure 8.8 Response Weighting and a Threshold of more than one

```
          TURF ddd [ DROP case.id www ],
                SIZE 3,  REACH.RESULTS rrr,
                RESPONSE.WEIGHTS,
                REACH.THRESHOLD 2 $


          LIST rrr [ CASES 1-2 ]$
 TURF results ordered by REACH, from input file ddd using 5,3


 size
 and              pct                item  item  item
 rank  reach  reached  freq  stats  .1    .2    .3

 3_1       8       50    16         v5    v1    v2
                             cum.r  2     2     8
```

Figure 8.9 Item Weighting and a Threshold of more than one

```
          /* create a file containing a weight value of 2 for item v3 */

          MAKE work1, VARS name:c weight;
          v3 2
          $

          TURF ddd [ DROP case.id www ],
                SIZE 3,  REACH.RESULTS rrr,
                ITEM.WEIGHTS work1,
                REACH.THRESHOLD 2 $

 Partial listing of the TURF analysis report

     -----------TURF analysis for file ddd completed-----------
     | OPTIONS: reach.threshold (2 was used)                   |
     |          item.weights                                   |
     |                                                         |
     | The ITEM.WEIGHTS were read from file work1.              |
     | One item used a weight that was supplied by that file. |


          LIST rrr $

 TURF results ordered by REACH, from input file ddd using 5,3

  size
  and              pct                 item  item  item
  rank  reach  reached  freq  stats  .1    .2    .3

  3_1      10     62.5    20         v3    v1    v2
                              cum.r  4     4     10
  3_2       6     37.5    12         v1    v2    v5
                              cum.r  -->   6     6
  3_3       6     37.5    12         v1    v2    v4
                              cum.r  -->   6     6
  3_4       4     25.0     8         v3    v2    v4
                              cum.r  4     4     4
  3_5       4     25.0     8         v3    v2    v5
                              cum.r  4     4     4
  3_6       4     25.0     8         v3    v1    v4
                              cum.r  4     4     4
  3_7       4     25.0     8         v3    v1    v5
                              cum.r  4     4     4
  3_8       4     25.0     8         v3    v4    v5
                              cum.r  4     4     4
  3_9       0      0.      0         v1    v4    v5
                              cum.r  -->   -->   0
  3_10      0      0.      0         v2    v4    v5
                              cum.r  -->   -->   0
```

The best group in the example in Figure 8.8 contains variables v1, v2 and v5, which reached 8 cases. Cases 1 through 6 were reached by a response of 1 to both v1 and v2; cases 15 and 16 were reached because of responses of 2 on v5. The FREQ score for that combination is 16.

The ordering of the items in the REACH file is:

```
V5      V1      V2
2       2       8
```

V5 reaches 2 cases all by itself because of the case weights. It is first because it adds more to the reach than either v1 or v2. Adding v1 by itself does not increase the reach. Adding v1 and v2 adds 6 more cases to the reach for a total reach of 8,

Figure 8.9 shows item weighting. Normally each item has a weight of one; each has the same contribution to a reach score. It is possible, however, to make some items worth more than other, possibly reflecting, for example, differences in costs of the items.

In this example, item v3 is weighted. This is conveyed in a P-STAT system file which has two variables. The first variable is a character variable which contains the variable names of any items which are to have a weight other than 1. The second variable in this file contains the weight to be applied to that item. In Figure 8.9 the MAKE command is used to create file WORK1 whose record defines a weight of 2 for item v3. This causes responses on v3 to be worth twice what they would otherwise be worth. As in Figure 8.6, a threshold of 2 is used.

The best group in this example is the combination of variables v1, v2 and v3, which reach 10 cases. Cases 1 through 6 achieve a reach score of 2 using items v1 and v2. Cases 8 through 11 have reach scores of 2 because of the item weight given to v3. V3 appears first as it is entered with no help from the other variables. The REACH does not increase until both v1 and v2 have been added.

```
 3_1       10      62.5     20          v3     v1     v2
                                  cum.r  4      4      10
```

The FREQ score for that combination is 20.

If the threshold is not reached when the first variable is entered the reach cannot be reported. To indicate that the reach cannot be reported the string "-->" is used.

```
3_2        6      37.5     12          v1     v2     v5
                                 cum.r  -->    6      6
```

This indicates that for this combination of 3 variables v1 by itself had no cases that could reach the threshold. The use of "-->" indicates that the threshold was not reached. In the combination that ranked 9th, the threshold was not reached even when all three variables, v1, v4 and v5 were entered. This is how that is represented.

```
3_9        0       0.       0          v1     v4     v5
                                 cum.r  -->    -->    0
```

8.6 Turf General Identifiers

Only two identifiers, TURF and either SIZE or STEP are required. All the others are optional

TURF xxx, This supplies the input filename. Except for an optional weight variable, all variables are treated as analysis items. The values on the analysis items should be zeros or positive numbers. A positive value signifies a "hit":. The maximum number of items is 210.

Cases with any missing or negative values on the analysis items are ignored. The SET.MISS.TO.ZEERO identifier, described below, sets missing analysis items to zeros. When case weighting is being used, any cases with a missing, negative or zero value on the weight variable are also ignored.

SIZE or STEP is required. They provide the size (or sizes) of the item combinations to be evaluated and indicate if the run should be stepped. SIZE, followed by one or more sizes, causes a separate, independent run for each given size. This evaluates all possible combinations for each size. One or more sizes can be done in one run. They are done in the order given.

```
SIZE 6
SIZE 4 TO 7 9,
SIZE 6 TO 3,
SIZE 4 6 8,

STEP 8 12,
STEP 1 TO 15,
```

STEP, followed by two or more sizes, processes the sizes in a stepwise manner. The sizes must ascend. The first step size is done normally, i.e., all possible combinations are evaluated, exactly like a SIZE run. Then, the items in its best combination for reach (or optionally for frequency) are fed into the second size, and so on.

A STEP run is usually much faster than a SIZE run because it is trying fewer combinations. However, it will probably not provide the absolutely best combination. STEP examples are provided later in this chapter. Note: STEP 8 12, does **NOT** mean that the first step does a size 8 and the second step does a size 12. It means that the first step does a regular size 8 run followed by a second step that adds 4 variables to **approximate** a size 12 run.

For SIZE 6 to 4, size 6 is done first. The final report shows the result for each size separately. The output files show the best results from the first size, then the second, and so forth.

Many (up to 40) sizes can be done in a run; each size must be from 1 to 60, and there should not be any repeated sizes in a run. Note...some sizes cannot be run in a reasonable amount of time. Consider 40 items. Depending on number of cases and on options:

```
1. Size  4 takes      91,390  iterations. Seconds.
2. Size  6 takes  3.8 million iterations. Minutes.
3. Size 10 takes  847 million iterations. An hour.
4. Size 15 takes   40 billion iterations. A day.
5. Size 20 takes  137 billion iterations. A week.
```

PUT with the COMBINATION function can be used to display the number of iterations that are required for a given number of items and sizes. Consider the following example of standalone PPL (P-STAT Programming Language):

```
DO  #j = 1, 20;
PUT #j (COMBINATIONS ( 40,#j));
ENDDO $
```

This produces 20 lines with the number of iterations for 40 items taken 1, 2, ... 20 at a time. STEP, which is discussed later in this chapter can be useful when the number of combinations is extremely large.

On PC/Windows the F2 key can be used to cause a TURF command to abandon the current size being processed. It will produce the report and the output files for the sizes already completed.

REACH.THRESHOLD 2, is optional; can be fractional. This permits the user to control what constitutes a successful “reach”. The default is one; if a case has a positive response on any of the items in a given combination, that case is added to the reach total for that set of items. Figure 8.9 illustrates a TURF command with a REACH.THRESHOLD.

Using REACH.THRESHOLD 3, for example, means a case needs a reach score of 3 or more to be reached on a given group. Having several responses increases a case’s reach score; weighting of either items or responses can also affect the reach score.

PROGRESS 5, is optional and controls how often the progress window or report line is updated. The default is 1, which means every million combinations. PROGRESS 0 turns it off.

SET.MISS.TO.ZERO, is optional. If it is used, missing analysis values in the input file are set to zeros. If missing values have been used instead of zeros in the input data, this is an easier way to do the transformation than

using the P-STAT programming language. This does not change the missing values in the input file. It does the transformation as the data are being read by the TURF command. Therefore it must be used every time the given P-STAT system file is used in a TURF command.

FULL.REPORT: the final report includes a summary about each size that was run. It uses 4 lines when there are 3 or fewer sizes, and uses a 2-line form when there are more than 3 sizes. Using the identifier FULL.REPORT causes the 4-line form (which contains more information) to be used in all situations.

8.7 Controlling Combination Makeup

There are three identifiers that can be used to control the variables which comprise each combination. These optional identifiers are FILTER FORCE, and STEP.

FILTER provides a limitation on the makeup of the combinations to be tried. It.is followed by the list of variables which make up a given combination and by 2 numbers which control how many of the variables must be included in the cluster.

```
FILTER list-of-vars  min  max,
```

Of the variables whose names (or ranges) follow FILTER, at least MIN of them and at most MAX of them should be in every combination that will be tried. The MIN value can be zero.

Up to 30 such FILTER phrases can be given. Combinations are used only if they fit within the filters in every one of the FILTER phrases. Each use of FILTER is followed by:

1. The names of the variables in the cluster. Ranges, like TOPPING.1 TO TOPPING.8, can be used.
2. The smallest number of those variables that are required. Can be zero. A combination must have AT LEAST that many of the variables in the cluster.
3. The largest number of those variables that may be used. A combination may have AT MOST that many of the variables in the cluster.

   All of the cluster could be used if the supplied number is equal to or larger then the size of the cluster. Therefore, using 999 is a vivid way of saying there is no upper limit for the cluster.

Figure 8.10 Specifying Clusters in TURF

```
TURF xxx, SIZE 8,
      FILTER  aaa         bbb to ddd  1 999,
      FILTER  eee to ggg  jjj to mmm  2 4,
      FILTER  yyy         zzz         0 1 $
```

In Figure 8.10 the only combinations that will be evaluated are those that have at least one variable from the first cluster, and at least two but no more than four variables from the second cluster, and no more than one variable from the third cluster. Any variable that is not in a cluster is treated like a cluster of 1 variable with a min of 0 and a max of 1.

FORCE vars, provides the names or ranges of items that should be in every combination. Suppose there are 30 items and size is 6; without force, 593,775 combinations are done, because we take 30 items 6 at a time. If 2 items are forced, only 20,475 combinations will be done because the run reduces to 28 items taken 4 at a time.

8.8 Weighting

CASE.WEIGHTS varname, is optional. The named variable will be used as a case-weight, and not as an analysis item. Figure 8.7 contains an illustration of CASE.WEIGHTS. The following contains a partial listing of the TURF report.

```
 ------------TURF analysis for file ddd completed-------------
| OPTIONS: case.weights (in variable www)                     |
|                                                             |
|        5 items were used in the analysis.                   |
|                                                             |
|       16 cases were read and used; the sum                  |
|          of the weights for these cases was 20.             |
|                                                             |
|       16 cases had at least one positive response.          |
|          The sum of the weights for these cases was 20,     |
|          making that the maximum possible reach.            |
|                                                             |
| SIZE   3 evaluated 10 combinations:                         |
|       17 was the best REACH, found in 1 combination.        |
|       17 was the FREQ value in that combination.            |
|       19 was the best FREQ in any size 3 combination.       |
```

RESPONSE.WEIGHTS, the default is to store the input data as zeros or ones, with one meaning a yes. This option leaves the input values intact; they should be in zero (no) or a positive value (not necessarily an integer) to show the INTENSITY of a yes. Figure 8.8 contains an illustration of RESPONSE.WEIGHTS. The following is a partial listing of the TURF report.

```
----------TURF analysis for file ddd completed-----------
| OPTIONS: response.weights                               |
|                                                         |
|        5 items were used in the analysis.               |
|                                                         |
|       16 cases were read and used.                      |
|       16 cases had at least one positive response,      |
|          making that the maximum possible reach.        |
|                                                         |
| SIZE   3 evaluated 10 combinations:                     |
|       14 was the best REACH, found in 1 combination.    |
|       14 was the FREQ value in that combination.        |
|       17 was the best FREQ in any size 3 combination.   |
```

ITEM.WEIGHTS filename, the default is treat all of the items the same, i.e., with weights of 1. When ITEM.WEIGHTS is used, it should be followed by the name of a P-STAT system file which itself has exactly 2 variables.

In each record, the first variable has the name of a item being used for the TURF analysis, the second is the weight to be used for that item. The first variable is therefore character, and the second is numeric. The file is not required to have a record for every item. In other words, some items can be given changed weights; others can be

left as is (i.e., still set to 1). The file can have names and weights for items not used in the current run; if so, they are ignored.

Figure 8.9 contains an illustration using ITEM.WEIGHTS. The following is a partial listing of the TURF report.

```
 -----------TURF analysis for file ddd completed-----------
| OPTIONS: item.weights                                    |
|                                                          |
| The ITEM.WEIGHTS were read from file work1.              |
| One item used a weight that was supplied by that file.   |
|        5 items were used in the analysis.                |
|                                                          |
|       16 cases were read and used.                       |
|       16 cases had at least one positive response,       |
|          making that the maximum possible reach.         |
|                                                          |
| SIZE   3 evaluated 10 combinations:                      |
|       14 was the best REACH, found in 1 combination.     |
|       18 was the FREQ value in that combination.         |
|       21 was the best FREQ in any size 3 combination.    |
```

8.9 THE RESULTS FILES

This section describes the REACH.RESULTS and FREQ.RESULTS files which are produced by the TURF command.

8.10 REACH.RESULTS File

The REACH.RESULTS file shows the combinations that had the best reach scores; the FREQ.RESULTS file shows the combinations that had the best freq scores. Most TURF runs will request a REACH.RESULTS file.

REACH.RESULTS rrr 300, This optional output P-STAT system file holds the combinations with the best REACH values. They are in descending order on REACH. Within ties on reach, the combinations are in descending order on FREQ (frequency). The item names in a combination are ordered by the reach contribution that each in turn adds.

The default is to write the 100 best combinations for each size. If an integer like 300 follows the file name, that many combinations are written for each size.

```
REACH.RESULTS reachout 1,
```

can be used to produce a results file with just the single best result included.

Each combination will take from 1 to 5 lines, as determined by the REACH.STATS identifier, described below. The default is two lines: one for the item names, the second for the cumulative reach for each successive item.

The names of the variables in the REACH.RESULTS file itself are listed below. Note, some (or all) of the initial 5 can be dropped by using the OMIT identifier.

1. SIZE.AND.RANK: the combination SIZE and RANK: the rank within size.
2. REACH: the reach value for the combination.
3. PCT.REACHED: the percent of usable cases that is reached by this combination. The usable cases are the number of cases with no invalid data. This includes cases with no positive responses whatsoever.
4. STATS: identifies the statistic printed on each line. The default is to print only the cumulative reach.

5. FREQ: the frequency value for the combination.

6. + ITEM.1, ITEM.2, ITEM.3, etc.: the names of the items that make up the combination. The item names are ordered by their contribution to the reach score. I.e., the name appearing under ITEM.1 is the “best” item in the combination.If sizes 6 and 8 are both being done, the file will have item.1 through item.8. The results for size 6 will have blanks for item.7 and item.8.

8.11 Ordering of the REACH Combinations

Suppose 6 items, AA, BB, CC, DD, EE and FF, make up a combination about to be written to the reach.results file. Before writing them, they are reordered so that the leftmost item is the one with the highest individual reach. The next item shown has, when paired with the leftmost item, the largest 2-item reach score, and so on.

The reordering is done in this manner.

1. FIRST PHASE: Find the smallest number of items (taken from the combination now being reordered) that gets a a nonzero reach, and determine the best item or group of items at that size. In the simple case, that will be just one item.

   When the reach threshold is 3 and there are no item or response weights, for example, we will not get a nonzero reach until groups of three are tried. We take the best of all possible groups of three.

2. SECOND PHASE: We now have one or more items as the beginning items for the reordered combination. If more remain, try each one with the beginning items, and add the item that adds the most to the reach count. Repeat this step until all items are added.

Suppose we are reordering a combination of 6 items, AA, BB, CC, DD, EE and FF. We start by trying each item separately. If one or more have a non-zero reach score, we place the item with the best reach in the first (leftmost) position. Of the remaining items, the one that adds the most to the first one is placed in the second position, and so forth. That is simple and obvious to show in the reach results file. For example:

```
DD     CC     EE     AA     FF     BB
43     62     71     78     81     82
```

However, suppose nonzero reaches did not begin until groups of three were tried. If BB, DD and EE were the best such group, they are shown in input order, with the reach score for the group under the last one. The symbol --> appears under the initial items in that group. For example:

```
BB     DD     EE     CC     FF     AA
-->    -->    37     55     61     64
```

This shows that no reaches occurred until groups of three items were tried, and that BB-DD-EE were the best of all 3-item groups. Since AA, CC and FF remained, BB-DD-EE-AA was tried, then BB-DD-EE-CC, then BB-DD-EE-FF. The output shows that CC added more than the others, bringing the reach for those four items up to 55, and so on.

The REACH.RESULTS file: TURF can be flummoxed by small, carefully constructed data sets. It should be noted that selecting the best two items in a stepwise manner is not quite the same as selecting the best two by trying all possible pairs. Suppose we have a file of 14 cases. Again, there are 4 items: AA, BB, CC and DD. We would like to find the 'best' two items.

```
AA reaches cases 1-10,
BB reaches cases 11-13,
CC reaches cases 1- 5 and 11-12,
DD reaches cases 6-10 and 13-14.
```

The stepwise approach selects AA and, having AA in hand, adds BB to get its best two items. They have a reach of 13. A non-stepwise approach tries all combinations of size 2 and would select CC and DD. They have a reach of 14. The TURF command uses a stepwise procedure ONLY in reordering the REACH.RESULTS (and FREQ.RESULTS); otherwise all runs are done trying every possible combination of the size being analyzed.

8.12 SHOW and OMIT

The default is for the REACH.RESULTS and FREQ.RESULTS files to have following five values before the items appear. These are:

1. SIZE.AND.RANK
2. REACH
3. PCT.REACHED
4. FREQ
5. STATS

Figure 8.11 TURF Ouput: Selecting Columns

```
TURF p0301,:
SIZES 4 TO 8,
SHOW REACH,
REACH.RESULTS work 2,
REACH.STATS NONE $
LIST work $
```

reach	item .1	item .2	item .3	item .4	item .5	item .6	item .7	item .8
212	VAR3	VAR4	VAR5	VAR23				
212	VAR3	VAR4	VAR5	VAR28				
213	VAR3	VAR4	VAR5	VAR23	VAR13			
213	VAR3	VAR4	VAR5	VAR23	VAR15			
213	VAR2	VAR3	VAR4	VAR5	VAR23	VAR13		
213	VAR2	VAR3	VAR4	VAR5	VAR23	VAR15		
213	VAR2	VAR3	VAR4	VAR5	VAR23	VAR13	VAR11	
213	VAR2	VAR3	VAR4	VAR5	VAR23	VAR15	VAR11	
213	VAR2	VAR3	VAR4	VAR5	VAR23	VAR13	VAR11	VAR17
213	VAR2	VAR3	VAR4	VAR5	VAR23	VAR13	VAR11	VAR10

Either a SHOW or an OMIT phrase can be used to specify which results are to be included in the file. The SHOW or OMIT phrase applies to both REACH and FREQ results files. In addition to the five specified above. SHOW may also specify:

1. NONE
2. PCT.OF.MAX.REACH

Figure 8.12 TURF Output: Selecting the Rows

```
TURF p0301 ,
SIZES 6 4,
OMIT FREQ,
REACH.RESULTS work 2,
REACH.STATS ALL $
```

```
    FILE work
size
and              pct          item    item    item    item    item    item
rank  reach  reached  stats    .1      .2      .3      .4      .5      .6

6_1     213   38.727          VAR2    VAR3    VAR4    VAR5    VAR23   VAR13
                       cum.r  187     199     208     210     212     213
                       add.r  187     12      9       2       2       1
                      %cum.r  34.00%  36.18%  37.82%  38.18%  38.55%  38.73%
                      %add.r  34.00%  2.18%   1.64%   0.36%   0.36%   0.18%
                       cum.f  187     373     551     729     815     933
                       add.f  187     186     178     178     86      118
                      unique  0       5       6       2       2       1

6_2     213   38.727          VAR2    VAR3    VAR4    VAR5    VAR23   VAR15
                       cum.r  187     199     208     210     212     213
                       add.r  187     12      9       2       2       1
                      %cum.r  34.00%  36.18%  37.82%  38.18%  38.55%  38.73%
                      %add.r  34.00%  2.18%   1.64%   0.36%   0.36%   0.18%
                       cum.f  187     373     551     729     815     931
                       add.f  187     186     178     178     86      116
                      unique  0       5       6       2       2       1

4_1     212   38.545          VAR3    VAR4    VAR5    VAR23
                       cum.r  186     208     210     212
                       add.r  186     22      2       2
                      %cum.r  33.82%  37.82%  38.18%  38.55%
                      %add.r  33.82%  4.00%   0.36%   0.36%
                       cum.f  186     364     542     628
                       add.f  186     178     178     86
                      unique  16      8       2       2

4_2     212   38.545          VAR3    VAR4    VAR5    VAR28
                       cum.r  186     208     210     212
                       add.r  186     22      2       2
                      %cum.r  33.82%  37.82%  38.18%  38.55%
                      %add.r  33.82%  4.00%   0.36%   0.36%
                       cum.f  186     364     542     602
                       add.f  186     178     178     60
                      unique  18      8       2       2
```

3. SIZE
4. RANK

This allows you to display SIZE and RANK in separate columns. OMIT ALL is the same as SHOW NONE.

8.13 REACH.STATS

When a reach.output file is written, the items within each combination are ordered by their reach contribution within the combination. This is always done. In addition, an extra line is written for each group which shows the cumulative reach as each item is added. That is the default, but it can be changed. As many as four extra lines are possible.

1. CUMULATIVE.REACH the increasing reach as each successive item is added. This is the default.
2. ADDITIONAL.REACH the additional reach provided by each successive item.
3. UNIQUE.REACH the reach that would be lost if the item were dropped.
4. CUM.REACH.PCT, the percent of the cases reached as each item is added.
5. ADD.REACH.PCT the additional percent of cases reached by each successive item.

REACH.STATS can also be followed by:

1. NONE by itself, no lines are written.
2. ALL by itself, 7 lines are written.

Figure 8.12 illustrates the effect on the REACH.RESULTS output file when the REACH.STATS identifier is used to include all possible formatting of the result statistics.

8.14 FREQ.RESULTS and FREQ.STATS:

```
FREQ.RESULTS  fff  500,
```

This optional output P-STAT system file holds the combinations with the best FREQ values. They are in descending order on FREQ. Within ties on FREQ, the rows are in descending order on REACH. The default is to write the 100 best combinations for each size. If an integer like 500 follows the file name, that many are written for each size. The item names in a combination are ordered by the frequency contribution that each in turn adds.

The FREQ.RESULTS file has the same variables as the REACH.RESULTS file. FREQ.STATS can be used to specify the details to be written out. It can be followed by:

1. CUMULATIVE.FREQ the increasing freq as each successive item is added. This is the default.
2. ADDITIONAL.FREQ the additional reach provided by each successive item.
3. NONE no additional lines are written
4. ALL both additional lines are written.

8.15 STEP

The STEP identifier can be used instead of the SIZE identifier. When SIZE is used the TURF command tries all combinations of each size specified. When STEP is used, the procedure selects the item that reaches the most cases and having that in hand, selects the next best item. STEP is followed by two or more sizes and processes the sizes in a stepwise manner. The sizes must ascend.

The first stepsize is done normally, i.e., all possible combinations are evaluated, exactly like a SIZE run. Then, the items in its best combination for reach (or optionally for frequency) are fed into the second size, and so on. A STEP run is usually much faster than a SIZE run because it is trying fewer combination. However, it will probably not provide the absolutely best combination.

Figure 8.13 STEP: The Greedy Method Command and Report

```
------ The Command ------

     TURF Step40,
          STEP 8 11,
          REACH.RESULTS st40reach 1,
          SHOW SIZE $


------ The Report ------

      ---------TURF analysis for file step40 completed---------
     | STEP:    2 sizes were done in a stepwise manner.       |
     |          The best reach combination in each step       |
     |          was carried into the second step.             |
     |                                                        |
     | OPTIONS: none                                          |
     |                                                        |
     |       40 items were used in the analysis.              |
     |      785 cases were read and used.                     |
     |      660 cases had at least one positive response.     |
     |                                                        |
     | SIZE   8 evaluated 76,904,685 combinations:            |
     |      652 was the best REACH, found in 8 combinations.  |
     |    3,273 was the best FREQ in those 8 combinations.    |
     |    3,454 was the best FREQ in any size 8 combination.  |
     |                                                        |
     | SIZE  11 evaluated 4,960 combinations:                 |
     |      659 was the best REACH, found in 2 combinations.  |
     |    4,288 was the best FREQ in those 2 combinations.    |
     |    4,312 was the best FREQ in any size 11 combination. |
     |                                                        |
     | The FREQ score for a combination is the count          |
     | of the non-zero responses for that combination,        |
     | summed over the reached cases.                         |
     |                                                        |
     | REACH.RESULTS file st40reach has the best combination  |
     | on reach from the first step, followed by the best     |
     | combination on reach from the second step.             |
     |                                                        |
     | The items are ordered by their REACH contribution.     |
     | Cumulative reach is shown.                             |
     |                                                        |
     | Time: 49.3 seconds.                                    |
     ----------------------------------------------------------
```

8.16 The Greedy Method

```
STEP 8 11
```

does a regular size 8 run and saves the results which are then forced into a size 11 run. This approximates a SIZE 11 run. Usually STEP is used when the number of combinations is so large that the run will take more than the available computer time. A look at the number of combinations when there are 40 variables that must be processed provides a vivid example.

The total number of combinations for 40 items taken 8 at a time is:

```
76,904,685
```

The total number of combinations for 40 items taken 11 at time is:

```
2,311,801,440
```

This is roughly 30 times more than (40,8) and will take approximately 30 times as long to process. Processing 40 items 8 at a time is done in minutes. Processing 40 items 11 at a time may take an hour or more. With 40 items using STEP 8 11 :

```
76,904,685   40 items 8 at a time
     4,960   32 items 3 at a time
```

This is a total of only 76,909,645 combinations.

The 8 items selected from the first step are forced into the second step. This leaves 32 items and only 3 more to be selected. The timing for this is very little more than the timing to process size 8. However, the more items that can be processed in the first step, the more likely it is that the approximation and the full solution are identical.

Figure 8.13 shows both the TURF command and the report that it produces. The command has 4 identifiers:

```
TURF Step40,                  the command and input file
STEP 8 11,                    the step sizes in ascending order
REACH.RESULTS st40reach 1,    the REACH.RESULTS filename and the number
                              of rows to be saved for each size
SHOW SIZE $                   limits the descriptive columns to SIZE.
                              More items now fit on a page.
```

Figure 8.14 REACH RESULTS: STEP Output

```
      FILE st40reach


TURF results ordered by REACH, from input file step40 showing 2 sizes.  PAGE 1

      item  item   item  item   item   item  item   item  item  item   item
size  .1    .2     .3    .4     .5     .6    .7     .8    .9    .10    .11

   8  var1  var15  var5  var19  var17  var6  var13  var2
      510   584    614   632    640    645   649    652

  11  var1  var15  var5  var19  var17  var6  var13  var2  var3  var18  var7
      510   584    614   632    640    645   649    652   655   658    659
```

STEP must have either a REACH.RESULTS file or a FREQ.RESULTS file. Providing a REACH.RESULTS output file indicates that the stepping should maximize the reach scores. Therefore, the items in the best combination on reach are forced into the next step.

Similarly, providing a FREQ.RESULTS output file indicates that the stepping should maximize the frequency scores. See Figure 8.17 for an example of the difference between a run which maximizes the reach and a run which maximizes the frequency.

Figure 8.15 Cascading STEP: Command and Partial Report

```
TURF steptest,
     STEP 1 TO 11,
     REACH.RESULTS reach1to11 1,
     SHOW size$


 --------TURF analysis for file steptest completed--------
 | STEP:    11 sizes were done in a stepwise manner.     |
 |          The best reach combination in each step      |
 |          was carried into the next step.              |
 |                                                       |
 | OPTIONS: none                                         |
 |                                                       |
 |       11 items were used in the analysis.             |
 |      785 cases were read and used.                    |
 |      632 cases had at least one positive response.    |
 |                                                       |
 | SIZE   1 evaluated 11 combinations:                   |
 |      510 was the best REACH, found in 1 combination.  |
 |                                                       |
 | SIZE   2 evaluated 10 combinations:                   |
 |      569 was the best REACH, found in 2 combinations. |
 |


   Additonal rows here for sizes 3 to 10


 | SIZE  11 evaluated 1 combination:                        |
 |      632 was the best REACH, found in 1 combination.     |
 |                                                          |
 | The FREQ score for a combination is the count            |
 | of the non-zero responses for that combination,          |
 | summed over the reached cases.                           |
 |                                                          |
 | REACH.RESULTS file reach1to11 has the best combination   |
 | on reach from each of the initial 10 steps, followed     |
 | by the best combination on reach from the final step.    |
 |                                                          |
 | The items are ordered by their REACH contribution.       |
 | Cumulative reach is shown.                               |
 |                                                          |
 | Time: less than 0.1 second.                              |
 ------------------------------------------------------------
```

STEP must also have at least 2 steps in ASCENDING order. More steps are possible. For example:

```
STEP 8 12 15,
```

If you have 50 items and are looking for the best 15, the total possible combinations is over 2 trillion. In general, the larger the initial step, the better, given that the running time is tolerable. STEP 9 12 could well yield a better result than STEP 8 12.

The timing of the runs is linear with the number of combinations. If you are using weights, runs will take longer. See the section later in the chapter on “PROCESSING SPEED” for more details.

Figure 8.14 shows the items that comprise the best combinations. As you can see the items from the first step have been forced into the second step in the order they appeared in the first step. The final report shows the result for each size separately. The output files show the best results from the first size, then the second, and so forth.

8.17 Cascading Step

```
STEP 1 to 11
```

is a cascading step. The first step is a size 1. The second step has the size 1 results forced into the procedure followed by a size 2. Step 3 has the results of the first 2 steps forced in, etc. This is very fast. With 50 variables, the first step is 50 combinations; the second step is 49 combinations, etc. This is a good solution if you want a result with the best single combination plus the best combination of the remaining 49 variables in turn. However, it will probably not give you the result with the greatest reach. The reach for the cascading step will never exceed the reach for SIZE. Figure 8.15 shows the command and part of the report for a file with 11 variables. Figure 8.16 is a partial listing of the REACH.RESULTS file.

If computing power is not the issue, the choice depends on what you are looking for in your data. There are many situations where some items must be part of the solution. The FORCE identifier is also available for this purpose.

Figure 8.16 Cascading STEP: Partial Results File

```
 TURF results ordered by REACH, from input file steptest showing 11 sizes.

       item  item  item  item  item  item  item  item  item  item   item
 size  .1    .2    .3    .4    .5    .6    .7    .8    .9    .10    .11

    1  var1
       510

    2  var1  var6
       510   569

Results from STEPS 3 to 9 here .........

   10  var1  var6  var5  var8  var3  var2  var4  var9  var7  var10
       510   569   594   607   614   620   625   628   630   631

   11  var1  var6  var5  var8  var3  var2  var4  var9  var7  var10  var11
       510   569   594   607   614   620   625   628   630   631    632
```

Figure 8.17 STEP: Reach and Freq Differences

```
TURF results ordered by REACH, from input file Steptest showing 2 sizes.

size
and   item  item   item  item  item   item  item   item   item  item  item
rank  .1    .2     .3    .4    .5     .6    .7     .8     .9    .10   .11

7_1   var5  var15  var2  var8  var19  var3  var13
      502   582    611   632   645    650   655

11_1  var5  var15  var2  var8  var19  var3  var13  var17  var1  var4  var16
      502   582    611   632   645    650   655    659    661   663   665


TURF results ordered by FREQ, from input file Steptest showing 2 sizes.

size
and   item  item  item  item  item  item   item   item   item  item  item
rank  .1    .2    .3    .4    .5    .6     .7     .8     .9    .10   .11

7_1   var1  var5  var2  var4  var3  var15  var13
      510   1012  1509  1977  2426  2846   3243

11_1  var1  var5  var2  var4  var3  var15  var13  var16  var6  var8  var7
      510   1012  1509  1977  2426  2846   3243   3611   3975  4324  4639
```

The commands which create the results in Figure 8.17 are:

```
TURF Steptest, STEP 7 11, REACH.RESULTS rrr 1,
     OMIT  REACH PCT.REACHED FREQ STATS $
TURF Steptest, STEP 7 11, FREQ.RESULTS  fff 1,
     OMIT  REACH PCT.REACHED FREQ STATS
```

Notice that is all these examples the REACH.RESULTS and FREQ.RESULTS identifiers have a second argument which controls the number of results that are saved for each size. One results row makes it easier to compare the rows. Asking for more than one does not provide much additional information as the same variables are forced into all the ranks within the step.

When SIZE is used the items are always reordered so that those that contribute the most are first in the output line. REORDER is not assumed with STEP and the items are in FORCE order in each output row.

```
TURF Step40, STEP 8 12, REACH.RESULTS St40Results, REORDER $
```

8.18 OTHER TURF OUTPUT FILES

The primary turf output files, REACH.RESULTS and FREQ.RESULTS are documented in the previous chapter. In addition there are four optional output files for the TURF command.

8.19 REACH.SUMMARY

REACH.SUMMARY qqq 200,

This optional output P-STAT system file tells you how many combinations had each of the reach values that were found. The default is to write the 100 best reach values for each size being processed. If an integer like 200 follows the file name, that many would be written for each size.

Figure 8.18 REACH.SUMMARY Partial Listing

```
  TURF P0301, REACH.SUMMARY work, SIZES 4 5 $

                    number
                        of    pct of  lowest  highest
 size  rank  reach  combos    combos    freq     freq

    4     1    212       2    0.0084     602      628
    4     2    211      21    0.0884     504      660
    4     3    210      72    0.3031     450      729
    4     4    209     151    0.6358     475      669
    4     5    208     105    0.4421     512      689
```

Each row in the reach.summary file contains:

1. SIZE: the combination size.
2. RANK: the rank within size.
3. REACH: a reach value.
4. NUMBER.OF.COMBOS: the number of combinations have that reach value.
5. PCT.OF.COMBOS: the percent of the combinations that have that reach value.
6. LOWEST.FREQ; the lowest freq value in the combinations at that reach value.
7. HIGHTEST.FREQ: the highest freq value in the combinations at that reach value.

FREQ.SUMMARY qqq 200, This output P-STAT system file tells you how many combinations had each of the freq values that were found. The default is to write the 100 best freq values for each size being processed. If an integer like 200 follows the file name, that many would be written for each size.

Each row in the freq.summary file contains:

1. SIZE: the combination size.
2. RANK: the rank within size.
3. FREQ: a freq value.
4. NUMBER.OF.COMBOS: the number of all combinations that have that freq value.
5. PERCENT.OF.COMBOS: the percent of the combinations that have that freq value.
6. LOWEST.REACH: the lowest REACH value in the combinations at that freq value.
7. HIGHEST.REACH: the highest REACH value in the combinations at that freq value.

8.20 FULL.OUTPUT

FULL.OUTPUT fff 200,

This optional output P-STAT system file has the results of combinations in the order that they were processed. The default is to write the results of the initial 5,000 combinations regardless of size. If an integer like 25,000

follows the file name, up to that many would be written. Such a large number should be used with caution, and only if really needed, because this can create a very large file.

Each row in the full.output file contains:

1. the REACH value for the combination.
2. the FREQ value for the combination.
3. the positions of the items that make up the combination. If the USE.NAMES identifier is also used, the names of the items will be used instead of the positions. Names, however, make the file larger.

Figure 8.19 TURF: Partial Listing of FULL OUTPUT File

```
    TURF P0301, FULL.OUTPUT 500, SIZES 4 5 $

             item  item  item  item  item
 reach  freq   .1    .2    .3    .4    .5

   210   729    1     2     3     4     -
   208   683    1     2     3     5     -
   208   679    1     2     3     6     -
   208   683    1     2     3     7     -
   208   679    1     2     3     8     -
   208   685    1     2     3     9     -
   208   689    1     2     3    10     -
   208   671    1     2     3    11     -
   209   669    1     2     3    12     -
   208   671    1     2     3    13     -
```

8.21 TURF Methods

This section describes the method for calculating the reach and freq scores for a case on the combination of items currently being processed. This type of scoring is done for each case on every one of the combinations which are being evaluated. The method for determining the importance of each item in a combination is done as the results files are written, and is described in the previous chapter.

8.22 General Considerations

INPUT FILE. A TURF run is done on an input file that contains some numeric items to be analyzed. The number of analysis items (NV) can be from 1 to 210, but will often be 20 or 30 or so. The file can contain thousands of cases.

CASE.WEIGHT. The file may also have a weight variable which provides a weight for each case. If not, a weight of one is assumed for each case.

SIZE. A combination size needs to be provided. This is the number of items to be examined in each pass over the data. Suppose NV is 30 and the size is 7. A pass over the data will be done for every different combination of 7 of the 30 items, causing 2,035,800 passes.

In each pass, the number of cases that have been reached by the current combination of items is counted. The goal is to identify the combination that reaches the largest number of the cases.

RESPONSE.WEIGHTS. The response values themselves can be used as weights that reflect the intensity of the response. Using this option causes the responses for each case to be placed in memory without any change.

When response.weights is not used, the responses for each case are stored in 0/1 form; 0 if the response was indeed zero, and 1 if the response was any value greater than zero. This takes much less memory space.

ITEM.WEIGHTS. The items themselves are assumed to be equally important. In other words, the default is for each of the NV items to have a weight of one in the reach scoring. Different weights can be provided for some or all of the items. These are read from a file associated with the ITEM.WEIGHTS option.

REACH.THRESHOLD. Finally, the reach threshold, which defaults to one, can be changed to 3, for example, by saying REACH.THRESHOLD 3. The threshold can be fractional, like 3.5.

8.23 How the Reach Scoring is Done

A case is reached when its reach score equals or exceeds the reach threshold. Suppose we are scoring a case on a combination that consists of items V2, V5, V11 and V17. Remember, the responses are stored internally as 0 or 1 except when the RESPONSE.WEIGHTS option is in use. The reach score for a given case is:

```
V2 response times V2's item weight, plus
V5 response times V5's item weight, plus
V11 response times V11's item weight, plus
V17 response times V17's item weight.
```

If that score equals or exceeds the reach threshold, the case's case weight is added to the number of cases that have been reached for that combination. One is used when there is no CASE.WEIGHT variable. When responses and items are unweighted and the threshold is one, a case will have been reached when it has a positive response on any item in the combination.

8.24 How the Freq Scoring is Done

The FREQ score for a combination is the sum of the freq scores of the cases that were reached. If the case.weight option is not in use, consider each case to have a weight of one. A case's freq score depends on the options in use.

1. No use of response.weights or item.weights: Count the positive responses within a case on the variables in the combination. Multiply that count by the case's weight.
2. Response weights are used, but not item weights: Sum the positive responses within a case on the variables in the combination. Multiply that sum by the case's weight.
3. Item weights are used, but not response weights: Sum the item weights for those variables in the combination that have a positive value. Multiply that sum by the case's weight.
4. Response weights and item weights are in use: Sum the item weight times the response value for those variables in the combination that have a positive value. Multiply that sum by the case's weight.

8.25 PROCESSING SPEED

This section describes the factors that determine how long a TURF command will take to run.

8.26 Processing Speed: Cases and Combinations

There are three components whose effect on the speed of a TURF run is more or less linear:

1. more cases. Twice the cases for the same analysis will take twice the time.
2. more combinations to be tested. 30 items taken 6 at a time (i.e., 30,6) has 593,775 combinations, 30,7 has 2,035,800. That is 3.4 times as many combinations. Therefore, it will take 3.4 times longer to run.

3. CPU speed. Since the data and the results are held within memory during a run, going from a 800 mHz chip to a 2.4 GHz chip should be about 3 times faster.

8.27 Processing Speed: Effects of Various Options

TURF speed is also greatly affected by the options chosen. The fastest run is one that uses none of the options. For example:

```
TURF INFILE, SIZE 6, REACH.RESULTS OUTFILE$
```

If this takes one second, how much longer do the various options take?

1. 10 seconds if adding just CASE WEIGHTING.
2. 28 seconds if using response weighting, item weighting or reach threshold (with or without case weighting).

8.28 Processing Speed: Output Files.

The various output files take very little extra time. A 29,7 run for 500 cases with no output files took 2.5 seconds. Adding the default sized (best 100) REACH.RESULTS or FREQ.RESULTS files made the runtime 2.7 seconds; doing both took 2.8 seconds. Asking for the 20,000 best REACH.RESULTS results instead of the default best 100 took 3.1 seconds instead of 2.7.

The REACH.SUMMARY and FREQ.SUMMARY output files take a little more time than the REACH.RESI:TS and FREQ.RESI:TS files. The FULL.OUTPUT file takes almost no time since there is no sort management involved.

8.29 Limitations on Run Size

ITEMS: Using lots of items can be done, but only with sensible combination sizes. For example, one might look at 4 items out of 200 (64.7 million passes), but even 6 out of 200 would become excessive (82.4 billion passes).

COMBINATION SIZE: The maximum combination size is 60 items. One could look at 16 out of 24, for example, or even 40 out of 45. However, 60 out of 210 is so large that it will never finish. This P-STAT command:

```
PUT( COMBINATIONS(50,7))$
```

returns the number of combinations that 7 out of 50 would require, for example, and may be useful in estimating the time of a prospective run.

How large a run can be done depends on the number of cases, the number of combinations, the options used, and the speed of the PC itself. If you are considering a large run, 10 billion combinations or more, you might try smaller run first, get it's time, and use the ratio of the combinations to estimate the time needed for the larger run.

A run of 10 combinations out of 50 items is 10.27 billion combinations and would undoubtedly take quite a few hours, but it is possible. The progress meter ticks every million combinations, so you can easily tell how long a run will take once it starts.

8.30 Limitations on Memory Size

This text describes some of the factors to consider when planning a large TURF run/

1. ITEMS: Using lots of items can be done, but only with sensible combination sizes. For example, one might look at 4 items out of 200 (64.7 million passes), but even 6 out of 200 would become excessive (82.4 billion passes).
2. COMBINATION SIZE: the maximum combination size is 60 items. One could look at 16 out of 24, for example, or even 40 out of 45. However, 60 out of 210 is so large that it will never finish.

This P-STAT command,

```
PUT( COMBINATIONS(50,7))$
```

would return the number of combinations that 7 out of 50 would require, for example, and may be useful in estimating the time of a prospective run.

HOW LARGE A RUN CAN BE DONE: as described above, it depends on the number of cases, the number of combinations, the options used, and the speed of the PC itself.

If you are considering a large run, 10 billion combinations or more, you might try a smaller run first, get it's time, and use the ratio of the combinations to estimate the time needed for the larger run.

IS 10 OUT OF 50 POSSIBLE: this is 10.27 billion combinations and would undoubtedly take quite a few hours, but is possible. The progress meter ticks every million combinations, so you can easily tell how long a run will take once it starts. Useful results for the impossible combinations can be obtained by using the STEP identifier which provides an approximation of the exact amount by doing the analysis in stages: For example:

```
STEP 7 10,
```

The first pass does a standard SIZE 7 run. The 7 variables that produce the best result are then forced into a second step. The relevant combinations are:

```
COMBINATONS  50  10    10,272,278,170
COMBINATIONS 50   7        99,884,400
COMBINATIONS 43   3            12,341
```

It is obvious that combinations of 50 variables 10 at a time is going to take a long time. However, the more variables that can be included in the first step, the more likely it is that the approximation is the exact result. If instead of doing 7 variables the first step and then the remaining 3, we do “STEP 6 10”, we get these figures:

```
COMBINATIONS 50   6       15,890,700
COMBINATIONS 50   4          135,751
```

This run will take less computer time but the results may not be as accurate. The moral is that “you gets what you pays for”.

8.31 Limitations on Memory Size

MEMORY CONSTRAINTS FOR INPUT: the input data must fit in memory. (We really don't want to read the data afresh from disk for each of 50 million different combinations.) In most situations memory should not be a problem because the data is usually stored very compactly. IF more than 50% of the available memory is used, the final report contains information on how much of the input data area was used.

MEMORY CONSTRAINTS FOR OUTPUT: the output files (except for the FULL.OUTPUT file) are collected in memory in sort order as the run progresses. The default sizes cause no problems.

If one asks for 20,000 results in the REACH.RESULTS or FREQ.RESULTS files, the run will be slightly slower but the file should fit.

SUMMARY

TURF

```
TURF Myfile,  SIZE 6,  REACH.RESULTS Myreach $
```

TURF stands for Total Unduplicated Reach and Frequency. The input is typically the results from a survey of user preferences such as: “Do you listen to?” or “Do you like?” followed by a list of radio or food items. The goal might be to determine which 4 of 20 provide the most listener/viewer contact or what new menu item will broaden our client base

. This is the “reach”, i.e. the number of individual cases with at least 1 positive response on a combination. The frequency is the total number of positive responses. It will never be less than the reach.

Required:

TURF fn

provides the name of the input file. PPL can be done as it is read. All variables are used in the analysis except an optional WEIGHT variable.

SIZE nn

provides the size of the combination to be used. Can be 1 to 60.

STEP nn nn

STEP may be used instead of SIZE to produce a stepwise result. STEP is described in detail in the next chapter.

Optional Identifiers: REACH and FREQ Output Files

REACH.RESULTS fn nn (optional)

provides the filename for a P-STAT system file with the REACH.RESULTS output. The 100 best combinations on REACH for each size are included unless an alternate number is provided.

REACH.STATS NONE / ALL
CUMULATIVE.REACH / ADDITIONAL.REACH
CUM.REACH.PCT / ADD.REACH.PCT
UNIQUE.REACH
CUMULATIVE.FREQ / ADDITIONAL.FREQ

If REACH.STATS is not specified, the default is CUMULATIVE.REACH, the cumulative contribution as each new item is added to the mix.

REACH.THRESHOLD nn

controls the number of positive responses required before a case is included in the reach. The default is 1.

FREQ.RESULTS fn nn (optional)

provides the name for a P-STAT system file of the frequency output. The 100 best combinations on FREQ.RESULTS are included for each size in the analysis unless an alternate number is provided.

FREQ.STATS ALL / NONE
CUMULATIVE.FREQ / ADDITIONAL.FREQ

Cumulative provides the increasing total frequency as each successive item is added. This is the default. Separate provides the additional frequency for each item. One or both may be requested.

Optional Identifiers: Weighting

CASE.WEIGHTS vn

provides the name for the optional case weight variable.

FILTER vlist nn nn

FILTER provides a limitation on the makeup of the combinations to be tried. It.is followed by the list of variables which make up a given combination and by 2 numbers which control how many of the variables must be included in the cluster.

```
FILTER list-of-vars  min  max,
```

Of the variables whose names (or ranges) follow FILTER, at least MIN of them and at most MAX of them should be in every combination that will be tried. The MIN value can be zero.

ITEM.WEIGHTS fn

provides the name of an optional input file which contains weights for the items. This file has just two variables. The first is a character variable with the name of the item. The second is numeric (can be fractional) with the weight to be used for that item.

RESPONSE.WEIGHTS

requests that the data itself be treated as a weight. The default is to use a value of 1 for any item which is greater than zero.

Provides the name for a weight variable.

Optional Identifiers:

FORCE vn vn TO vn

provides the name of variables to be forced into every combination

FULL.REPORT

The final report includes a summary about each size that was run. It uses 4 lines when there are 3 or less sizes, and uses a 2-line form when there are more than 3 sizes. Using FULL.REPORT causes the 4-line form (which contains more information) to be used in all situations.

OMIT SIZE & RANK / REACH
PCT.REACHED / FREQ / STATS

The default in the REACH.RESULTS and the FREQ.RESULTS to have all 5 of these columns included. Any or all of them may be omitted from the files. In addition SIZE and RANK can be individually specified and ALL may be used to produce files which only contain the selected items.

PROGRESS nn

specifies how often the progress window is updated. The default is 1 (one million).

SET.MISS.TO.ZERO

sets missing data in the input to zero. Any case with missing data is dropped from the analysis. This saves the need to use the programming language to do the recodes.

SHOW **SIZE & RANK / REACH**
PCT.REACHED / FREQ / STATS

These five are usually included in the REACH.RESULTS and FREQ.RESULTS files. Any or all of these may be selected. In addition SIZE and RANK may be used individually. SHOW NONE is the same as OMIT ALL.

TEMPLATE **fn**

The TEMPLATE output file is used as input to the TURF.SCORES command. This command is used when you wish to learn more about the people who were reached. T URF.SCORES can be run without any input from the TURF command but use of the T EMPLATE file is the most convenient method. TURF.SCORES is documented in the manual “P-STAT: Basic Statistics”.

9 Simple Commands and Keywords

The simple commands and keywords covered in this chapter are of 3 different types:

1. Simple commands to control the environment and facilitate printing.
2. Keywords that can stand by themselves and/or be used as an identifier to another command.
3. Programming language that is not connected to a specific file.

Most commands have identifiers which modify and control command input and output. In addition there are some identifiers that are not unique to a particular command but can be used with any command. A good example of a "general identifier" is "IDEN" which provides a list of all the identifiers associated with a given command.

```
TEXTFILE.IN, IDEN, ..... $           produces:

 The TEXTFILE.IN command has these identifiers.....

 character                ignore.strings     skip
 decimal.comma            label.changes      string.boundary
 delimiter                labels             textfile.in
 extended.labels          numeric
 honor.strings            read
```

When a general identifier is used within a command it applies only to that one command. When a general identifier is used as a standalone command it influences all the commands that follow. PR is an example of an identifier that can act either as a standalone command or as an identifier in another command. The PR identifier when used in a specific command tells P-STAT where to write the output of that command.

```
LIST MyFile, PR 'listing.txt' $
```

When the PR identifier is used by itself as in:

```
PR 'listing.txt' $
```

it causes all the commands that follow to be printed to that file until another PR setting is encountered.

This chapter is organized not by the type of command or identifier but by the more general area of application. Commands and identifiers that are associated with P-STAT system files are documented in the next chapter, "AUTOSAVE: P-STAT System Files".

9.1 COMMAND OUTPUT

Command output comes in 2 basic forms. The first output form is a new or revised P-STAT system file. The second form is any output other than a P-STAT system file. Many commands produce both a system file and a report about the operation that created that system file. Other commands do not create P-STAT system files but are designed to print reports in the forms of listings or tables.. This is the output we are concerned about in this chapter. This output may be a display on the screen, a file to be stored on disk or sent to a printer. The destination helps determine attributes such as output width, number of lines and print destination.

If you are running P-STAT in interactive mode, reports and listings will automatically be displayed on your terminal. Output written to the terminal has an extra attribute, the number of lines on the screen. The default value for the "SCREEN" is 24 lines and its job is to give the user time to read the contents before it scrolls and displays the next 24 lines. The SCREEN command is used to change the screen "holding" value.

SCREEN 30 $

When the screen is full, P-STAT pauses so that you can have a chance to read the contents. At the bottom of the screen you will see the message:

```
holding .
```

When you are ready you can either press the return/enter key for more printout or enter "Q" to stop the printout. There 3 levels of Quit;

1. Q1 or just Q stops the current section of printout.
2. Q2 stops the current command
3. Q3 stops the current command as well as any expanded macros, transfer files, subfile loops or editor generated commands.

If you are running a batch job the reports and listings will automatically be written to the output destination. In either batch or interactive mode you can direct your output to one or more specific "print" files, each with its own attributes. Output that is going to a "print" file is automatically formatted to fit a page that is 59 lines long and 132 characters wide. Output that is going to the terminal is formatted to fit within an area that is 80 columns wide and 24 lines long. There is an interaction between SCREEN and LINES. The only use of SCREEN is to prevent the output from scrolling away before you can read it. The LINES setting determines when TITLES, i.e., the start of a printed page, is displayed.

9.2 Print Files

The PR identifier is used to supply the name of a file where P-STAT should write command output. If it is used as a command, all output that follows will be written to that output file.

```
PR 'c:\workdisk\august.txt' $
```

If PR is used as a local identifier, only the output from the current command will be written to that output file.

```
LIST myfile, PR 'c:\workdisk\august.txt' $
```

There are four attributes which control the appearance of the printed output. They are:

1. OW or OUTPUT.WIDTH — the output width of the page. The default OW for the screen is 80 characters. The default OW for other printout is 132.
2. LINES — the number of lines on the page. The default settings are: 22 for a terminal, 59 for printout.
3. PAGE.CHARACTER — the page change character. This is usually set to an ASCII 12 but can be changed to a blank, a 1 or any special character required by the printer being used. PAGE.CHARACTER is not relevant in screen output.
4. VBAR — supplies an alternate character for the vertical bar.

OW and LINES are general identifiers, that means that they can be used as an identifier in any command. If the printout is going to the screen and the OW setting is too wide, the printout will be wrapped and difficult to read. If the LINES setting is longer than the height of the terminal, the SCREEN setting is all that will prevent the listing from scrolling away.

An easy way to control the printout is to define one or more print destinations with appropriate attributes for the commands you are using. The commands DEFAULT.SETTINGS and PRINTER.SETTINGS are used to provide the appropriate settings.

```
DEFAULT.SETTINGS,  OW 140,  LINES 60,   PAGE.CHARACTER 12 $
PRINTER.SETTINGS 'prt.file', OW 200, LINES 56, VBAR 'I' $
```

In this example the print file named “prt.file” has a lines setting of 56, an output width of 200 and takes the value for the page change character from the DEFAULT.SETTINGS.

(PRINT.PARAMETERS and PP are both aliases for PRINTER.SETTINGS)

9.3 TABFILE.OUT / TEXTFILE.OUT

```
TABFILE.OUT  pstatfile,  write 'xxx',  no labels  $
TEXTFILE.OUT pstatfile,  file  'xxx',  delimiter comma   $
```

The TABFILE.OUT command reads a P-STAT system file and writes a delimited ascii file in a format that EXCEL, among others, can read. The name TEXTFILE.OUT can also be used to access this command. In a delimited file, each value is followed by a character which signals the end of the value; i.e., the character DELIMITS the value. As the command name suggests, the default is to use a horizontal tab as the delimiter character. The DELIMITER (or DEL) identifier can be used to cause the output file to be delimited by a different character. COMMA and BLANK are other commonly used delimiters.

TABFILE.OUT (or TEXTFILE.OUT) is followed by the name of the P-STAT file whose contents (after optional PPL) are to be written as an ascii file in delimited form. All missing values are written as NULL values in the output file, i.e., nothing but the delimiter is written. E format is used when needed to provide an accurate representation of numeric values.

When the delimiter is not a blank, a non-missing character value is copied through the rightmost non-blank. Any CR, LF or delimiter characters in the value itself are changed to blanks; if this were not done, the file would be unreadable. A totally blank character value is represented by one blank and then a delimiter in the output file.

When the delimiter is a blank, a character value is copied from its leftmost nonblank through its rightmost nonblank. CR, LF and blanks in the value are changed to periods. A blank value is written as a period and the delimiter (a blank).

WRITE or FILE is followed by the name of the delimited ascii file being written. The name is in quotes, and can include a path. It may be wise to add an extension of .txt . The assumption is that the variable names in the P-STAT file will be written as the first record in the output file unless “NO LABELS” is specified. The P-STAT PPL rename function can be used to shorten the labels when the software that will read the file only supports, for example, 8 character labels.

The default is to write the full labels of the variables to the output file. These can be as much as 64 characters each. SHORT causes just the tag section to be shown, if possible. If a variable has no tag, the initial 16 characters are printed. SHORT.TAGS and TAGS are synonyms. SHORT.16, causes the initial 16 characters to be shown. Tags are ignored in this option. SHORT.OLD, uses the initial 16, but without colons, and with underscores changed to dots. This makes it a valid name in the previous versions of P-STAT.

These SHORT options could cause two or more variables to have the same 16-character representation. When this occurs, the last few characters of each is changed to nn, where nn is the position of each variable in the file

```
CR     LF     NL     CRLF
```

These are ways of ending records. CR is ascii 13, or CRLF. LF is ascii 10 (as is NL), and CRLF is 13 and 10. A record, whatever its size, holds all of the values for a single case. CRLF is the default when P-STAT is running on a PC/Windows system, otherwise the default is LF. One of these can be supplied to override the default.

```
EOF
```

requests that an end-of-file character (ascii 26) be written at the ending of the file. The default is not to write an EOF.

```
DELIMITER  ‘/’
```

provides a value delimiter character to be used instead of the default horizontal tab (9). DEL can be used instead of DELIMITER. The argument can be a single character in quotes, or the ascii value (0-255) of the character, or the names BLANK, COMMA or TAB. For example, DEL ' ' and DEL 32 and DEL BLANK are equivalent.

```
DECIMAL.COMMA,
```

This causes the decimal separator within numbers to be a comma instead of the default period. Thus, 123456,78 would be written rather than 123456.78 .

When LABELS are written, the initial record will contain the names (ie, labels) of the variables, and the subsequent records will contain the data. They are separated by delimiters, with an end-of-record (eor) after the last label. The eor can be CR, LF or CRLF.

When NO LABELS is used, the file contains only data records. A data record contains the values for one case. In a data record, each case's values are delimiter-separated, and each case ends with an end-of-record. An end-of-file is written to end the file only when identifier EOF is used in the command.

9.4 SYSTEM VARIABLES

System variables are set by P-STAT and P-STAT commands to pass information between commands and to return information from commands in a form that can be tested and used. System variables differ in form from other variables because they both begin and end with a decimal point. The system variables which contain information about page numbers, time and dates are covered in Chapter 5, “Labels and Titles”.

Many of the system variables are used in the programming language (PPL) to control the process of modifying a file. Examples of these variables are discussed in detail in “P-STAT: A Guide to the Programming Language”. They are here to provide a glimpse of the power that is available for handling your data. One group of system variables provides information about the processing of a file.

.FILE**.** refers to the current P-STAT system file. It is used in PPL to locate either the first or last case in a file. The following example produces a 1 line report showing the number of cases in the file which have the same value on variable age as does the first case in the file. The PROCESS command is similar to the MODIFY command except that PROCESS never creates an output file. PROCESS uses PPL to garner information from the file and store it in scratch variables for use in other commands.

```
PROCESS test [ IF FIRST ( .FILE. )
               GEN  ##counter = Age, GEN ##number = 0;

               IF Age NE ##counter DELETE;
               INCREASE  ##number; ] $

PUT ’File test contains ’ ##number ’ rows with a value of ’
        ##counter ’ on variable Age.’ $
```

Produces the following sentence:

```
File test contains 30 rows with a value of 27 on variable Age.
```

This small example illustrates the use of a system variable (**.**FILE**.**), 2 scratch variables ##number and ##counter, and standalone PPL with the PUT statement. The rest of this chapter will provide brief descriptions of other system variables. Most of them are discussed in full in the programming manual cited above and they all have entries in the master index which not only tells you the manual, chapter and page but will also open the appropriate manual when you click on its entry. The manuals are all available without charge at the PSTAT.COM web site.

9.5 Testing variables

There are many system variables that are useful for testing and recoding the values in a file. These system variables refer to the value of a given variable.

1. .G. IF AGE .G. if variable age good, i.e., not missing
2. .M. IF AGE .M. if variable age missing
3. .M1. .M2. .M3. IF AGE missing type 1, type 2, or type 3

The following list of system variables refer to the variables in a case or to a case within the file.

1. .HERE. The count of cases processed as a given PPL statement is invoked.
2. .CHARACTER.The list of character variables in the file.
3. .N. The current case number after case (row) selection.
4. .NUMERIC. The list of numeric variables in the file.
5. .NEW. All variables created by PPL in the current command.
6. .NV. The current number of variables in the file.
7. .ON. The remaining variables in the current case.
8. .OTHERS. All variables other than those explicitly referenced in the PPL.
9. .PUT. The number of "PUT" statements for the current case.
10. .USED. The number of cases used after all the PPL is processed.

There are system variables to hold specific values.

1. .E. The system value for e, the base of natural logs. It equals 2.718281828.
2. .PI. The system value for pi. It equals 3.141592654.
3. .DATE. The current date.
4. .TIME. The current time.

Most of these are designed for the user with a complex problem or very "messy" data. The most generally useful system variables are illustrated in the following examples:

```
SURVEY Myfile [ DROP .character.], .....;
SURVEY Myfile [ KEEP .numeric.  ], .....;
LIST   Myfile [ KEEP First.Name, Last.Name, .others. ] $
```

KEEP and DROP are used to select the variables to be used in the current command. The first 2 examples have the same effect: a file which contains only numeric variables. The final example is used to rearrange the order of the variables in the file.

There are several other system variables for date and time which are documented in "P-STAT:A Guide to the P-STAT Programming Language (PPL)". There are also a few system variables which are useful if you are using P-STAT macros or the subfile capabilities. These are all covered in the relevant P-STAT manuals.

System variables are also used by several P-STAT commands and PPL instructions to store results. The CHECK command looks at a P-STAT system file and checks to see that it is a valid system file. If for some reason you lose power during a P-STAT job, it is good to use CHECK on the active files to make sure that they have not been corrupted. The results are provided in a brief report. The results are also stored in system variables. In an interactive run, the report is sufficient. In a batch run it is useful to have the results stored as variables so that they can be tested and the run aborted if there are problems with the P-STAT system file. The 3 system variables for CHECK are .CHECKFILE., .CHECKVARS., and .CHECKCASES.

The COLLECT PPL instruction also uses system variables to store its results. They are:

1. .COLLECTSIZE. The number of cases in the most recent collect
2. .COLLECTMIN. The size of the smallest collected group so far
3. .COLLECTMAX. The size of the largest collected group so far
4. .COLLECTIONS. The total number of collects that occurred
5. .COLLECTSUM. The number of cases that have been collected.

Macros and subfiles are areas which use system variables for communication. These are fully covered in the PPL (programming language) manual.

9.6 More About Dates

FORMAT.DATE is a date/time function that provides considerable flexibility in formatting date-time values. It has two arguments: the character value to be formatted and the format to be used. The first argument is a date value. A date value is not a special data type but it is a character value that can be interpreted without ambiguity. For example "May 18 2008" is readily interpreted, but if the 3 fields are numeric (05092008) the date must first be processed by one of the many date conversion routines. There are 6 functions for the possible combinations of the 3 fields.

```
DAY.MONTH.YEAR        DAY.YEAR.MONTH
MONTH.DAY.YEAR        MONTH.YEAR.DAY
YEAR.DAY.MONTH        YEAR.MONTH.DAY
```

Use of dates can be complex. Once the date is in a character variable it can be displayed in a variety of formats. The following example stores the numeric data in a permanent character scratch variable named ##D1 and prints it using the default date format.

```
GEN ##D1:c = DAY.MONTH.YEAR ( 13042012 ) $
PUT ##D1 $
```

produces:

```
Fri April 13, 2012
```

To get the current date printed substitute .DATE. for ##D1 in the PUT statement.

If you prefer a different format you can use FORMAT.DATE to provide the template you prefer. Here ##FMTx is replaced by one of 4 different date templates.

```
GEN ##this.date:c = FORMAT.DATE ( ##D1, ##FMTx ) $
```

Here are four different date templates and the resulting character string stored in variable "this.date".

```
##FMT1:c = 'Month-dd-yyyy'       April-13-2012
##FMT2.c = 'mon dd yy'           april 13 12
##FMT3.c = 'dd/n.month/yy'       13/05/12
##FMT4.c = 'Dow Mon dd yyyy'     Fri April 13, 2012
```

With ##FMT4 we again have the default format.

There are many variations possible. They are fully documented in the help file and the PPL manual. Any one project will seldom have more than one or two preferred arrangements for dates. These can be defined at the start of the job in scratch variables and then used in PPL throughout the run. The formats can also contain time information with hours, minutes, seconds and representations for AM and PM.

9.7 Arrays, Vectors and Standalone PPL

The basic array in PPL is "V" which is the array of variable positions. Sometimes it is easier to point to a variable by position than by name. This is even more useful with variable names which can be as much as 64 characters long. The position in V is the current position after any other PPL has been processed.

```
LIST Myfile [ KEEP V(2), V(5) to V(11) ] $
```

In the example below the variables that are kept are from the numeric variables remaining after the DROP has been accomplished.

```
LIST Myfile [ DROP .character.; KEEP V(2), v(5) to V(11) ] $
```

A second vector is known as the permanent vector or “P”, This vector is as long as the maximum number of variables allowed in a single system file. In Whopper 2 this is 6,000 variables. The elements of the P vector are initialized to missing when the P-STAT run begins and remain missing until you change them. The P vector can only hold numbers but it provides an easy way to pass a large number of values across cases or between commands. Since the P vector exists in memory rather than in a P-STAT system file it can be referenced even when there is no active P-STAT system file available. The following example stores the totals across all the numeric variables in the input file and then displays the first 2 values.

```
PROCESS Myfile [ DROP .CHARACTER.;
     DO #j = 1, .NV.;
     IF FIRST ( .FILE. ) SET P(#j) = 0;
     IF V(#j) .GOOD. INCREASE P(#j);
     ENDDO; ] $
PUT P(1) P(2) $
```

An even more powerful tool in the PPL arsenal is the ARRAY. Like scratch variables, arrays are defined during a run and used in PPL statements. An array can have up to 7 dimensions, and can be character or numeric. Array names are just two characters, the second character is the same as the first, like XX or cc or Zz. Case doesn’t matter. Thus there can be up to 26 active arrays. Arrays and the P vector are documented in full in the PPL Manual.

Figure 9.1 is a more complex example of standalone PPL. It is here to provide a glimpse into the power of PPL and its ability to handle complex problems. It is an example of the use of both temporary and permanent scratch variables; of arrays and of standalone PPL. Standalone PPL can only be used when the computations do not require input from a data file.

Figure 1.1 Arrays in PPL

```
DEFINE.ARRAY xx(3,5) to 0 $
GEN ##n = 0 $

DO #j = 1,3;
DO #k = 1,5;
INCREASE ##n;
SET xx(#j, #k ) = ##n;
ENDDO;
ENDDO $

PUT 'XX(2,3) = ' xx(2,3) $

XX(2,3) = 8
```

For the user of the P-STAT TURF command the following is a very useful piece of standalone PPL.

```
GEN ##c = COMBINATIONS ( 27, 8 ) $
PUT '27 items 8 at a time is ' ##c ' passes through the data.'$
```

```
27 items 8 at a time is 2220075 passes through the data.
```

When running a TURF analysis this is the number of iterations that it takes to do the computation for the given combination. A TURF run of 27 items, taken 8 at a time will require more than 2 million iterations to complete and may take many minutes of machine time.

SUMMARY

CHECK

The CHECK command is used to determine whether a P-STAT system file is complete or corrupt. CHECK produces a brief report and uses system variables which can be tested by subsequent commands to provide the details. The 3 system variables for the check command are:

.CHECKFILE. .CHECKVARS. .CHECKCASES.

The following is a typical report f

----------CHECK results for file paceset----------

File paceset is in Version 3 format.

It was made on Mar 20, 2012 at 17:29:00
using P-STAT version 3.01 rev 1.

It has an extension of PS1.

The file has 15 variables.
It has 80 cases.

Required Identifiers

CHECK fn

specifies the name for the P-STAT system file to be examined. The only supported PPL is [current] or [previous]. The command produces a brief report which includes the number of variables that can be read.

Optional Identifiers

MAX nn

MAX can be used to test that the file has at least that many good cases.

RESET

RESET can be used to reset the 3 system variables to 0. BRANCH can be used with a test of these 3 values to jump across commands depending on a CHECK result.

MODIFY SomeFile [IF state .eq. “OHIO:”, RETAIN], OUT = HAVEOHIO $
CHECK HAVEOHIO $
IF .CHECK.CASES. EQ 0, BRANCH STEP2 $

.....more processing here

STEP2: continue the run $

DEFAULT.SETTINGS / PRINTER.SETTINGS

DEFAULT.SETTINGS, OW 132, PAGE.CHARACTER 12, LINES 56 $

PRINTER.SETTINGS 'MyPrintFile', OW 200, VBAR ‘I’ $

LIST work, PR 'MyPrintFile', LINES 999 $

The arguments for DEFAULT.SETTINGS and PRINTER.SETTINGS can be used in either command or as standalone identifiers in any command which is producing printed output.

PR fn

PR provides the name for an output destination. Any argument in the current command takes precedence over an argument found in PRINTER.SETTINGS which takes precedence over an argument found in DEFAULT.SETTINGS.

LINES nn

Provides the number of print lines per page.

OW nn

Provides the maximum number of characters per line.

SCREEN nn

Provides the number of lines available on the screen so that screen holding can take place before the information scrolls away.

PPL: STANDALONE COMMANDS

Programming Language (PPL) which does not require P-STAT system file input can be used independently. This often involves a combination of text and one or more permanent scratch variables.

GEN ##temp = 0;

PROCESS Sales [IF month = 'MAY', increase ##temp by Items;] $

PUT 'Total items sold in MAY: ' ##temp $

SYSTEM VARIABLES

System variables differ in form from other variables because they both begin and end with a decimal point so that they cannot be confused with user variables. Because they are variables they can be tested and action taken as needed. If you are interactive, it is easy to make changes or fix a command but in a batch run where you cannot see the results as they happen these system variables are extremely useful.The manual "P-STAT: A Guide to the P-STAT Programming Language (PPL)" covers them in detail.

COLLECT

The PPL function COLLECT which is used to collect several cases and create a single case also makes good use of system variables.

1. .COLLECTSIZE. The number of cases in the most recent collect
2. .COLLECTMIN. The size of the smallest collected group so far
3. .COLLECTMAX. The size of the largest collected group so far
4. .COLLECTIONS. The total number of collects that occurred
5. .COLLECTSUM. The number of cases that have been collected.

DATES

There are many date functions. They are documented in “P-STAT: A Guide to the P-STAT Programming Language (PPL)”. These 6 basic functions are used to convert numeric dates to the P-STAT’s internal date format.

```
DAY.MONTH.YEAR        DAY.YEAR.MONTH
MONTH.DAY.YEAR        MONTH.YEAR.DAY
YEAR.DAY.MONTH        YEAR.MONTH.DAY
GEN Date:c = MONTH.DAY.YEAR ( 09152002 ) $
```

The statement “PUT .DATE. $” writes the current date in a default format. The function FORMAT.DATE can be used to supply an alternate format for printing dates.

10 AUTOSAVE: P-STAT System Files

Data values are stored in *P-STAT system files*. System files are made within the P-STAT software. The data are either entered into the system file interactively or read by P-STAT from an *external* file on disk. A P-STAT system file is a *self-documenting* file — that is, it has information about the data, as well as the actual data values themselves. A system file contains the following:

1. its file name and creation number,
2. names and descriptive information for each of the variables,
3. the heading information that was current when the file was built,
4. a short history of the command that created the file, and
5. the numeric and/or character data values.

When a P-STAT command reads or writes a P-STAT system file, that file is referenced by name in an appropriate phrase in the command.

P-STAT system files are *AUTOSAVE* files — that is, they are automatically saved when they are created or modified. They are written on disk and not just in a temporary workspace. When a system file is created, a *current* version of the file exists. If a new system file with the same name is created (possibly by modifying the existing one), the new system file becomes the *current* file and the older file becomes the *previous* version. (The *creation number* NOT the external file extension distinguishes between previous and current versions of a given system file.)

P-STAT system files are distinct from other files that P-STAT uses — files that are referred to as *external* files. System files are *binary* files that may be read and written only by P-STAT. External files are files that may be created outside of the P-STAT software. They can be text or binary files which can be read and written by many software packages. P-STAT uses several types of external files — raw data files, label files and command files, for example.

This chapter describes the elements of P-STAT system files, aspects to consider about organizing data for system files, and how P-STAT creates system files. In addition, the chapter discusses automatic file saving, erasing system files, the naming of autosave files, external files and temporary work files.

10.1 ELEMENTS OF SYSTEM FILES

A P-STAT system file can have an unlimited number of *cases* (rows) and 6,000 to 250,000 *variables* (columns), depending upon the size of the P-STAT system in use. Whopper/2, which is the version usually supplied, can have 6,000 variables per file.

Each case contains the same number of variables, and thus the file may be thought of as rectangular. Each variable in the file has a name. If variable names are not provided by the user, P-STAT generates names that are a combination of the letters VAR and the position of the variable in the file — that is, VAR1, VAR2, and so on.

The data values supplied to P-STAT are generally in the form of character strings or numbers. P-STAT converts these character strings and numbers into a form that permits them to be processed rapidly by P-STAT commands and that uses a minimal amount of storage space. Files written with this type of representation are called compressed binary files.

10.2 Case Identifiers

P-STAT system files often have one or more variables that identify each case in the file. They are typically variables such as “Case.Number,” “ID,” “Last.Name,” “SS.Number” and other similar identifying aspects of cases. Cases are generally data about one person, one trial event, one questionnaire or one item of some type. The case identifier variable has a unique piece of information about each case. It may be a numeric or character variable, and its presence in the system file is optional.

Some P-STAT system files have *row labels*. (These were required in very early versions of P-STAT.) A row label is a specific character variable that has the name “row.label” and that identifies a case. It can be renamed, dropped or tested, just like any other character variable. In most files, character variables such as “Name” or “ID.Number” identify each case. However, certain files produced by some P-STAT commands, such as correlation matrices, have row labels. Each value in a correlation matrix represents a relationship between two variables. The variable names indicate which variable is represented by each column of the matrix. The row label generated by the CORRELATE command indicates which variable is represented by each row of the matrix.

10.3 Relational Capabilities

Often, data in one case is related to data in one or more other cases. For example, this is the situation when the cases in a file represent household members — all the cases that are members of the same household are related by their common dwelling place. Data of this type may be stored in relational or hierarchical system files. Because P-STAT permits many system files to be active simultaneously, the choice of the type of file used for relational data is purely one of convenience and economy of storage.

The data in one file may easily be linked with that in a related file. For example, a case from a person file can be linked to the appropriate case in a house file. All that is needed is a common variable in both files to link the correct cases together. This common or *linking variable* provides *relational* capabilities. If the linking variables do not have the same name in both files, the RENAME instruction may be used to rename one or more of the variables. In this example, the variable House.Number is the link that relates the cases in the person file with the corresponding cases in the house file:

Person File:

Person Number	House Number	Age	Sex
1	103	26	1
2	103	24	2
3	103	2	2

House File:

House Number	Number of Rooms	Number of Bathrooms
103	6	1.5

The decision on whether to build many *relational* files or to hold all the information in a few *hierarchical* files, with some of the information repeated for successive cases, depends to some extent on how the files are to be used, the size of the files, and the availability of disk storage. An advantage of the relational multi-file model is that information is stored only once, rather than, for example, as duplicate household information for each case or person in the same household in one file. All household information is in a house file; all person information is in a person file. The prior example illustrates the relational model.

Relational files have several advantages. First, less space is used to store all the information since any given household is not duplicated. Second, corrections and updates to any household are done only in one place, to the particular case in the household file. The change is made once and can immediately be applied to all the persons in that household. However, it is necessary to combine the related files to access all information pertinent to all cases.

Hierarchical files store all information in one file. Information that is the same for several cases must be repeated:

One File:

Person Number	House Number	Age	Sex	Number of Rooms	Number of Bathrooms
1	103	26	1	6	1.5
2	103	24	2	6	1.5
3	103	2	2	6	1.5

The advantage of hierarchical files is that multiple files do not need to be combined to access all information pertinent to all cases. With small data sets, it does not matter how the data is stored; with large files, it may.

10.4 Creating System Files

Several commands exist for building P-STAT system files from data with a variety of input formats. The most important of these are TEXTFILE.IN, MAKE and MAKE.FIXED. There is an Introduction to TEXTFILE.IN and the MAKE command in Chapter 4 of this manual.. TEXTFILE.IN, MAKE and MAKE.FIXED are covered in detail in the manual “P-STAT: File Management”.

There are a number of commands which provide easy interfaces to data produced by other programs. TEXTFILE.IN and TEXTFILE.OUT can be used to read and write comma or tab delimited files with or without an initial record containing variable names. These provide easy access to data created by PC/Windows program such as EXCEL. SPSS.IN and SPSS.OUT are used to read and write data in SPSS’s portable file format.

10.5 Packing System Files

P-STAT system files are carried in a compressed or *packed* format. Character variables are carried without trailing blanks. Because it wastes space to use an entire 32-bit word for a small integer, several small integer values are packed into a single word wherever possible.

Numeric integer values carried in packed format require less space in storage, but they require slightly more computation time because the data must be unpacked before use. The use of packed files is usually advantageous.

10.6 Communicating With Other Systems

Because no single computer program can fulfill all needs, it is possible to read the data files produced by other programs and to write out the information from a P-STAT system file in a form that other programs can read. UNMAKE, FILE.IN and FILE.OUT are commands designed to read and write either binary or formatted records. Similarly, there are commands to read and write SPSS export format files and the tab-delimited files produced by EXCEL. These commands are documented in “P-STAT: File Management”.

10.7 AUTOMATIC FILE SAVING

P-STAT is an *AUTOSAVE* system. System files referenced in commands are automatically located.

```
LIST  Weather  $
```

Usually each file will be given a different name. However, it is not necessary to supply a different name for an output file. The name you like and remember, the input file name, can also be used for the output file:

```
SORT  Messages,  BY  Number,  OUT  Messages  $
```

Losing a P-STAT system file because of power outages or user errors is not likely. Merely restart after the crash — any completed files and the previous versions of files in use during the crash are there. If P-STAT has a problem reading an input file use the CHECK command *before* you try to do any modifications. If the file is corrupt, CHECK reports how many cases it can read and provides the instructions to retrieve that portion.

All files have a *current* version and, after they have been modified in any manner, a *previous* version. P-STAT keeps track of which is which. If the power fails during MODIFY, for example:

```
MODIFY  Class236,  OUT  Class236  $
```

the input file remains intact. If the MODIFY command did what you told it but not what you meant:

```
MODIFY  Dept16
 [SET  Commission = Commission + (.5  * New.Sales) ],  OUT  Dept16 $
```

the unmodified version is available:

```
ERASE.CURRENT  Dept16  $

MODIFY  Dept16
 [SET  Commission = Commission + (.05 * New.Sales) ],  OUT  Dept16 $
```

If you are done with a P-STAT session, pleased with all your output files, but concerned about disk space, one command makes housekeeping simple:

```
ERASE.PREVIOUS  $
```

gets rid of all previous versions of system files referenced in the session.

The following sections discuss the various aspects of the autosave system. The final four sections on directories, temporary files, and initialization options may be skipped by new users or postponed until they gain some experience and have more complex needs.

10.8 Current and Previous Files

P-STAT keeps track of the previous and current versions of files. You supply a file name of *sixteen* or fewer characters, and P-STAT adds the extension (suffix) ".PS1" or ".PS2". As that file is modified, the extension name alternates. However, at all times, P-STAT knows which file is the current one and which is the previous one. You use only the name you gave the file:

```
PLOT  Cells;
```

for example, and P-STAT inputs the current version to PLOT. However, if you want the prior version for some reason, just say so:

```
PLOT  Cells  [ PREVIOUS ] ;
```

The PPL instruction PREVIOUS is enclosed in square brackets and follows directly after the file name. It must be the first PPL clause. Additional PPL clauses may follow. The comparable instruction CURRENT is also available. When neither is used, it is assumed that the current file is the desired one.

PREVIOUS may not be used when the output file has the same name as the input file — that is, the previous version of a file cannot be modified to produce a current version. This is because the current version exists already. Erase the current version first:

```
ERASE.CURRENT  VirusA  $
```

and then do the modification:

```
MODIFY  VirusA  [ SET Count = Count/100 ],  OUT  VirusA  $
```

or, produce an output file with a different name:

```
MODIFY  VirusA  [ PREVIOUS;  SET Count = Count/100) ],
   OUT  VirusA2  $
```

10.9 Controlling AUTOSAVE

The current version of a P-STAT system file exists at all times; the previous version exists once an output system file of the same name is produced. Another output file with the same name overwrites the previous file, never the current one. AUTOSAVE automatically manages your system files. However, some aspects of automatic file saving may be controlled by users.

If you are super cautious and prefer to be queried when a previous file will be overwritten, use.

```
NO AUTOERASE  $
```

P-STAT will query you before replacing a previous file. Use:

```
AUTOERASE  $
```

to restore the regular autosave system.

Alternatively, if you are super confident, use:

```
REPLACE  $
```

Only current versions of a file remain after a command completes successfully. Protection still exists — the previous file is erased only *after* the new file is completely written. If a crash takes place, the current file still exists. Use:

```
NO REPLACE  $
```

to turn file replacement off. Most people use neither NO AUTOERASE nor REPLACE, but they are there in case you need them for certain situations. For example, REPLACE may be useful when you are working with large files and disk space may not be sufficient for permanent storage of two copies of each file.

10.10 Erasing Files

Previous and current versions of P-STAT system files may be selectively erased. Use:

```
ERASE.PREVIOUS  $
```

to erase *all* previous versions of all system files. Only files mentioned in the P-STAT session are affected. If only a current version of a file exists, the file is not erased. ERASE.PREVIOUS may be used with specific file names as arguments to erase the previous versions of just those system files:

```
ERASE.PREVIOUS  Employee  Accounts  $
```

The ERASE.CURRENT command requires file name arguments. It erases the current versions of system files:

```
ERASE.CURRENT  Exper101  $
```

If a previous version of a file does not exist, the current version is not erased. A general erase of all current files is not permitted.

The ERASE command is used to erase *both* the current and previous versions of P-STAT system files. If only a current version exists, it is erased; if both a current and previous version exist, they are both erased. ERASE requires one or more file names:

```
ERASE  Quest42  Quest49  Reply7  $
```

All P-STAT system files mentioned or used in a P-STAT session may be erased using the ERASE.AUTOSAVE command. ***Both the current and previous versions of all system files are erased.*** If only a current version exists, it is erased. Specific file names may not be given:

```
ERASE.AUTOSAVE  $
```

Files not mentioned in the P-STAT session are *not* erased. If any doubts exist, use the AUTOFILES command to list all referenced autosave files (see the next section) before using ERASE.AUTOSAVE.

Files other than P-STAT system files may also be erased. External files are erased using the ERASE.EXTERNAL command:

```
ERASE.EXTERNAL  'Session1.Lab'  'Plots.Lst'  $
```

Names of the external files to be erased must follow ERASE.EXTERNAL, and they should be enclosed in single or double quotes. External files include label files, print files, command files, libraries and raw data files.

The files are expected to be in the current directory unless their names contain the complete information necessary to locate them:

```
ERASE.EXTERNAL  'B:/City/StLouis.Dat'  $
```

Any information required by your operating system to find files should be included in the quoted name. Multiple quoted names may follow ERASE.EXTERNAL.

10.11 File Names

P-STAT autosave files must have unique file names of *sixteen* or fewer characters. The names should begin with a letter and may contain letters, numbers and decimal points (periods). The extension or suffix of ".PS1" or ".PS2" that P-STAT adds is in addition to the sixteen-character name. Autosave files are referenced within P-STAT with just the brief name:

```
LIST  Trial10  $
```

The name of the file as it appears in the directory listing, "TRIAL10.PS1" and "TRIAL10.PS2" (if another version of the file has been produced) is of no concern within P-STAT.

The FILES command displays the names of all P-STAT system files and external files mentioned or used in a P-STAT session. Both the current and previous versions of system files are listed, and the directories in which the files are located are shown. In addition, the names of external files and libraries (external files containing non-autosave P-STAT system files) are displayed. The AUTOFILES command may be used to show only the names of P-STAT system (autosave) files.

The letters in the extension "PS" that P-STAT appends to file names may be changed. The AUTOEXT (AUTOsave file EXTension) command specifies the desired letters enclosed in quotes:

```
AUTOEXT  'SB'  $
```

Now, autosave files will have the extensions ".SB1" and ".SB2". This makes it possible for multiple users to work in the same directory and produce P-STAT system files with the same names, without overwriting each other's files. In subsequent P-STAT sessions, each person would have to use AUTOEXT to specify the same extension to access his or her files.

When AUTOEXT is used without an argument:

```
AUTOEXT  $
```

the extension for autosave files is reset to "PS".

If you are planning to use AUTOEXT, you must be sure to use it every time or you will not be able to find your files. A good way to ensure this is to include the commands in a PSTART file which gets executed every time the a P-STAT run begins. PSTART files are explained in the first chapter of this manual.

10.12 DIRECTORIES

The directories (folders) where files are located can be specified at the beginning of a run by using the commands PSFILES, PSTEMP, PSAUTO and PSDATA.

```
PSFILES  '/usr2/lewis/cars' $
```

The directory name is enclosed in single or double quotes. PSFILES sets the three directories:

1. PSAUTO, the directory for all P-STAT system file;
2. PSDATA, the directory for all external files; and
3. PSTEMP, the directory for temporary files

to the same location. If the other commands are not supplied, all files used during the run will be read from or written to the PSFILES directory. PSTEMP can only be changed at the beginning of a P-STAT job PSAUTO and PSDATA may be changed as needed. If PSFILES is changed after the run begins, PSTEMP is not changed but both PSAUTO and PSDATA will now be set to the PSFILES directory.

If none of the directory commands are used and the environment variables are not set, the PSFILES path is assumed to be the current working directory. This is usually the directory from which P-STAT is invoked. If P-STAT is invoked from the program manager, the current working directory is the directory where P-STAT is installed.

10.13 Autosave Files

The PSAUTO command is used to specify the location for the P-STAT system (autosave) files only. The directory name is enclosed in single or double quotes. After PSAUTO specifies an autosave directory, all referenced P-STAT system files are expected to be located in this directory and all new autosave files are written in this directory. The autosave directory is the *pathname* for autosave files.

```
PSAUTO  '\usr2\lewis\cars'  $
```

When a pathname is defined (as in a PSAUTO command) a separator character at its end is assumed. Thus,

```
PSAUTO  '\usr2\lewis\cars' $        and
PSAUTO  '\usr2\lewis\cars\' $
```

do the same thing. The '\' is a PC Windows separator. On Unix the seperator is '/".

The LOCATE command may be used to provide an alternate directory for *specific* P-STAT system files. If the files exist they are read from that directory. If the files do not exist, they are written to that directory when the are created.

```
LOCATE  Trucks  Trains  '\usr3\lewis\transport'  $
```

The P-STAT system file names are not enclosed in quotes; the directory name is. All P-STAT system files that are not mentioned in a LOCATE command are read from and written to the current autosave directory.

PSAUTO may be used several times within a run to switch autosave directories. When PSAUTO is used without arguments:

```
PSAUTO  $
```

the autosave directory is reset to the initial autosave directory. Similarly, when LOCATE is used with just P-STAT system file names and no quoted directory name, the specified files are expected to be in the present autosave directory. This usage may precede changing the autosave directory:

```
LOCATE  Wind  Fire  $
PSAUTO  'E:\sue\psfiles'  $
```

In this example, the P-STAT system files “Wind” and “Fire” are expected to be in the present autosave directory. All other P-STAT system files will be read from or written in the new autosave directory given within quotes.

10.14 External Files

External files (*non*-autosave files such as data, labels, transfer and other files) that are referenced without a full path name are assumed to be in the PSDATA directory. Like PSAUTO, PSDATA can be set as an environment variable. If it is not specified, the PSDATA directory is assumed to be the same as the PSFILES directory.

```
PSDATA 'C:\Lewis\cars' $
```

For external files which are not in the PSDATA directory you may supply their complete names enclosed in single or double quotes. Alternatively, use the EQUATE command to give a short name that will be used to reference the external file:

```
EQUATE  CarLabs  '/usr2/lewis/cars/CarLabs'  $
```

The short reference name is not enclosed in quotes. The complete name that the host operating system uses to locate the file is enclosed in single or double quotes. (Note that this is the complete file name, and not just a directory.)

```
PSDATA  '/usr2/lewis/cars'  $
MAKE Cars, FILE 'car.dat' DEF 'car.def' $
SURVEY Cars, LABELS 'CarLabs';
```

The MAKE input (both the data and the definitions) and the labels for SURVEY are found in the PSDATA directory without the need to provide the full path with each file name.

The PSDATA command can be used to change to an alternate directory as needed. Individual external files can always be referenced with the full path and file name in quotes. EQUATE is useful when a single external file, which is not in the PSDATA directory, is referenced several times during a P-STAT run.

The FORGET.EXTERNAL command forgets external files. It is useful when a nonexistent external file has been equated with a short reference name. The external file is forgotten and the reference name may be equated with the actual external file:

```
FORGET.EXTERNAL  CarLabs  $
EQUATE  CarLabs  '/usr2/lewis/cars/Car.Labs'  $
```

FORGET.EXTERNAL may be used without arguments to forget all external files that were referenced in the current P-STAT session.

10.15 Temporary Files

There are two types of temporary files created during a P-STAT session.

1. Work files are temporary P-STAT system files.
2. Scratch files are not P-STAT system files but have a variety of different internal formats and are used by P-STAT commands to hold intermediate results.

If you are making many intermediate P-STAT system files that you do not want hanging around after your P-STAT session, give them names beginning with “WORK”. They are automatically erased after you exit P-STAT with the END command. Only files beginning with “WORK” that are referenced in a P-STAT session are affected. However, it would be prudent not to give files names beginning with “WORK” unless they really are temporary files. The names “WORK0001” through “WORK9999” are reserved. P-STAT uses these names for temporary files created during macro processing (files in macros whose names begin with “MACFILE”) and during processing of some commands such as COUNTS (interim output files that are listed).

A file named “Work1” is stored on disk with a name like:

3. PC/Windows W_100#WV9@N1$$$$.PS1 .

4. UNIX W_10cYET5L.PS1 — The encoding includes the process id which makes this name unique for up to 9999 work files in a single P-STAT session.

P-STAT uses *scratch* files to hold intermediate results in the processing of commands and for other temporary purposes. For example, in sorting a large P-STAT system file, several scratch files of partially sorted cases may be produced before the final merge step combines these scratch files and produces the output system file.

Most users are unaware of these scratch files, unless a P-STAT session aborts because of a power failure, insufficient disk space or another reason. These files have a the prefix "P_" and end with the extension ".TMP". The rules for providing a unique name are the same as the rules for work file names. Thus a short name for a scratch file on DOS is something like "P_286LAR.TMP".

Temporary files, both work files and scratch files, are located in the current working directory, unless a PS-TEMP directory has been specified. PSTEMP may be provided either as an environment variable or as a command. However, if it is used as a command it can only be used at the beginning of the P-STAT run. The following is a command appropriate for the PC.

```
PSTEMP  'C:\usr\top\will' $
```

The directory name is enclosed in quotes. After this PSTEMP command, all temporary files are opened on the "c:\usr\top\will" directory.

10.16 INITIALIZATION COMMANDS

Various commands help initialize a P-STAT session the way you would like it to be. These commands may also be used as environment variables by users with the various PC or UNIX operating systems. The environment variables are set *prior* to evoking the P-STAT software. Environment variables are discussed in the first chapter of this manual. The following environment variables may be provided or changed as commands within a P-STAT run.

1. PSFILES provides the full path to a directory where the files to be used are located. It sets the initial values for PSAUTO, PSTEMP and PSDATA. If PSFILES is used after the run begins it does not change the setting for PSTEMP but it does change PSAUTO and PSDATA..

2. PSTEMP provides the full path to a directory where temporary files are to be stored. PSTEMP must be at the beginning of the P-STAT session.

3. PSAUTO gives the complete name of the autosave *directory* when it is other than the PSFILES or current working directory. LOCATE can be used with individual files to override the PSAUTO setting.

4. PSDATA gives the full path to a directory that contains external files such as labels files or the input data for the MAKE command. If PSDATA is set, full pathnames are needed only for those files that are not stored in that directory.

```
PSDATA 'F:\LABELS\P0198'  $
 LIST STUDY4,  LABELS 'S4.LAB' $
```

The labels used for this list are in file F:\LABELS\P0198\S4.LAB .

5. PSHELP gives the complete name of the P-STAT help file: It is used when the name of the help file is changed, when the help file is located in a different directory than expected, or when the environment variable PSHELP has not been set.

```
PSHELP  'C:\PSTAT\NEW\PSTAT.HLP' $
```

SUMMARY

AUTOERASE

```
NO AUTOERASE  $
```

The AUTOERASE command automatically manages P-STAT system files. When a new version of a file is produced, both the current and the previous versions are available. When another version of the file is created, the previous file is erased. The current file becomes the previous file and the new file is the current file. (P-STAT system files should have names of eight or fewer characters. The names should begin with a letter and may contain letters, numbers and decimal points.)

AUTOERASE is assumed by P-STAT. NO AUTOERASE may be specified. P-STAT then queries a user before overwriting the previous version of a file. In batch mode, the run ends if NO AUTOERASE is specified and a third version of a system file is produced.

AUTOEXT 'cs'

```
AUTOEXT  'GJ'  $
```

The AUTOEXT command supplies a two letter string to be used in P-STAT file name extensions. The specified string replaces the “PS” normally used in extensions. This permits multiple users to work in the same directory and have files with the same names, without overwriting each other’s files. AUTOEXT, without an argument, restores “PS” as the file name extension. If you use AUTOEXT as a regular part of your naming scheme, you should include it in your PSTART file so that it is executed automatically when a run begins.

AUTOFILES

```
AUTOFILES  $
```

The AUTOFILES command displays the names and pathnames (associated directories) of P-STAT autosave system files. Use FILES to show libraries and external files, as well as autosave files.

END

```
END  $
```

The END command exits P-STAT. The current and previous versions of all P-STAT system files remain. When P-STAT is reentered subsequently, a user need only mention a file by name. P-STAT checks the autosave directory and uses the current version of that file.

Temporary files are given names beginning with “WORK”. The files exist throughout the P-STAT run. Upon exiting P-STAT, however, temporary files are erased — that is, END erases all system files referenced in the P-STAT session whose names begin with “WORK”.

EQUATE fn 'fn'

```
EQUATE  Field.Labs  'd:\bill\expmts\Field.Labs'  $
```

The EQUATE command gives a brief name to use in referencing *external* files that have long names or that are located in directories other than the current one.

ERASE fn fn ....

```
ERASE  Trial1  Trial2  $
```

The ERASE command deletes *all* versions of a file. Both the current and previous versions of the specified files are erased. See ERASE.PREVIOUS if only previous versions of a file or files should be deleted.

ERASE.AUTOSAVE

```
ERASE.AUTOSAVE  $
```

The ERASE.AUTOSAVE command deletes *both* the current and previous versions of *all* P-STAT system files used in the P-STAT session. It does not erase files not mentioned during the P-STAT session.

ERASE.CURRENT fn fn ....

```
ERASE.CURRENT  Sales87  Taxes87  $
```

The ERASE.CURRENT command deletes the current versions of specified files. If a previous version does not exist, the file is not erased. One or more file names must follow ERASE.CURRENT.

ERASE.EXTERNAL 'fn' 'fn' ....

```
ERASE.EXTERNAL  'Grass.Lab'  'Grass.Lst'  $
```

The ERASE.EXTERNAL command deletes external files. One or more file names must follow ERASE.EXTERNAL. The file name should be enclosed in quotes. A full path should be supplied when the current path for external files is not appropriate (see PSDATA).

```
ERASE.EXTERNAL  'B:\bill\seedmix.dat'  $
```

External files include label files, command files, print files, libraries and raw data files. The files need *not* have been mentioned in the current P-STAT session.

ERASE.PREVIOUS fn fn ....

```
ERASE.PREVIOUS  BrandX  BrandY  $
ERASE.PREVIOUS  $
```

The ERASE.PREVIOUS command deletes the previous versions of specified files. When no file names follow ERASE.PREVIOUS, *all* previous versions of files are erased. If only a current version of the file exists, the file is not erased. Files not referenced in the P-STAT session or files that are "forgotten" (see the FORGET command) are not erased.

FILES

```
FILES  $
```

TheFILES command displays the names of *all* files referenced during a P-STAT session — P-STAT autosave system files, libraries and external files. Use AUTOFILES to see just the autosave files.

FORGET.EXTERNAL fn 'fn' ....

```
FORGET.EXTERNAL  Boys.Lab  $
FORGET.EXTERNAL  '/usr2/tom/study12'  $
```

The FORGET.EXTERNAL command forgets the specified external file or files. When the argument for FORGET.EXTERNAL is a reference name (specified previously with EQUATE), it is not enclosed in quotes. When the argument is the actual file name, it is enclosed in single or double quotes. Without any arguments, FORGET.EXTERNAL forgets *all* external files referenced in the current P-STAT session.

LOCATE fn 'fn'

```
LOCATE  fibers  colors  'B:\pstfiles\threadco'  $
```

The LOCATE command associates specific P-STAT system files with the directory or path given within quotes. Those system files are read from or written to the specified directory. Other P-STAT system files are expected to be in the present autosave directory.

LOCATE may be used without arguments to associate specific system files with the present autosave directory prior to changing that directory:

```
LOCATE  Exper12  Exper13  Exper14  $
PSAUTO  '/usr2/liz/wilcomp'  $
```

See the PSAUTO command for further information on its use.

PATHNAMES

If P-STAT cannot find a file, it may be because it is looking in the wrong directory. PATHNAMES prints the names of all the assumed or defined paths. You can then use PSAUTO, PSDATA or LOCATE to correct the problem

PSAUTO 'dir'

```
PSAUTO  'E:\pkgs\tony\psf'  $
```

The PSAUTO command gives the complete name of the autosave directory. All P-STAT system files are read from and written in the autosave directory. The assumed setting for the autosave directory when PSAUTO is not used is the *current working directory* or, if used, the PSFILES directory. PSAUTO may be used as an *environment* variable in systems that support such variables.

PSAUTO may be used without arguments to set the autosave directory back to the initial run-starting autosave directory:

```
PSAUTO  $
```

See the LOCATE command to reference specific P-STAT system files in other directories, without changing the autosave directory.

PSDATA 'fn'

```
PSDATA  'E:\Projects\1298X' $
```

PSDATA gives a path to a directory where data files are located. If PSDATA is not used, external files which do not specify a path are assumed to be in the *current working directory* or, if used, the PSFILES directory. PSDATA may be used as an *environment* variable in systems that support such variables. Supplying the full path and file name in quotes or using EQUATE are other ways to reference external files without changing the PSDATA directory.

PSFILES 'dir'

PSFILES tells P-STAT what directory to use for the files in a P-STAT session. It sets PSAUTO, PSDATA and PSTEMP.

PSHELP 'fn'

```
PSHELP  'c:\pstat\old\pstat.hlp'  $
```

The PSHELP command gives the complete name of the P-STAT help file. PSHELP may also be used as an environment variable.

PSTEMP 'dir'

```
PSTEMP  '/usr/tmp/john'  $
```

PSTEMP gives the complete name of the directory where temporary files are to be written. PSFILES can also be used to supply a PSTEMP directory. Temporary files include P-STAT system files which begin with "WORK" and files used for intermediate results during the execution of many P-STAT commands. Temporary files are automatically erased when P-STAT ends normally. PSTEMP, like PSFILES must be one of the first commands entered in the P-STAT session.

PPL Instructions:

CURRENT

```
COMPARE    File12  [ CURRENT ]   File12  [ PREVIOUS ],
  BY  ID,  OUT  FileDifs  $
```

The *PPL instruction* CURRENT may follow any file name in any P-STAT command to specify that the current version of the file is desired. This instruction must be the *first* PPL clause; additional PPL clauses may follow. When neither the CURRENT nor PREVIOUS instruction follows a file name, it is assumed that the current version of the file is desired.

PREVIOUS

```
LIST  Males  [ PREVIOUS;  CASES 1 TO 20 ]  $
```

The *PPL instruction* PREVIOUS may follow any file name in any P-STAT command to specify that the previous version of the file is desired. This instruction must be the first PPL clause; additional PPL clauses may follow. When neither the CURRENT or PREVIOUS instruction follows a file name, it is assumed that the current version of the file is desired.

PREVIOUS may not be used when the output file has the same name as the input file — that is, the previous version of a file cannot be modified to produce a current version. This is because the current version exists already. Use the ERASE.CURRENT command to erase the current version of the file; then the previous version of the specified file is used as input (it is then the current or only version).