QTAB

A Professional Cross-Tabulation Program

User's Guide

Revision: 21.1.5

QTAB is the tabulation component of the QUIP System

QUIP System software is available from:

www.quipsoftware.com

QTAB documentation by

Jan Werner Data Processing

TABLE OF CONTENTS

INTRODUCTION

The QUIP System is a software package that allows almost any data set to be tabulated and the results presented as fully annotated tables. It includes programs to handle data manipulation and conversions, validity checking, marginal counts and weighting, as well as cross-tabulation.

The QUIP System was designed to meet the needs of data processing professionals. Most of the programs are script-driven using a common language. This allows for batch processing or for users to create their own custom environment.

The programs in the QUIP System share a common specification language that allows users to describe extremely complex combinations of logical conditions and numeric expressions. This language is compiled in memory at run-time, so all programs run very quickly on large data files.

This guide provides an introduction to QTAB, the cross-tabulation component of the QUIP System. It aims to teach beginners the basics of QTAB, and to provide a reference for more experienced users.

The following conventions are used throughout this document:

Angle brackets:	<...>	A mandatory item to be specified.
Square brackets:	[...]	An optional item to be specified.
Vertical bar:	..|..	Only one of the listed items should be specified.

QTAB keywords are in uppercase and the minimum abbreviation for each in bold uppercase, lowercase letters are used where a number should be entered (e.g., **WID**TH nnn).

1 THE SPEC FILE

QTAB instructions are called specifications, or "specs" for short. QTAB reads the specs from an ASCII text file called a <u>spec file</u>. The spec file contains all the information the program needs to process the data and produce the desired output.

Specs must be written using a text editor that can write files that do not contain any embedded control codes or tab characters. Case is ignored for specs and keywords, but is preserved in annotation, tags, dictionary definitions and literal strings within quotes.

Column position on the text line is significant in many QTAB instructions: Table definitions and post processing instructions use column position to identify certain fields and a specific column is used to separate specifications from label text in variable definitions. Spaces are used as separators between keywords, but the number of spaces does not matter. Tab characters are not allowed in specifications and are treated as non-blank characters in label annotation.

Any line that begins with an asterisk (*) in column 1 is treated as a comment. It will print in the spec listing but otherwise be ignored by the program, as will any completely blank line.

QTAB compiles the spec file in a single pass so forward references are not allowed. Items such as a numbersets, post processing instructions and dictionary entries must be defined before they can be referenced elsewhere in the spec file.

After a spec file has been written, QTAB may be run from a command prompt by entering:

```
            QTAB <filename> [command line directives|parameters]
```

QTAB will read the spec file named on the command line and compile it, checking for syntax errors and writing a listing file back to disk as it proceeds. If no fatal errors are found, QTAB will tabulate the data file(s) and write out a report file containing the tables generated and optionally, a table of contents file, both of which can be printed or viewed with a text editor.

1.1 Essential Elements

Every QTAB spec file must contain at least four basic sets of specifications, which should appear in the following order in the file:

DATA DIRECTIVES:	Instructions that describe the input (data) and output (listing, tables and contents) files.
GLOBAL LEVEL OPTIONS:	Instructions and parameters that apply to an entire QTAB run.
VARIABLE DEFINITIONS:	Sets of instructions that tell QTAB what to count and how to label rows, columns and tables.
TABLE DEFINITIONS:	Instructions that tell QTAB which variables to cross-tabulate by each other.

Other elements, such as post-processing instructions, numbersets and weights are placed with the variable definitions. Dictionaries may appear anywhere before the table definitions section. All variables defined in a spec file are processed during the compile phase of a QTAB run, but data is only tabulated for those variables actually referenced in the table definitions.

1.2 Keywords and Statements

Most options in QTAB are specified using keywords separated by spaces. Many keywords accept options which follow after one or more spaces. Other keywords, such as those used in the specifications area of variable definitions, are followed immediately by their parameters. When multiple options or parameters are used, they are separated by either commas(,) or semi-colons(;) depending on the keyword, and with no spaces allowed in the option list.

A QTAB statement is a line of text beginning with a keyword in column one and often followed by a name and/or various options.

An END statement begins with the keyword END in column one and contains no other text.

The sections and elements of a QTAB spec file begin with a statement. Some may be entirely defined in a single statement line while others may contain many lines of specs, terminated by an END statement. When the end of one element is clearly defined by the statement beginning the next element, an implicit END statement is generated, and the physical END statement may be omitted, as, for example, between variables or post processing instruction sets.

An explicit END statement must always be used to separate major sections of the spec file, such as to mark the start of the table definitions, to end a dictionary, and before and after any part of a spec file controlled by such as #INCLUDE or STOPSPEC.

2 DATA DIRECTIVES

Data directives identify the input data files and the output files created by QTAB. Each data directive begins with a keyword followed by the file name.

2.1 File Names

QTAB accepts the following data directives:

INTAP[n]	<[path\]filename.ext (lrecl> [V]	Input data file.
LISTING	<[path\]filename.ext>	Job spec listing file.
REPORT	<[path\]filename.ext>	Output tables file.
CONTENTS	<[path\]filename.ext>	Table of contents file.
COMPUTE	<[path\]filename.ext>	Compute file (see below)
RECOMP	<[path\]filename.ext>	Recompute file (see below)

If no LISTING file is specified, QTAB will create one in the current directory, generating a unique name for it. All other output files are created only if that data directive is present in the spec file. If no explicit path is given, QTAB always reads from and writes to the current directory. QTAB accepts any valid DOS or Windows long file names with no embedded spaces.

INTAP names an input data file followed by a left parenthesis and the record length in bytes. Record length is assumed fixed unless specified as variable, meaning that each consecutive block of bytes of the specified record length is considered a separate record and QTAB checks that the file size is a multiple of the record length, issuing a warning if it is not.

The letter "V" after the record length indicates variable length records, each of which must be terminated by an end-of-line marker (Carriage Return-Line Feed). For variable length records, the specified record length must be equal or greater to that of the longest record in the file, not counting the 2 bytes for the CR-LF.

A QTAB run may have any number of INTAP files (usually numbered INTAP1, INTAP2, etc.). Data files are processed in the order they appear in the spec file, regardless of the numbering used for the INTAP keywords. All files should have the same record length specified.

If no INTAP is specified, QTAB will issue a warning but will otherwise run normally, generating empty table shells if a report file is called for.

2.2 Compute and Recompute

If a COMPUTE data directive is given, QTAB writes out a file containing the raw table counts in a binary format. The tables can be rerun from that file by naming it in a RECOMP data directive and calling for the RECOMPUTE run option (see section 3.2).

In a recompute run, the names, sequence and dimensions of tables must not be changed. Subtotals (see section 6.1) may not be added or removed, but the actual subtotal specifications can be changed and will be recomputed. Any text labels and table options may be changed. The #A array field operator (see section 6.3.3) should never be used in recompute runs.

2.3 Command Line Data Directives

Data directives may optionally be specified on the command line using the following switches:

/C=<[path\]filename.ext>	COMPUTE file name
/I[n]= <[path\]filename.ext>	INTAP file name
/IL[n]= <nnn>	INTAP record length
/L=<[path\]filename.ext>	LISTING file name
/R=<[path\]filename.ext>	RECOMP file name
/S[n]=<[path\]filename.ext>	SPEC file name
/T=<[path\]filename.ext>	REPORT (tables) file name
/TC=<[path\]filename.ext>	CONTENTS file name

Multiple data directives are separated by spaces (e.g., /X=xx /Y=yy /Z=zz). The total length of the command line (including program invocation) may not exceed 127 characters. Up to 4 spec files may be listed on the command line; they are read in numeric sequence and treated as a single spec file. Data directives contained in the spec files always override those provided on the command line.

Up to 4 INTAP files may be specified for a run, with a matching record length for each (/ILn=). Variable length record files cannot be specified in command line data directives.

3 GLOBAL OPTIONS

Global options apply to an entire QTAB run. Most are specified on a RUNOPS line which must appear immediately after the data directives at the beginning of the spec file. Most run options will default to a specific setting unless they are explicitly defined otherwise.

3.1 Address Mode

QTAB requires a data address mode to be defined before any specifications can be processed.

The SETUP ADDRMODE statement determines whether specs refer to 1-byte ASCII characters or the 2-byte column-binary format used mostly in marketing research. It also specifies whether the data are addressed using card/column or direct offset notation.

The concept of the address mode is fundamental to understanding how QTAB processes data: The address mode determines how the program interprets the specification of data locations and their contents in the remainder of the spec file. For more on how ADDRMODE affects data addressing, see section 5.1.1.

To set the address mode, use one of the following statements at the beginning of the spec file:

SETUP ADDRMODE B	Column-binary data addressed by card and column.
SETUP ADDRMODE C	Character data addressed by position in record.
SETUP ADDRMODE D	Character data addressed by card and column.
SETUP ADDRMODE E	Column-binary data addressed by position in record.

If address mode is not explicitly stated, QTAB defaults to ADDRMODE B (column-binary data, card/column addressing).

3.2 Run Options

Run level options are specified on a line beginning with the keyword RUNOPS. Options are invoked by keywords separated by spaces, some of which may be followed by one or more parameters. RUNOPS line options are listed below with their parameters.

EXCEL	Delimit cells with tab characters (See section 12.2 for details).
ERROR ALL|nn	Set level for warning or error messages to prevent QTAB from running after compilation. ALL forces an end if any messages were generated. Level 4 allows warning messages.
FOOTPOS	Position footnotes one line after the last line instead of at the bottom of the last page of each table.
LENGTH nnn	Maximum page length nnn lines. Default is 63.
LOTUS	Output individual tables as separate files (See section 12.2).
NAME C|R|L|A	Print table name Center/Right/Left/Alternate sides of each page. Default is don't print.
NOCENTER	Do not center tables on the page (all tables, titles and footnotes will be left-justified). Default is centered within page width.
NLZ	Do not print a zero before the decimal point for numbers less than 1.0 (e.g., print ".5" instead of "0.5"). Default is to print left zero.
NOCHECK	Do not check for spec references beyond the input data file record length (note that this can cause the program to crash at run time if locations outside the data area are actually accessed).
NOCONTINUED	Do not print "(Continued)" under the table name on continuation pages after the first page of a table. Default is to print.
NOREPEAT	Do not reprint the % base row when continuing a table to a new page. Default is carry % base.
NOWTMSG	Do not show individual warning messages for each case with a record level weight that evaluates to zero (see section 5.3).
PAGENO R|L|A[,nnn]	Number pages at the top Right/Left/Alternate side, starting optionally with page nnn. Default is no page numbering.
RECOMPUTE	Use the compute file named in the RECOMP data directive as input instead of tabulating the data (see section 2.2).
TCONT	Print the table of contents only (must specify CONTENTS data directive, see section 2).
TEST SPECS|nnn|ALL	Stop after testing specs/Run only on first nnn cases/all cases. Default is run ALL cases.
WIDTH nnn	Set the page width for output. Default is 132.

The **STAR** and **ZCELLS** table options (see section 7.4) may also be specified on the runops line, in which case they will apply to all tables in the current run.

3.3 Run Titles

Run titles, a block of text that will appear at the top and/or bottom of every page of output, may be specified by a RUNTITLES statement, followed by the desired text, and terminated by an END statement. The format is:

```
RUNTITLES   TOP|BOT   R|L|C
Text goes here
(as many lines as needed)
END
```

The parameters indicate whether the text is to be printed at the top or bottom, and at the right, left or center of each page. Two separate RUNTITLES sections may be specified in a spec file: one for the top, and one for the bottom of each page.

3.4 Summary Row and Column Labels

In addition to the rows and columns explicitly defined in the specs (detail rows and columns), QTAB always computes several summary rows and columns for each table: total cases, number answering, number not answering, total mentions (total number of responses tabulated in all the rows or columns of the table). Rows and columns outside the table area are also reserved for table-level statistics. Summary rows and columns are only printed if called for using the SUM or STATS keywords in a table definition (see section 7.4). By default, QTAB assigns the labels listed in the table below to these summary vectors or table-level statistical vectors.

The summary vector names and their default labels are:

	Vector Name	Row Label	Column Label
Total	TT	TOTAL	TOTAL
No Answer	NA	NO ANSWER	NA
Total Answering	TA	TOTAL ANS	TA
Total Mentions	TM	TOTAL MENT	TM
Mean	MN	MEAN	MEAN
Standard Deviation	SD	STD. DEV.	SD
Standard Error	SE	STD. ERR.	SE
Variance	VR	VARIANCE	VAR
Median	MD	MEDIAN	MEDN

The following labels are applied to the vectors named when alternate print options are called for with the SUM table option, as described in section 7.4.4:

TT - Unweighted cells	TU	BASE	BASE
TM - at end of table	MB	SIGMA	SIGMA

Default row labels may be changed in a LABELTEXT section, starting with a LABELTEXT statement and is terminated by an END statement. Definitions begin with the vector name in cols. 1-2 and text starts in col. 5. Continuations on a 2nd line are indicated by the letter C in column 4. There is no limit on width, but at print time, text continuing past the stub width for any table will be truncated. The following example shows the use of LABELTEXT:

```
LABELTEXT
TT  Total Respondents
TM  Total Mentions
MB  Total Mentions when printed
MB Cat the bottom of tables
END
```

4 VARIABLE DEFINITIONS

Variables are sets of instructions that define the rows or columns of a table and supply the text labels to identify them in the report. Every QTAB variable begins with a VAR statement giving it a unique name and providing basic information about the class of variable and text options. All spec lines following the VAR statement, up to the next END, VAR, NS, MACRO or PP statement, are used to define the cell contents, labels and title information for that variable.

Each spec line of a variable specification is divided into two distinct parts: the left-hand side is used to specify cell contents (specs), while the right-hand is used to provide text annotation for the cells or title information for the variable. The two parts of each spec line are separated by the "spacer column" in the Label parameters of the VAR statement.

There are three basic classes of variables:

HEADER variables:	Define the contents and the labels for the columns of a table (Headers are sometimes called "Banners").
STUB variables:	Define the contents and the labels for the rows of a table.
BASE variables:	Define filters which apply to the entire contents of a table.

While the same specification language is used to define cell contents, cell labels are defined differently for each class of variable.

4.1 The VAR statement

The VAR statement begins with the keyword VAR followed by name up to 8 characters long, the LABEL keyword and the label options which identify the class of variable and spacer column.

```
        VAR <name> [TYPE S] LABEL <type>|<Equate>[,equate,...] [PP <set1[,set2,...>]
```

The variable name may use any printable ASCII characters. Case is ignored. Both variable class and spacer column (section 0) must always be specified, either explicitly or implicitly through an Equate to another variable (section 0).

Keyword "TYPE S" forces the variable to be single valued: If used, a case will only be counted once, in the first row or column for which it meets the conditions specified, regardless of how many subsequent rows or columns it would qualify to be counted in.

The PP keyword indicates that post processing (see section 8) is to be executed on tables using this variable. The PP keyword must be followed by the names of one or more post processing sets separated by commas.

4.1.1 Classes of Variables

The three classes of variables are defined as follows:

Header variables:	**H**<ww><N|V><ss>
Stub variables:	**S**[<ww><N|,|R|G>]<ss>
Base variables:	**B**[<ww><N|,>]<ss>

Where the parameters following the class code are as defined as follows:

ww	Text width: the maximum width of row labels for stubs or the nominal width of each table column for headers.
N	Normal fixed width text labels.
,	(Comma) Same as N, in stub and base variables only.
V	Headers only: Indicates variable column widths (see section 4.2.3)
R	Stubs only: Indicates that the column following the spacer column for a stub variable is reserved for ranking level codes (see section 4.3)
G	Stubs only: Generates labels automatically using the actual row specifications.
ss	Spacer column: The column separating specifications (on the left) from label annotation (on the right). Contains the codes that describe the annotation lines for headers and the vertical spacing for stubs.

The spacer column must always be specified, either explicitly or through an equate (see below). Width and class must be specified for headers. For stubs and bases, labels default to a normal width of 24 if not specified.

4.1.2 Examples of VAR statements

The format of the VAR statement is illustrated by the following examples:

`VAR banner1 LABEL H08V30`	A header variable named "banner1" with a nominal width of 8 characters per column. Column spacing is variable and text begins in column 31 of each specification line.
`VAR income LABEL S24,16`	A stub variable named "income" with an expected maximum width of 24 characters for the row labels which start in column 17 of each line.
`VAR whybuy L S30R24`	A stub variable named "whybuy" whose rows will be printed in rank order (See section 4.3 below). For each line, column 24 is the spacer column, column 25 is reserved for rank level codes and columns 26 to 54 are used for row label text.
`VAR women LABEL B20`	A base variable named "women", with cell text starting in column 21 of each line.

4.1.3 Variable Equates

LABEL options may include "equate" codes telling QTAB that specs or labels are not defined in this variable, but should be copied from some other variable previously defined in the spec file.

Each equate code must be followed by the name of a valid variable with no spaces in between. Multiple elements may be equated, separated by commas.

The equate codes are:

E<var>	Copy all label text, except for titles and footnotes, from the named variable to this variable.
T<var>	Copy title text from the named variable to this variable.
F<var>	Copy footnote text from the named variable to this variable.
P<var>	Copy post processing (see section 8) from the named variable to this variable.
Z<var>	Copy both title and footnote text from the named variable to this variable.
Vnn;<var>	Vector offset: copy the vector definitions from the named from the named variable to this variable, offsetting all specifications by nn data columns.

The following examples illustrate variable equates:

`VAR attr2 LABEL Eattr1`	A variable named "attr2" of the same class and using the same labels as a variable named "attr1", previously defined.
`VAR c1 L V2;a1,Eb2`	A variable named "c1" of the same class and using the same labels as variable "b2", with specs copied from variable "a1", incrementing all data addresses by 2 columns from those specified in "a1". Both "a1" and "b2" must have been previously defined.
`VAR c1 L V-10;a1,Eb2,Pc3`	Same as the previous example except that all data addresses are decremented by 10 columns and post processing is copied from variable "c3".

Equates copy an specs or label text as a whole unit, with the only change allowed a common column offset that affects all specifications in the variable. For a more flexible way to generate many similarly structured variables from a common template, see section 10, Macro Templates.

4.2 Labels and Table Annotation

Annotation for QTAB tables is entered into the spec file exactly as it will be printed in the final output. This allows existing text (from questionnaires, code lists, etc.) to be used for labels and titling. QTAB treats all text to the right of the spacer column of a variable as annotation. Codes in the spacer column tell what the annotation applies to and what formatting options to apply.

A spec line in a variable definition with no annotation in the spacer column or beyond is ignored for labeling purposes. To generate a blank line, there must be a code in the spacer column or a non-printing character (such as a tab character) in the text area.

4.2.1 Titles and footnotes

All variable lines with a "T" or a "Q" in the spacer column are considered titles and print at the top of the table, above the header annotation. All lines with an "F" in the spacer column are considered to be footnotes and will print after the last row of the table.

Title and footnote lines may be as wide as the page width specified in the run options and print exactly as entered, in the order they appear in the variable definition. Titles and footnotes are centered on the page unless the NOCENTER run option is used to left-justify (see section 3.2).

Each variable may have up to 10 lines of titles and 10 lines of footnotes.

4.2.2 Stub Annotation

Stub variables define the rows of a table and provide label text for each row. Label annotation is entered following the spacer column just as it is to appear at the left margin of printed tables. QTAB checks the number of rows labeled against the number defined in the specification part of all stub variables; if they do not match, a warning message is issued and tables referencing that variable will be skipped if the error level (see section 3.2) is set to allow warnings.

Labels for the rows of data in a table are identified by the following codes in the spacer column:

(blank)	Initial line of text for this row (if the rest of the line is not blank).
A	Skip an extra line after this row.
B	Skip an extra line before this row.
C	Text continuation line for the same row as in the previous line.
U	Text continuation printing on the same line as the previous one (QTAB issues a CR without a LF). Used to underline or double-print.
D[(nn)]	Do not print this row, [optionally, nn rows beginning with this one].
E	Eject the page after this row (formfeed after).
P	Eject the page before this row (formfeed before).
V	Omit the normal blank line between this row and the next row.
Z	Override zero suppression for this row, i.e., print the row even if there is no data and the ZEROSUP option applied to the table (see section 7.4).

Subtitles are lines of label annotation that do not correspond to rows of data in a table and are identified by the following spacer codes:

N	Subtitle with a page eject (formfeed) before it
S	Subtitle with an extra line skip before it
X	Ordinary subtitle text
Y	Subtitle with an extra line skip after it

All summary labels described for base variables (see section 4.2.4) may also be used in stubs.

Text wider than specified in the VAR statement (see section 4.1.1) causes a warning message. If the error level allows warnings (see section 3.2), subtitle labels wider than the nominal width specified may be used to extend annotation into the body of a table between the data rows.

4.2.3 Header Annotation

Header variables (sometimes called "Banners") define the columns of a table.

Label parameters N or V in the VAR statement (see section 4.1.1) determine whether the detail columns will be spaced at a constant or a variable width. Summary columns (see section 3.4) always print at the nominal column width specified by the text width parameter and cell contents are always right-justified within each column.

Header annotation is entered exactly as it should be printed over the body of the table and need not be aligned with the cell contents. Up to 10 lines of text may be specified and may extend beyond the right edge of the table body. For normal (constant) spacing, the table body width is determined by the number of columns to print.

For variable spacing, the last line of the header label annotation must be a width mask template (see spacer code "W" below) that indicates the exact placement of each detail column and determines the width of the table body. When variable spacing is used, the number of columns specified is checked against the mask and a warning issued if they do not match.

The following spacer codes may be used in header variables:

(blank)	Normal line of text (if the line contains any label text)
H	Normal line of text (allows a blank line to appear in the annotation)
U	Line to be printed on the same line as the previous one (QTAB issues a CR without a LF). Used to underline or double-print.
W	Width mask for variable width headers. Equal signs (=) are used to mark where the right edge of each column is to be positioned. The last column is marked by a colon (:). No other text is allowed.

4.2.4 Base Annotation

Base variables provide filters that apply to tables overall. Only titles, footnotes and summary row annotation are printed for bases, so text width is not needed and will be ignored if specified.

Base variables may specify annotation for the table summary rows to be printed instead of the default or LABELTEXT labels (see section 3.4). These are indicated by an asterisk (*) in the spacer column followed by a letter code identifying the applicable summary row. These are technically titles, so multiple lines are used to print multiline labels:

***T**	TT row - Table Total
***N**	NA row - No Answer
***A**	NA row - Total Answering
***M**	TM row - Total Mentions
***S**	Subtitle annotation printing immediately before the first detail row.

4.3 Ranking and Rank Levels

QTAB provides for ranking the rows of a table in descending order based on the contents of a specific column (see section 7.4.3). By default, all rows in the table are ranked in descending order, but it is often desirable to force certain rows to appear at the top or the bottom of the table, or to rank sections of a table and rows within those sections. For example, in responses to open-ended questions, one might want to see all the positive responses before the negative ones and also to rank individual responses within one or more levels of category nets.

The R parameter in a stub variable definition (see section 4.1.1) tells QTAB that the column after the spacer column is reserved for a rank level code that provides for nested ranking up to eight levels deep and further controls whether a row will be ranked or not within each level.

A stub variable must have exactly as many rank codes as there are rows defined, so no rank code should be assigned to continuation lines or subtitles (see section 4.2.2). When rows are reordered, subtitles move with the next physical row specified.

The rank level codes are 1 through 9, 0, A, B, C, D, E, F, where each pair of codes defines a level within which the odd numbered codes are ranked, and the even numbered codes are not. Thus, 1 is the code for first level, ranked and 2 the code for first level, unranked; 3 is the code for second level, ranked, and 4 the code for second level, unranked; E is the code for eighth level, ranked, and F the code for eight level, unranked.

Rank levels may be visualized as concentric groups of rows that are rearranged as a whole at the same level. Within each individual group, all the groups at the next level are rearranged, and so forth. At each level within a group, an even rank code terminates the ranking process so that that row, and all those following it within the group, will not be ranked. The process resumes with the next group at a higher level. This means that an odd numbered code can never follow an even numbered code of the same or higher level.

The following example illustrates the use of rank level codes (specs are omitted):

```
    VAR            L S40R16
                    1 1st level ranked         [01]
                    3   2nd level ranked       [02]
                    5     3rd level ranked     [03]
                    6     3rd level unranked   [04]
                    3   2nd level ranked       [05]
                    5     3rd level ranked     [06]
                    4   2nd level unranked     [07]
                    1 1st level ranked         [08]
                    3   2nd level ranked       [09]
                    3   2nd level ranked       [10]
                    2 2nd level ranked         [11]
```

In this example, if [08] is greater than [01] in the rank column, rows 8-10 would print as a group before rows 1-7, while row 11 would remain at the end of the table, regardless of its contents. Rows 9 and 10 will print in rank order following row 8. Within the group of rows 1-7, if [05] is greater than [02], rows 5-6 would print as a group before rows 2-4, and row 7 would appear at the end. Rows 3 and 4 will print in rank order following row 2.

5 THE QUIP SPECIFICATION LANGUAGE

Specifications describe the conditions that must be met for a case to be counted in a vector.

5.1 Data Specifications

When tabulating data, QTAB reads one record at a time into a work area and applies the specs to the contents, incrementing counts and cumulating events for vectors that meet the specified test conditions as it proceeds sequentially through the data file. This section describes how to use the QUIP specification language to define conditions and events directly.

5.1.1 Addressing Data

QTAB addresses data using absolute column locations within each record. The location may be specified directly as the number of columns offset from the beginning of each record or using the card/column notation favored by many statisticians and marketing researchers. Data may also be defined in character mode (e.g., ASCII), where each column occupies a single byte, or in column-binary mode, where each column occupies two adjacent bytes.

QTAB provides four addressing modes to handle the possible combinations (see section 3.1).

In address modes C and D, a column occupies a single byte, while in address modes B and E, a column occupies two bytes. Address modes C and E use direct offset notation, so in address mode C, column x is byte x of the record, while in address mode E, column x consists of the two consecutive bytes 2x-1 and 2x.

In card/column notation, the column number within the card is a two-digit number ranging from 01 to 80 and must be preceded by a card number. The first 80 columns of a record are written as 101 through 180, the next 80 columns as 201 through 280, and so on. The data in columns 180 and 201 would be addressed as columns 80 and 81 in direct offset notation.

QTAB interprets data locations according to the current address mode automatically. Any data type can be used in any address mode but even-numbered bytes cannot be addressed directly in column-binary address modes since addresses correspond to 2-byte column locations.

5.1.2 Fields and Numbers

A data address followed by one of the codes listed below will be treated as a field or a number. QTAB recognizes the following field or numeric codes (either upper or lower case):

An	Column number or field, 'n' digits or columns long, where column depends on address mode. An translates to Bn when the address mode is B or E, and to Cn when the address mode is C or D (see section 3.1).
Cn	Character number or field, 'n' digits or characters long
Bn	Column-binary number or field, 'n' digits or columns long
Xn	Extended column-binary number, 'n' digits long, where an 'X' punch repeated 'n' times represents 10 to the power n (e.g. for X2, 'XX' = 100).
H	Short, signed integer (sometimes called a 'halfword')
I	Short, unsigned integer
F	Long, signed integer (sometimes called a 'fullword')
G	Long, unsigned integer
Q	Single-byte, signed integer (sometimes called a 'quarterword')
R	Single-byte, unsigned integer
E	Single-precision floating point (occupies 4 bytes)
D	Double-precision floating point (occupies 8 bytes)
Pn	Packed decimal number 'n' digits long; 'n' is an odd number between 1 and 15

Fields (codes A, B, C, X) may be from 1 to 15 characters long for numbers or field operations. Column-binary fields (A, B, X) may be up to 80 columns long and character fields (A, C) may be up to 250 bytes long when testing for their being blank or non-blank.

Leading blanks are allowed in numeric fields, embedded blanks and non-numeric characters are not, except for a leading minus sign to indicate a negative value. For situations when numeric fields may contain non-numeric values, codes A, B, C and X may be followed by a suffix indicating how to handle special conditions: "B" indicates that blanks are to be given the value zero, and "E" indicates that any non-numeric value be given the value zero (e.g., 101be2 represents a 2 column column-binary field in which any non-numeric entry would translate to 0).

Numeric fields of data type A may have an additional suffix "X" that works like the data type X (extended column-binary) for either column-binary of ASCII data. In ASCII data, an X-punches are represented in data by minus signs is used in data for X-punches, so for a field defined as 21AEX3, an entry of "---" would translate to 1000, "--1" to zero, and "-10" to minus 10.

5.1.3 Columns and Punches

An address with no field code identifies a single column that contains "punch" data and may contain only the values that correspond to the punches in a column of a Hollerith (or IBM) card. The punches are 1 through 9, 0, X and Y, in that order. In ASCII modes, the X punch refers to a minus sign (-) and the Y punch to an ampersand (&) in the actual data.

In column-binary data, a column may contain multiple punches in any combination. This allows as many as 12 different codes to be stored in a single 2-byte column in column-binary data, as compared to the 12 bytes that would be required to store the same information as ASCII. For this reason, column-binary is often used for data sets containing many multiple response items.

5.1.4 Literals, Constants and Numeric Expressions

Literals are constant values used in comparisons and usually follow a logical operator. They are indicated in specs by single quotes. QTAB treats literals differently depending on the context in which they appear. In column tests, they are punch lists; in numeric tests, literals are numeric values and in field comparisons they are character strings. In some situations, multiple literals may be specified within a pair of single quotes (see section 5.2).

Constants are values used in numeric expressions and are indicated by prefixing a number by the letter "K", thus K132 represents the constant value 132.

Numeric expressions are built from numeric fields and constants using the four standard arithmetic operators: **+**, **-**, ***** and **/** for addition, subtraction, multiplication and division. Parentheses **( )** may be used to provide precedence among operators (up to 15 levels).

QTAB converts all numeric values internally to double-precision for evaluation or computation, so spec writers do not need to be concerned with mixing data types or multiplying values to preserve their precision.

5.2 Counting Cases

A case is counted by QTAB when specified conditions test true. Field and numeric tests require an operator. Punch tests apply to single columns and no operator is used. Complex conditions may be created by using Boolean operators to combine simple tests.

5.2.1 Field and Numeric Tests

The following operators test a field for the conditions indicated:

B	Test field for being Blank (blank means that the field contains all spaces for ASCII data, all nulls for column-binary data).
P	Test field for being Packable (the field contains a valid numeric value).
Q	Test the field for being either Packable or Blank.
U	Test the field for being Unpackable (the field is not a valid numeric value).

The following operators test the numeric value of a field against one or more number literals:

='nn[;mm]'	Equal to a numeric value. A maximum of 4 numeric values may be specified for the literal, separated by semi-colons.
G'nn'	Greater than or equal to a numeric value.
L'nn'	Less than or equal to a numeric value.
>'nn'	Greater than a numeric value.
<'nn'	Less than a numeric value.
R'nn;mm'	Falls within a numeric Range (from;to).

The following operator compares the contents of an ASCII field to one or more string literals:

@[-n]'mm'	Character string compare [optionally for "n" contiguous fields] (used with ASCII data only). Accepts up to 4 literals separated by semi-colons, all of which must be of the same length as the field being compared.

The following operators test the value of an ASCII field against the ASCII string value of literals (the relative value of an ASCII string is determined by its sort sequence):

@**G**'aa'	Greater than or equal to an ASCII string value.
@**L**'aa'	Less than or equal to an ASCII string value.
@**>**'aa'	Greater than an ASCII string value.
@**<**'aa'	Less than an ASCII string value.
@**R**'aa;bb'	Falls within a Range of ASCII string values (from; to).

The sense of all field and numeric tests may be reversed (negated) by the "Not" operator **N**. For field and numeric tests, the N is placed immediately in front of the operator. For the field compare test, the N is placed immediately before the literal, outside the quote. For string value tests, the N is placed between the @ sign and the comparison operator.

5.2.2 Punch Tests

Punch tests are specified by a column address followed immediately by a literal containing a list of punches to be tested for. The list of punches is called a punch mask and the test is satisfied if any of the punches in it are present in the column. A range of consecutive punches may be specified using a hyphen (-), so '1-Y' is equivalent to '1234567890XY' and '1-39-X' is the same as '12390X'. Note the order of punches in the mask (see section 5.1.3). Punch tests may test for the absence of punches by prefixing either the literal or the punch mask itself with an N.

For column-binary data only, a column may be tested for the number of punches it contains by using the tally operator:

T<op><n>'pp'	Where <op> is a numeric comparison operator (=,<,>), n is a number from 0 to 12 and 'pp' is a list of the punches to be counted in the tally.

5.2.3 Boolean Operators and Compound Tests

The following Boolean operators are used to combine conditions into compound tests:

&	Logical AND: The result is true if, and only if, both conditions are true. (Ampersand)
!	Logical OR: The result is true if either condition, or both, are true. (Exclamation point)
( )	Parentheses: The entire expression within a matching pair is treated as a single condition or value in a Boolean or arithmetic expression.
{ }	Braces: Allows an arithmetic expression to be used in a test as if it were a single numeric field in a test condition.

The maximum length of a QTAB specification is 500 bytes, and expressions may be nested in parentheses up to 15 levels deep, so very complicated conditions may be specified. There is no Boolean "not" operator, so compound expressions cannot be negated as a whole.

Any numeric field or arithmetic expression that is not generate a valid number will cause the entire specification in which it appears to immediately evaluate as false.

5.2.4 Examples of Test Conditions

The following examples illustrate data tests using the QUIP specification language. Note that tests on single column fields and punch tests can almost always be used interchangeably, but the punch tests are often easier to specify and are more readable:

`11'125'`	Test for punches 1, 2 or 5 in column 11
`11C1R'1;2'!11C1='5'`	Same result, using value tests
`11C1R'1;5'&11N'3;4'`	Same result, different logic
`21C5='125'`	Test a 5 byte field for the numeric value 125 (the test will be true for 125, 0125, 00125)
`21C5@'00125'`	Test a 5 byte field for the string "00125" (the test will be false for " 125" or " 0125")
`21C3B`	Test for a 3-column field being blank
`21C3P`	Test for a valid number in a 3-column field
`101'1-8'`	Test for the presence of punches 1 through 8 in column (any combination will satisfy this in column-binary data)
`101N'1-8'`	Test for the absence of punches 1 through 8 in column (any punches other than 1-8 may be present)
`101'N1-8'`	Same as the preceding
`101T=1'1-8'`	Test for the presence of exactly 1 punch in the range 1-8 in column (column-binary data only)
`101T>2'1-8'`	Test for the presence of more than 2 punches in the range 1-8 in column (column-binary data only)
`{21C3+24C3}='100'`	Test that the sum of the values in the 3-column fields beginning in 21 and 24 is equal to 100

`{21CB3+24CB3}='100'`	Same as above, but set the value to a blank field to zero so that it does not cause the entire specification to fail.
`21C3@-3'100'`	Test for the character string "100" beginning in columns 21, 24 or 27
`21C1@R'A;Z'`	Test for any uppercase letter in a 1 column field
`121X2='100'`	Test for a value of 100 in a 2 column field in column-binary data (represented by an X punch in both columns)
`121AX2='100'`	Same as above, but for either column-binary or ASCII data, depending on the address mode

5.3 Weights

A weight is a numeric value assigned to each record in a data set that is used as a multiplier whenever that record is counted and is used for such purposes as to adjust counts to known population characteristics or to tabulate usage rather than cases.

In addition to volumetrics, which are weights applied to individual table vectors (see section 6.1), QTAB allows table level weights, defined in a WEIGHT statement and applied to tables by a table definition option (see section 7.4). The statement begins with the keyword WEIGHT or WT, followed by a name of up to 8 characters and a valid numeric field, constant or numeric expression (see section 5.1.4):

WEIGHT <name> <expression>

A weight value specified as a character or column-binary numeric field can be adjusted for implicit decimal places by adding a period and the number of decimals by which to adjust the value entered in the field. This is equivalent to dividing the value by a constant of ten to the desired power.

QTAB will stop evaluating an expression if any part does not generate a valid numeric value in the data. In a weight definition, this will cause the value to be set to zero. Weights that evaluate to zero will cause a warning message to be displayed on screen and in the spec listing unless the NOWTMSG run option is invoked (see section 3.2), and a count of zero weights will be shown in the run statistics at the end of the listing.

Some examples of weight definitions:

WEIGHT adults 171CE3.2	The weight is the value found in the 3 character field beginning in 171 read as having 2 implied decimal places, and with invalid values set to zero
WEIGHT wght 171ce3/k100	Defines the same weight as above
WT K k20	The weight is a constant value of 20

6 SPECIFYING ROWS AND COLUMNS

The left-hand side of each line in a variable definition is reserved for the specifications (specs) that determine who and what will be counted in the cells of the variable. In stub variables, the specs define the row vectors and in header variables, column vectors. At a minimum, every cell in a table contains the count of cases that meet the conditions specified by the row and column vectors that intersect at that location.

Each individual specification begins in column 1 of a line and may extend up to, but not into, the spacer column. Specifications too long to fit in the available space may be broken and wrapped to the next line with a backslash (\). Embedded blanks and tabs are not allowed.

Specs may be preceded by a tag by which the vector may be referenced in nets and subtotals (see section 6.3) and by punctuation codes that affect how the cells will print (see section 6.6).

Base variable specifications consist of a single condition that applies as a filter to all the cells of a table, so they are not considered in the remainder of this section.

6.1 Volumetrics

In addition to the specification of *who* will be counted, each individual vector of a stub or header variable may specify a numeric value that determines *what* will be counted, called a volumetric or event. The volumetric is specified after the test condition for a vector by a numeric field or expression following a semi-colon (;).

When volumetrics are used, QTAB can tabulate both case counts and the event totals for each cell of a table simultaneously. If a weight is also applied to the table, as many as four separate counts may be tabulated for each cell: weighted and unweighted case counts and weighted and unweighted event totals (see section 7.3).

The following examples illustrate volumetric definitions:

`121C2P;121C2`	Cumulate the values in 121C2 among those cases that have a valid numeric value in that field.
`;121C2*121C2`	Sum the squares of the values in 121C2 for all cases.
`;(12CE1+13CE1)/K2`	Sum the means of the values in 12C1 and 13C1, with non-responses counted as zero values, for all cases.

6.2 Generated Vectors

QTAB provides several shorthand methods for generating a number of row or column vectors with a single spec.

In stub and header variables, in a comparison test against single values, strings or punch masks, the literal definition may contain a list of items separated by commas. Each item in the list generates a separate row or column, as if a series of consecutive specifications had been written, one for each item in the original list.

A list consisting of consecutive integers or punches may be generated even more succinctly by indicating the range to be tested by means of a colon (:) in the literal definition.

If a volumetric is specified after a test using a literal list, that event is cumulated for each of the vectors generated by the list. Vectors must be specified on separate lines if different events are needed.

Boolean operators should never be used with lists of literals as the results are unpredictable. The following examples illustrate lists of literals.

`101'1,2,3,4,5,6,7,8'`	Generate 8 vectors: the 1st tests for punch 1, the 2nd for punch 2,...the 8th for punch 8
`101'1:8'`	Same as the preceding, specified as a range of punches
`101'1:8,N1-8'`	Generate 9 vectors: the first 8 as above, the 9th vector will count all cases not included in the first 8
`101C1='1:8'`	Generate 8 vectors: the 1st tests for value 1, the 2nd for value 2,...the 8th for value 8
`101C2>'1:8'`	Generate 8 vectors: the 1st tests the value in a 2 character numeric field for being greater than 1, the second for being greater than 2,... the 8th for being greater than 8
`101C1@-4'A,B'`	Generate 2 vectors: the 1st tests for the character "A" in columns 101 through 104, the 2nd for "B"

6.3 Value, Range and Array Field Operators

QTAB provides three special operators that may be used in stub variables only to generate an entire series of row vectors based on the contents of a numeric field or expression, optionally with a set of associated labels.

The syntax and usage of the three operators are similar. Each must appear before any other specs (except for a tag) on the line. When any of these operators is used to specify rows in a variable, the associated label function may be used to generate labels for those rows. If the full field operator definition is followed by a specification for a condition test or event, these will be applied to every row generated by the field operator.

Each specification begins with the field operator followed by the definition of the source of the distribution and the scope of the distribution as described in the individual descriptions below. When used to generate associated labels, the same keyword is used in the text part of the line, with the # sign in the space column. Labels may optionally be right justified at a specified margin from the left edge of the table stubs.

6.3.1 The Value Field Operator

The value field operator **#V** generates a distribution of integer numbers in ascending sequence and, optionally, the corresponding labels. The format is:

#V<field>'pp;qq' **[#V'nn[,mm]']**

Where pp and qq are the beginning and end of the distribution range on the field, nn is the number of rows generated and mm the right margin for label text. Range Field Operator

6.3.2 The Range Field Operator

The Range field operator **#R** generates a distribution on consecutive ascending numeric ranges and, optionally, the corresponding labels. The format is:

#R<field>'a;b;c;...n' **[#R'nn[[,mm],dd]']**

Where a, b, c,... are the lower bounds of the ranges against which the field will be tested, and the last number is the upper bound of the last range (the number of ranges defined is one less than there are number in the list), nn is the number of rows generated, mm the right margin for label text and dd the number of decimal places to be used in the label text.

6.3.3 The Array Field Operator

The Array field operator **#A** generates a frequency distribution and its associated labels based on the values found in a numeric field in the data. It is the only QTAB operator for which values to be tabulated are not specified explicitly. The operator is followed by the field and a literal that reserves a number of rows sufficient to hold the array of unique values in the distribution. If the number of unique values found exceeds the number of rows reserved, new values encountered in the data will be lost, but values matching existing cells will continue to be tabulated. The array is sorted and printed in ascending sequence. The format is:

#A<field>'nn' **#A'nn[[,mm],dd]'**

Where nn is the number of rows generated, mm the right margin for label text and dd the number of decimal places to be used in the label text. Generated labels must be used with array fields since the values are not known in advance. The distribution is built in memory as the data are tabulated and cannot be saved for a recompute run (see section 2.2).

```
#A111b3'100'111b3r'101;200';114b2      #A'100,10,1'
```

In this example, each cell generated by the #A operator is filtered by the test: 111b3r'101;200' and weighted by the value in the field 114b2. The array reserves 100 rows for values. Labels will print at an indent of 10 spaces from the left margin and will show values to 1 decimal place.

6.4 Nets and Subtotals

Nets and subtotals provide a means for defining rows and columns as logical combinations or arithmetical expressions of other rows or columns within the same variable. Both nets and subtotals ordinarily reference other vectors by their position counting from the first detail row or column defined in the variable.

6.4.1 Tags

Tags provide a means by which vectors can be referenced directly in nets and subtotals, as well as in post processing instructions (see section 8). Tags are 2 character labels surrounded by underscores (e.g., _AA_). Case is significant in tags, so _AA_ is not the same as _aa_, but tags to be referenced in post processing instructions should only use uppercase or numbers.

6.4.2 Nets

A Net is a row or column derived from a list of other vectors in the same variable using Boolean logic. Nets are counted as the data is tabulated and their counts are also cumulated into the Total Mentions row or column. Nets apply logic to all cells of a vector including events which are effective treated as counts rather than volumetrics.

A net definition begins with the keyword NET or NEN, followed within single quotes by a list of vectors separated by semicolons (;). A range of consecutive vectors is specified by a period (.).

The following describe the net instructions:

NET'<list>'	Count those cases that were counted in one or more of the vectors listed
NEN'<list>'	Count those cases that were counted in ALL of the vectors listed
ALL	Forces a count of all cases, whether or not they are counted elsewhere.

6.4.3 Subtotals

Subtotals are vectors that are computed arithmetically from the contents of other vectors after the data has been tabulated and before any percentaging, post processing or print formatting is performed on tables. They are not counted as the data is tabulated and so are not cumulated into the Total Mentions row or column. The arithmetic is performed on all cells, so the events cell of a subtotal is derived using the same formula as the counts cell.

A Subtotal begins with **ST** followed by an optional pass level in quotes, then an expression combining other vectors from the same variable and constants, using the four standard arithmetic operators: **+**, **-**, ***** and **/** for addition, subtraction, multiplication and division. Parentheses **( )** may be used to provide precedence among operators (up to 15 levels). A range of consecutive vectors may be summed by using a period (e.g., ST01.10). Subtotals may also reference other subtotals and the summary vectors for Total, No Answer, Total Answering and Total Mentions.

Unless pass levels are specified explicitly, QTAB computes all the subtotals for a vector in the order in which they appear, in a single pass. Multiple computation passes may be specified by a pass number in single quotes after the ST and before the expression. Multiple passes should be specified when any subtotal references another to force the correct computation order.

By default, QTAB computes the row subtotals before the column subtotals for a table, although that order can be reversed (see section 7.4). When both the stub and header variables for a table contain subtotals, pass levels should be specified to make sure that results are correct.

6.4.4 Examples of Nets and Subtotals

The following illustrate the use of nets and subtotals:

`NET'1;3;5;7'`	Net vectors 1,3,5 and 7
`_D1_NET'_A1_._C1_'`	Net vectors labeled _A1_ through _C1_
`ST1+3+5+7`	Add vectors 1,3,5 and 7
`_LL_ST(_KK_*K100)/TA`	Multiply vector labeled _KK_ by 100 and divide the result by the Total Answering vector
`_MM_ST'2'_LL_*_LL_`	Multiply vector labeled _LL_ by itself on the second pass (this will force a second subtotal computation pass)
`STK0`	A constant value zero (creates an empty vector)
`STNA`	Get the contents of the No Answer vector

The last example is often the easiest way to create a No Answer row or column.

6.4.5 Special Purpose Subtotals

A few special purpose subtotals have very limited applications but provide a solution to some esoteric problems:

Array maximum and minimum subtotals: **STAHI**<xx> and **STALO**<xx>, where <xx> is the initial vector of an array on a numeric field specified using the #A operator (see section 6.3.3). QTAB will enter in the F-cell the highest or lowest array value that was counted in each column. Note that this is the automatically generated stub label, not the count for the value.

Frequency maximum and minimum subtotals: STFHI<xx;yy> and STFLO<xx;yy>, where xx and yy are two row vectors. QTAB will enter the highest or lowest frequency found in any cells in each column between rows xx and yy, inclusive.

6.5 Statistical Subtotals

Basic statistics, including weighted means and medians, may be computed from the rows and columns of a table. The cell weights (for means and related statistics) or ranges (for medians) may be defined using numbersets or obtained from the data.

6.5.1 Numbersets

A numberset is a static vector consisting of a list of constant numeric values, Numbersets are defined outside of variables and are used in post processing instructions (see section 8) as well as to provide category weights or ranges for statistical distributions.

A numberset is indicated by the keyword NS in column 1, a name, and a series of numbers separated by blanks. An X may be used instead of a value to indicate a row or columns to be excluded from computations in statistical subtotals. A backslash (\) may be used to indicate that the list of numbers continues on the following line.

NS Scale1 1 2 3 4 5 6 7 8 9 10

NS Scale2   -5 X -3.33 -1 0 1 3.33   X   5

6.5.2 Statistical Subtotals Using Numbersets

Statistical subtotals may be used to compute statistics for a distribution using a numberset to supply the category weights for mean, variance, standard deviation, and standard error. Medians may also be computed using a numberset to define the lower limit of the ranges that will be used to compute the median in the cell containing the 50th percentile.

Statistical subtotals on distributions begin with the ST keyword, followed optionally by a pass level (see section 6.4.3), then the type of statistic, the range of the distribution identified by the first and last vectors separated by a semi-colon (;), and the name of the numberset providing the category weights or ranges.

The format for statistical subtotal on distributions using numbersets is:

ST['n']MNaa;zzNSabc	Mean on the distribution in vectors aa through zz, with category weights from the numberset named "abc" and optionally computed in subtotal pass n

The statistics that may be computed this way are:

MN	Mean
VR	Variance
SD	Standard Deviation
SE	Standard Error of the mean
MD	Median (interpolated from base of 50th percentile range)
DMD	Discrete Median (base value of 50th percentile range)

For means, variances, standard deviations and standard errors, the numberset named must have at least as entries as there are vectors in the distribution. For medians, there should be at least one more entry to define the top of the last range. Additional entries are ignored.

The following examples illustrate statistical subtotals on distributions:

`STMN1;20NSwgts`	Mean on rows (or columns) 1-20 weighted by values in numberset "wgts"
`STSE1;20NSwgts`	Standard Error of mean on same vectors
`STMD1;10NSbounds`	Median on rows (or columns) 1-10 using the ranges bounded by the numbers in numberset "bounds"

6.5.3 Extended Statistical Subtotals

The Mean, Variance, Standard Deviation or Standard Error of the distribution of values in a field or expression may be computed directly from the data by using an extended statistical subtotal that points to a vector specifying the base count for the statistic. The vector immediately after the base count vector defines the sum of values (sum of X) as an event, and for VR, SD and SE, a third vector must follow defining the sum of squared values as an event.

STXMNaa	Mean using the condition defined in vector aa as base and the contents of the event cell of vector aa+1 for the sum of values
STXVRaa	Variance using the condition defined in vector aa as base, the contents of the event cell of vector aa+1 for the sum of values and the contents of the event cell of vector aa+2 for the sum of squares
STXSDaa	Standard Deviation using the same vector information as above
STXSEaa	Standard Error of the mean using the same vector information as above

QTAB applies the algorithm without checking the vector contents, so it is up to the spec writer to make sure that these are correct.

The following example illustrates the use of STX for these statistics (note that all three work rows specify the same condition, otherwise the two sums might include extraneous values):

```
STXMN_AA_                 Mean (Base in _AA_, sum in _AA_+1)
STXVR_AA_                 Variance using same values
STXSD_AA_                 Standard Deviation using same values
STXSE_AA_                 Standard Error using same values
_AA_141c2r'0;90'         DBase for mean (N = count of X)
141c2r'0;90';141c2       DSum of X
141c2r'0;90';141c2*141c2  DSum of squares (needed for SE, SD, VR)
```

Medians may be computed using the distributions generated by the value, range or array field operators (see section 6.3) with an **STXMD** extended subtotal definition. Four versions provide options as to how to interpolate the median value:

STXMDBaa	Median of a distribution beginning in vector aa and interpolated using the lower bound of the 50th percentile cell as the Bottom of range
STXMDTaa	Median of a distribution beginning in vector aa and interpolated using the lower bound of the 50th percentile cell as the Top of range
STXMDMaa	Median of a distribution beginning in vector aa and interpolated on the range defined by the midpoints of the cells immediately above and below the 50th percentile cell
STXMDDaa	Discrete median of a distribution beginning in vector aa: the lower bound of the 50th percentile cell is used as the median value

The following example illustrates use of medians for a generated array of up to 100 values:

```
_AA_#A111c2'100'    #A'100,12,0'
F2DKAKSTXMDB_AA_    BMedian (Bottom)
F2DKAKSTXMDT_AA_     Median (Top)
F2DKAKSTXMDM_AA_     Median (Midpoint)
F2DKAKSTXMDD_AA_     Median (Discrete)
```

Other QTAB extended statistical subtotals compute certain specialized functions such as the "Reach" measurement used in media research. These are beyond the scope of this guide.

6.6 Specifying Cell Punctuation in Variable Specifications

The PCELLS keyword is used in table definitions (see section 7.4) to indicate which cells to print and the punctuation to use for them. This can be overridden for individual rows or columns by specifying punctuation codes before the vector specifications in variables. The cells to which the codes apply are specified using the names listed in section 7.3, followed by the punctuation code and the number of decimal places desired.

The punctuation codes used in are:

n	Show n decimal places, where n can be from 0 to 3.
.Cn	Use commas to punctuate numbers 1,000 or larger.
.Mn	Toggle % signs on/off (normally only shown for whole percents).
.Pn	Put parentheses around numbers.
.+n	Add leading signs to all numbers (not just negatives).
.$n	Add leading dollar signs.
K	Kill (suppress) cell contents from printing. No decimal places should be specified

Some examples of cell punctuation strings in variables:

`F2DK`	Print F-cell to 2 decimal places, kill Down percents.
`F.C2A.M0`	F-Cell with commas and 2 decimals, Across percents with no decimals and percent signs (%) toggled off.

Any cell punctuation may be modified in post processing instructions (see section 8), but any cells suppressed by the K punctuation code in the variable definition cannot be shown at all.

7 TABLE DEFINITIONS

The last section of a QTAB spec file, after all variables and other elements have been defined, contains the table definitions that instruct QTAB how to construct tables from the variables and how to format the output. The beginning of the table definition section is indicated by a TABLE statement (a line with the keyword TABLE starting in column 1 and no other text) following an END statement. The table definition section ends with another END statement.

At the very minimum, a table name, a stub variable and a header variable are required to define a table. One or two base variables may also be specified. Options are specified by keywords on the table definition line and table titles may also be attached directly to the table definitions.

7.1 The Table Definition Line

A table definition line consists of a single line divided into two parts: The first 42 columns are reserved for the names of the table and of the component variables, as shown in the list below. The remainder of the line, from columns 43 on, lists the table options indicated by keywords and the associated parameters. A backslash at the end of any line indicates that the list of options continues onto the next line in the spec file. Text in continuation lines may be in any column.

The required format for a table definition is:

Columns	Contents
1 - 8	Table name (may include blanks)
9 - 10	Table type (for derived tables only, see section 7.5)
11 - 18	Stub variable name, left justified
19 - 26	Header variable name, left justified
27 - 34	First base variable name (optional), left justified
35 - 42	Second base variable name (optional), left justified
43 +	Keyword options, in any order. Use a backslash (\) to continue options on one or more following lines.

Up to 10 lines of title text for a table may be entered immediately after the table definition line. On table title lines, columns 1-8 must match the name on the table definition line, and all text from columns 9 on is title text. Titles specified with table definitions print along with those from the variable definitions. This may be controlled through the TITLES keyword (see section 7.4).

Table names print in the report and table of contents exactly as entered within the 8-character field on the table definition line. Table names are case sensitive (1A and 1a are not the same) and may include embedded or leading blanks, except when the table will be referenced later by a derived table (see section 7.5).

Table names do not have to be unique, but consecutive tables cannot have the same names, since the second table line would be interpreted as title text for the first. Duplicate names will also cause problems with derived tables (see section 7.5) and recomputes (see section 2.2).

7.2 Dupe and Merge Lines

QTAB provides two template forms for table definition lines called DUPE and MERGE lines.

DUPE and MERGE lines are written exactly the same way as a regular table definition line, but do not actually define a table. Rather, all variables and options specified in the template will be carried into every following table definition lines until the next template line of the same type or the end of the spec file is reached.

A DUPE line always overrides the corresponding variable or options in the table definitions that come after it, whereas those specified in a MERGE line are overridden by the contents of the subsequent table definitions.

DUPE and MERGE streams are independent of each other and both may be used at the same time to specify different instructions.

7.3 Table Cells and Contents

QTAB can display as many as eight different data cells simultaneously for each table cell. Except for the frequency (F) cell, none of the cells contains data unless the appropriate keyword is used in the table options (see section 7.4). Whether or not they contain data, all cells may be printed and may be accessed using post processing instructions (see section 8).

Four cells contain tabulated frequencies and weighted counts obtained directly from data, the remaining four cells contain percentages base on the contents of the frequency cells.

In addition, a text cell, accessible only through post processing (see section 8) may be used to print text characters below the data cells for such purposes as flagging significance tests.

7.3.1 Tabulated Data Cells

The F (frequency) cell contains the ultimate result of the counts generated by the combination of specifications in the variable definition and table weighting options. If called for, the U, W and E cells contain the intermediate counts obtained in the process of tabulating the frequency cell.

The four cells tabulated from the data are:

F Frequency cell. The contents of this cell depend on the combination of table weighting and vector specifications as per the following table:

Table weighting/vector specification	Contents of F cell
Unweighted, no event specified:	Case count
Unweighted, event specified:	Volumetric sum
Weighted, no event specified:	Weighted case count
Weighted, event specified:	Weighted volumetric sum

U Unweighted case counts for volumetrics or weighted tabulations

W Weighted case counts for weighted volumetric tabulations

E Unweighted volumetric counts for weighted volumetric tabulations

The F cell is always tabulated and prints by default. The U, W and E cells are only tabulated when invoked with the TCELLS keyword, and only printed if also invoked with the PCELLS keyword in the table options (see section 7.4). If not tabulated, the U, W, and E will not be written to any compute file and will not contain data in a recompute run (see section 2.2).

7.3.2 Percentage Cells

QTAB provides four additional cells that are normally used to display percentages computed after the data have been tabulated, based on the contents of the F cells. Percentages are both computed and printed when requested after the PCELLS keyword; they are computed without being printed when called for after the COMP keyword.

The four percentage cells are:

D	Percent Down (Column percents), default base: TT row
A	Percent Across (Row percents), default base: TT column
C	Corner percents (Entire table is percentaged on the contents of the corner cell), default base: TT corner
I	Index (Ratio of % down in each column to % down in base column) , default base

The PCELLS or COMP keyword may specify a different base for percentage cells, and PCELLS is also used to indicate the number of decimal places to print (see section 7.4).

7.4 Table Options

Table options are invoked in the table definition by keywords, followed in most cases by one or more parameters. The table options keywords are:

COMP	Compute percentages without printing them.
DErived	Derive this table from the existing tables named (see section 7.5).
FLAG n,nn	For weighted tables, print 1 asterisk (*) next to the f-cell if the u cell is less than n, 2 asterisks (**) if the u-cell is less than nn.
FOrce	Force percents to 100% if frequencies add to base.
FPrint	Show frequency cell for bases on percent-only tables.
PCells	Specify cells to print, punctuation and percent bases.
POst	Apply post-processing instructions in order listed (default B1,B2,H,S,T).
RAnk	Rank the rows on the contents of the F-cells of a specified column.
SIngle	Suppress the blank lines between rows of a table.
STAR 'x'	Put 'x' in non-zero cells rounding to zero (default '*').
STats	Compute and print statistical summary rows/columns.
SUBtotal	Specify subtotal computation order: R (default) or C first.
SUMmary	Specify table summary rows/columns to print.
TCells	Specify cells to be tabulated from the data.
TItles	Use table titles and footnotes from sources listed (default B1,B2,H,S,T).
WT	Weight table by values in named weight definition.
ZCELLS 'x'	Put the character 'x' in zero cells (default '-').
ZErosup	Suppress rows with no data from printing.

Except for FORCE, FPRINT, SINGLE and ZEROSUP, all of these keywords must be followed by the appropriate parameters.

7.4.1 The TCELLS, PCELLS and COMP Keywords

Only the F cell is tabulated and printed by default. All other cells are generated by means of the TCELLS and PCELLS or COMP keywords and printed by means of the PCELLS keyword.

The TCELLS keyword tells QTAB which cells to tabulate from the raw data, using as parameter a list of tabulated data cells as described in section 7.3.1 and separated by commas.

PCELLS forces the computation and specifies the base for percentage cells (see section 7.3.2). It also tells QTAB which cells to print and specifies their punctuation. The parameter is a list of cells with punctuation codes, separated by commas, in the order in which they should appear in the output. Cell punctuation codes are of the form: C[B].[P]D, where C is the cell code as per section 7.3, B an optional base row or column for percentage cells, P a punctuation code and D the number of decimal places (0-3) to display. Data cells (except F) tabulated from the raw data must also be specified with the TCELLS keyword.

The following punctuation commands may be used in PCELLS arguments:

n	Print numbers to n decimal places, where n can be 0, 1, 2 or 3.
Cn	Use commas to punctuate numbers 1,000 or larger.
Mn	Toggle % signs on/off (normally only shown for whole percents).
Pn	Put parentheses around numbers.
+n	Add leading signs to all numbers (not just negatives).
$n	Add leading dollar signs.

The COMP keyword is used to force computation of percentage cells that may be needed for post processing but should not be printed and takes as a parameter the list of percentage cells to compute and their bases, separated by commas, with no punctuation codes.

The following example prints integer frequencies, column (down) percents to 2 decimal places based on total answering, and unweighted counts in parentheses with no decimals shown. It also computes, but does not print, row (across) percents based on the total column.

```
PCELLS  F.0,DTA.1,U.P0   TCELLS F,U   COMP ATT
```

7.4.2 The POST Keyword

The POST keyword specifies the order in which post processing (see section 8) instruction sets should be executed for a table, using the following codes separated by commas to describe the possible sources:

B1	Base variable specified in first base field
B2	Base variable specified in second base field
H	Header variable
S	Stub variable
T	Table options

If no POST keyword is given, all applicable post processing sets will be executed in the default sequence B1, B2, H, S, T.

If a POST keyword is given, only the sources listed will be used and the order of execution will be that in which they are listed (e.g., PO S,B1,T).

7.4.3 The RANK Keyword

The RANK keyword, optionally followed be a column identifier, causes the rows of a table to be printed in descending order as ranked on the contents of the F-cells of the column named, or of the TT summary column if no column is specified. If the stub variable for the table was defined with the R parameter as using rank level codes, these will be applied during the ranking process (see section 4.3 for information on using rank levels).

It is possible to rank a table on the contents of other cells, or in other than descending order, by loading the values desired into the F-cell of an unused column (typically a summary column) using post processing (see section 8), and specifying that as the column to rank on.

7.4.4 The SUMMARY Keyword

The SUMMARY keyword is followed by a list of vector names prefixed by "R" or "C" to indicate row or column and separated by commas. See section 3.4 for default vector labels and how to change them. The available summary vector print options are:

Name	Description	Prints at:	Labeling options
TT	Total	Beginning	LABELTEXT, stub variables
NA	No Answer	Beginning	LABELTEXT, stub variables
TA	Total Answering	Beginning	LABELTEXT, stub variables
TM	Total Mentions	Beginning	LABELTEXT, stub variables
TU	Total - Unweighted cells	Beginning	LABELTEXT
MB	Total Mentions	End	LABELTEXT
TB	Total	End	None - uses TT label
TE	Total - Unweighted cells	End	None - uses TU label

The following example prints the total, no answer and total answering rows at the top of the table, the total mentions and unweighted total at the bottom, the total column at the left and the total mentions column at the right:

```
SUM RTT,RNA,RTA,RMB,RTE,CTT,CMB
```

7.4.5 The STATS Keyword

The STATS keyword allows means and medians to be computed on the detail rows or columns of a table using weights supplied by a numberset (see section 6.5.1). The statistics are stored in summary vectors outside the specified table area and printed at the end of the detail vectors. In most cases, statistical subtotals (see section 6.5) provide far greater flexibility than statistics computed using the STATS option and should be used instead.

The STATS keyword takes as its parameter a list of statistical vectors with the numberset name to use after an equal sign (=), and separated by semicolons (;). When a mean is requested, the standard deviation, standard error and variance can also be displayed by entering their vector names separated by commas (,) before the equal sign.

By default, all statistics print to two decimal places, but this can be changed by appending the number of decimals desired (for 0 to 3) to the vector name with a period. A discreet median may be specified by using the name DMD for the median vector.

The example below prints the row mean, standard deviation, standard error and variance using numberset AVGWTS, the row median to one decimal place using numberset RANGE, and a discreet column median with no decimals using the numberset CRANGE.

```
STATS RMN,SD,SE,VR=AVGWTS;RMD.1=RANGE;CDMD.0=CRANGE
```

7.4.6 The TITLES Keyword

The TITLES keyword specifies the order in which table title lines will be printed in a table, using the following codes separated by commas to describe the possible sources:

B1	Base variable specified in first base field
B2	Base variable specified in second base field
H	Header variable
S	Stub variable
T	Table options

If no TITLE keyword is given, all title lines will print in the default order B1, B2, H, S, T.

If a TITLE keyword is given, only the titles from the sources listed will be printed and the order will be that in which the sources are listed (e.g., TI S,H,B2).

7.5 Derived Tables

A derived table is created by the cell by cell addition. subtraction, multiplication or division of other component tables previously defined in the same spec file. A derived table is defined by the table option keyword DERIVED, followed immediately by the names of the component tables separated by a comma. The method of derivation must also be entered in columns 9-10 of the table definition line.

A derived table has the dimensions of its component tables, which must be the same.   Except for subtotals (See section 6.4.3), the contents of each cell are derived from the corresponding cells in the component variables, so all other specifications in stub, header and base variables are ignored and only the annotation is used.

Derived tables are created before subtotals are computed in a QTAB run. This guarantees that subtotals defined in the stub or header variables will be correctly computed in the derived table, but it also means that subtotal rows and columns in component tables will not contain any data if they are not also defined as subtotals in the derived table.

Derived tables may be used as components of other derived tables defined later in the spec file.

The different types of derived tables are as follows:

A	Add the first table to the second table
S	Sum of all tables from the first through the second table
M	Subtract the second table from the first table
Pn	Multiply the first table by the second table, dividing by 10**n (n=0...9)
Qn	Divide the first table by the second table, multiplying by 10**n (n=0...9)
R	Repeat the table named (takes a single component name)

The following illustrates the use of derived tables:

```
TABLE
A1        Stub1   Header
A2        Stub2   Header
A3     A  Stub2   Header                   DERIVED  A1,A2
A4     S  Stub2   Header                   DERIVED  A1,A3
A5     Q0 Stub2   Header                   DERIVED  A3,A1
A6     R  Stub2   Header                   DERIVED  A1
```

In this example, table A3 is the sum of A1 plus A2, A4 is the sum of A1, A2 and A3, A4 is table A3 divided by A1 (multiplied by 10**0 = 1). Table A6 is the same as table A1.

7.6 Line Tabulations and Spreads

Line tabulations are tables that show multiple variables side by side, rather than cross-tabulating two variables. One variable is specified, and an INCREMENT that lists the offsets in the data file for each new variable to be tabulated. The offset applies to all elements of the variable. To permit individual filters for each variable in the line-tabulation, another variable may be mapped against the increment cell to cell, or against the incremented variable. This variable used as a list of filters is called a SPREAD.

Keywords in the table definition indicated the direction and name of increments and spreads: INCH increments a header variable, and INCS a stub variable. SPREADH applies a header variable as a spread to the columns of a table, SPREADS applies a stub variable to the rows.

The following example illustrates line tabulations and spreads:

```
INCREMENT atts 0 1 1
VAR rate  LABEL H08N16
175'1:5'        Excel.  V.Good    Good    Fair    Poor
VAR list  LABEL S24,11
ALL       Col. 175
ALL       Col. 176
ALL       Col. 177
VAR filt  LABEL S24
174'1,2,3'
TABLE
LT-1      list    rate                  INCH atts
LT-2      list    rate                  INCH atts  SPREADS filt
```

Table LT-1 will tabulate the data coded in column 175 in the first row, 176 in the second row, and 177 in the third row. Table LT-2 will have the same contents, but row 1 will be filtered on 174'1', row 2 on 174'2' and row 3 on 174'3'. In general, any variable can be used as a spread against another of the same direction (stub or header) and size.

8 POST PROCESSING

Post processing is one of the most powerful features of QTAB. Post processing instructions allow individual rows, columns or cells of any table to be manipulated arithmetically after the data have been tabulated and percentages computed. Post processing can also be used to alter the print format of cells, generate ranking information and load characters into text cells.

Post processing can be used to perform significance testing on the rows or columns of a table, and QTAB is distributed with libraries of post processing instructions to perform such tests that can easily be modified by sophisticated users to meet their own statistical requirements.

In general, post processing instructions work on specific cells of row or column vectors and the results are placed in specific cells of the same or other vectors of the same type. Restrictions can be placed on the range of operations so that, say, an instruction will only affect row r from column c1 to column c2.

Numbersets may be used in post processing instructions, both as input and as output, so they may be used to carry rows or columns of data to subsequent tables in a run. Constants may be used as input to numeric computations.

8.1 Post Processing Sets

Post processing instructions are grouped in sets identified by a PP statement, beginning with the keyword PP in column 1 followed, after one or more spaces by a unique name up to eight characters long. Each line after the PP statement contains a single instruction using a fixed position format. The set ends with an implicit or explicit END statement.

The PP keyword is also used in VAR statements to attach post processing sets to variables (see section 4.1) and in table definitions to attach a set directly (see section 7.4). Up to ten sets may be attached to each variable and table definition, so up to 60 sets may be executed for any table. The PP sets attached to any variable or table are executed in the order listed, and the order between components of the table is set by the POST keyword (see section 7.4.2).

In the following example, the order of execution will be: set3, set1, set2.

```
VAR  test      LABEL S24,30   PP set1,set2
...
TABLE
1        test    banner               POST T,S  PP set3
```

8.2 Post Processing Operands and Results

An operand in a post processing instruction can be a cell definition, a numberset , a constant (for computations) or a literal (for text manipulations). The result of a post processing instruction can be a cell, a cell mask (see section 8.2.3) or a numberset.

A numberset used in a post processing instruction must have been previously defined in the spec file, even though any values originally defined for the numberset will be replaced for all subsequent tables when it is used as the result of a post processing instruction.

Cell and cell mask definitions must specify both a vector name and a cell name, they may also have a loop increment if looping is called for (see section 8.3.3).

8.2.1 Vector Names

Post processing instructions reference row or column vectors using the 2-character vector name for built-in summary vectors and by vector number or for detail (user specified) rows or columns. Tags assigned in variable definitions (see section 6.4.1) may also be used to identify detail rows or columns. Tags cannot be checked during compilation, so if tags are used in post processing without being assigned in a variable, this will cause a run-time error.

The vector names are shown below, with the summary vectors (VR - TM) listed in the order in which they occur in the table matrix:

Vector	Contents
VR	Variance
MN	Mean
SD	Standard Deviation
SE	Standard Error
MD	Median
CH	Chi-Square
TT	Table Total
NA	Total No Answers
TA	Total Answers
TM	Total Mentions
1,2...	First, second, etc... user-defined row or column
_**XX**_	User defined row or column tagged with label _XX_ in variable definition

8.2.2 Cell Names

All eight data cells are available for use in post processing instructions, regardless of whether they have been invoked through TCELLS or PCELLS (see section 7.3). The text cell is always initially empty and can only be loaded through post processing instructions.

Cell	Contents
F	Frequency count (see section 7.3.1 for the actual contents of this cell in different situations)
U	Unweighted case count (volumetrics and weighted tables)
W	Weighted case count (weighted volumetric tables only)
E	Unweighted Event count (weighted volumetric tables only)
D	Percent Down (column or vertical percent)
A	Percent Across (row or horizontal percent)
C	Corner Percent (table percent)
I	Index (ratio of percent in current column to percent in total column)
T	Text (accessible only through post processing)

8.2.3 Cell Punctuation Masks

Each table cell has a mask byte which controls how the cell contents are displayed in the report. The mask byte is normally set by the PCELLS options (see section 7.4.1), which is overridden by any punctuation codes entered in variable specifications (see section 6.6). The mask byte can also be loaded with punctuation codes directly in post processing.

The mask byte for a cell is addressed as the cell name prefixed by the letter M (e.g., TTMF for the mask byte of the Total Frequency cell). The punctuation codes are:

Mask Code	Punctuation
0,1,2,3	Show numbers to no, one, two or three decimal places
4	Add commas to numbers 1,000 or greater
8	Add a leading sign whether positive or negative
16	Show cell in parentheses
32	Toggle percent (%) sign on or off
64	Precede numbers with a dollar sign
255	Suppress cell from printing (this code is not additive)

Punctuation mask codes are additive: To show a number with 2 decimal places and preceded by a dollar sign, the code would be 66 = 64 + 2.

8.3 Post Processing Operations

Post processing instructions fall into two broad groups. The first category of operations is used to perform computations using the contents of data cells, placing the results back into data cells or cell masks within the table. The second category of operations, used mostly to flag statistical significance or other conditions, loads character strings into text cells.

Post processing instructions are entered using a fixed position format in which each component occupies a specific position. For all computational operations, the format is:

Columns	PP instruction element
1	Type of vector: R for row, C for column
2-5	Starting vector within row or column (optional)
6-9	Ending vector (optional)
10-11	Operation code
12-15	Times to loop (optional)
17-36	Argument 1 (1st operand)
37-56	Argument 2 (2nd operand)
57-70	Argument 3 (result field)
71+	May be used for comments

For text operations, the format is essentially the same, except that for operations which allow more than two operands, the operands and result fields may be placed anywhere between columns 17 and 70 of the instruction line.

Two special instructions provide a shorthand for expanding post processing sets without writing out extra instructions, both start in column 1 of a line:

CALL <name>	Inserts the instructions from the post processing set named as if they had been entered at that point in the spec file. The called set must have been previously defined and it may in turn call other post processing sets.
LOOP <m n i1 i2 i3>	Repeats the next m instructions n times, incrementing the arguments by counts of i1, i2 and i3 respectively. This instruction is independent of the post processing looping described below (see section 8.3.3).

8.3.1 Computational Operators

In the following table, A and B represent the 1st and 2nd operand, and C the result field:

Opcode	Operation	Notes
+	C = A + B	If no B, then C = A
-	C = A - B	
Pn	C = (A x B)/10 to nth power	n = 1 through 4
Qn	C = (A x 10 to nth power) / B or blank	n = 1 through 4
K	C = square root of A	no B operand allowed
R or **RD**	C = Rank (Descending) based on A	Columns only, B = ties
RA	C = Rank (Ascending) based on A	Columns only, B = ties
Tn	C = A rounded to n decimal places	n = 1 through 9
<	C = 0 if A < B	
<=	C = 0 if A <= B	
>	C = 0 if A > B	
>=	C = 0 if A >= B	
Cn	C = Reach for n issues of a publication	n = 3 or 4

QTAB traps all attempts to divide by zero in post processing and sets the result field to zero.

The ranking operators (R, RD and RA) will only operate with column vectors. The 2nd operand should contain a constant value that determines how ties are handled in assigning the sequence of rank numbers, as follows:

k0	Ties are ranked by the order in which they appear (no equal ranks are assigned).
k1	Ties receive the same rank and the next rank number assigned will skip as many numbers as needed to account for the number of items ranked (e.g.,1,2,3,3,5,...)
k2	Ties receive the same rank and the next rank is assigned the next number.

The Reach operator is used in media research to determine the average reach for 3 or 4 issues of a publication when the average reach for 1 and 2 issues is known, using the Metheringham beta binomial method. The 1st operand must contain the average reach for one issue and the 2nd operand must contain the average reach for two issues.

8.3.2 Text Operators

QTAB provides three operators that load text strings into the T-cells of a vector when certain conditions are satisfied by the operands. Each of these operators has two forms, to overwrite of any existing contents of the text cell, or to append to any existing contents.

The format of the text operators is slightly different from the computational operators because there may be one or three operands, and there may be one or two results, each of which has two arguments, the text string to be loaded and the destination text cell.

In the following, A,B,C represent operands as used in computational post processing, t,t1,t2 are text strings, and T,T1,T2, text cells in which these strings will be loaded:

Opcode	Operation	Description
TO, TA	A 't' T	If A >= 0, load 't' into T
XO, XA	A B C 't' T	If B=< A < C, load 't' into T
FO, FA	A B C 't1' T1 't2' T2	If A >= 0, then if B > C, load 't1' into T1 and if B=<C, load 't2' into T2

TO, XO and FO overwrite the contents of the text cell they load into, while TA, XA and FA append to the contents of the text cell.

The primary purpose of text cells is to provide flags for significance testing between banner points, where upper case represents one confidence level and lower case represents a lower confidence level, so the contents are sorted in ASCII sequence (ABCabc) and when a letter appears in both upper and lower case, the lower case letter is deleted.

8.3.3 Post Processing Loops

Any post processing instruction many be repeated by entering the number of times to loop in columns 12-15. When looping is called for, at least one vector argument should be immediately followed by a number that indicates by how many vectors it should be incremented for each new repetition of the instruction. Vector arguments may all have different increments.

8.4 Examples of Post Processing

```
PP sample
*--------10-----17--....--37--....--57--....--71--...
C1       -      ttf       1f        2f         (col.3 = total-col.1)
Rtt  tt  Q21000 3f01     2f        3d01        (percentage rows 3+ on row 2)
C1   20  R 10   01f01     k2        01a01      (rank cols.1-10 on f-cell
C1   20  + 10   k16                 1ma1            into a-cell, in parens)
R        +      taf                 NScarry    (Save TA row in numberset)
R1   8   < 5    ttf       K250      1f1        (Zero rows 1-5 if TT<250)
R1       XA     01f  k1 k2  'a'     03t        (If 1=<01f<2,add 'a' to 03t)
END
```

9 DICTIONARIES

A Dictionary is a lookup table by which labels are assigned definitions that will be substituted for them whenever referenced in the spec file. Dictionary substitutions may be used anywhere in the spec file, including in subsequent dictionary definitions. A QTAB spec file may contain any number of dictionaries anywhere before the table definition section.

9.1 Dictionary Labels and Definitions

A dictionary begins with a DICT statement and ends with an END statement. Every line between them, except for comments and blank lines, must be a dictionary entry beginning with a label in column 1, followed by two definitions separated by spaces and enclosed by double quotes ("). All text after the second definition is treated as a comment and ignored, although it will appear in the listing file. The format is:

```
DICT
<LABEL>     <"Definition 1">     <"Definition 2">
 ...
END
```

For each entry, the contents of the first definition will be substituted whenever the label appears in the specification area of a variable or weight definition. The contents of the second definition will be substituted for the label whenever it appears anywhere else in the spec file.

The contents of a definition are only checked if and when a substitution is made in the spec file. A dictionary definition may refer to another dictionary item previously defined in the spec file.

Labels are case sensitive and may be of any length, terminated by a space. Labels are read sequentially from left to right and each label must be unique in a spec file. QTAB checks labels for uniqueness as it compiles the spec file: if ABC is a label, then ABCD will be rejected, because it will be recognized as ABC after the first three characters have been read, whereas ABD will be accepted, because AB was not recognized as a label.

9.2 Using Dictionary Substitutions

Once a dictionary entry has been defined, it may be used anywhere in the spec file by entering its label, prefixed by a double pound sign (##), exactly where the substitution is desired.

Substitutions take place immediately, as the spec file is compiled, and work exactly as if the contents of the definition had been entered where the label appears in the specs. The following example illustrates how dictionary entries are defined and substitutions are called for in specs:

```
DICT
brand          "121"         "Our Brand"
r1             "1"           "first"
r2             "2"           "second"
END
VAR      RANKS        L S36,24
##brandC1@'##r1;##r2'   ##brand ranked ##r1 or ##r2
##brandC1@'##r1'          ##brand ranked ##r1
##brandC1@'##r2'          ##brand ranked ##r2
END
```

The result of the preceding would be exactly as if the variable RANKS had been written as:

```
VAR      RANKS        L S36,24
121c1@'1;2'             Our Brand ranked first or second
121c1@'1'                 Our Brand ranked first
121c1@'2'                 Our Brand ranked second
END
```

Note how the first definition is always used when a substitution is made on the specification side of a variable definition line, while the second definition is always used in the label area.

QTAB parses the text immediately following the ## keyword for a valid label, and continues until it finds the longest one so far in its dictionary as it processes the spec file. This allows labels to be concatenated with additional text, or even other labels, to determine which dictionary entry will be used for a substitution.

9.3 Built-in Dictionary Labels

QTAB provides several built-in dictionary labels for which the definitions are generated by the program when it runs. These are:

##SYS_DATE	The current date at the time of execution, as it appears at the top of the listing, in the format: Mth dd yyyy (e.g., Dec 16 1998)
##SYS_TIME	The time at which QTAB began execution, as it appears at the top of the listing, in the format: hh:mm (e.g., 15:30)
##SYS_INTAP	The name of the input file as it appears in the first INTAP data directive specified
##SYS_REPORT	The name of the report file as it appears in the REPORT data directive

These built-in dictionary entries have the same contents for both spec and text definitions.

9.4 Command Line Parameters

QTAB provides for pairs of parameters and their arguments to be passed to the program on the command line. Each parameter becomes a dictionary entry label, and its argument is used as the contents of both spec and text definitions for that dictionary entry. The format is:

QTAB specfile **/# Parameter1 Argument1 Parameter2 Argument2 .../#**

The following example demonstrates how a command line parameter can be used with a table of dictionary entries to select the output format of a QTAB run:

```
QTABD QTDEMO.QTS  /# RUNTYPE P /#                <--- Command line
...
DICT
T_OPT_P  ""  "PCELLS dta.0  SUM rta  FPRINT"      Use for Percents only
T_OPT_F  ""  "PCELLS f.0,dta.0  SUM rta"          Use for Freqs & percents
END
...
MERGE     STUB    HEADER                 ##T_OPT_##RUNTYPE
...
```

In this example, the argument for the command line parameter "RUNTYPE" is "P" which will be substituted for ##RUNTYPE in "##T_OPT_##RUNTYPE " in the MERGE statement. This will be recognized by QTAB as the dictionary label T_OTP_P, and the text definition for that dictionary entry will be substituted into the MERGE statement, specifying the desired table options. When using command line parameters in this manner, care must be taken that any argument supplied is accounted for in the specs, otherwise the reference will not resolve, causing a fatal error.

10 MACRO TEMPLATES

QTAB allows a variable to reuse the specs or label text from another variable through the use of equates (see section 4.1.3), but it also provides a more powerful method for generating similarly structured variables quickly and reliably from macro templates.

A macro template is a model variable identified by the keyword MACRO, instead of VAR, in which any elements that may need to be changed are replaced by place markers identified by a caret (^) followed by a case-sensitive alphanumeric character label (^A and ^a are different). Place markers may be used anywhere in the body of a variable, except in the spacer column of any line other than row label text in a stub variable. Post processing cannot be called directly from a macro template.

10.1 Defining Variables From Macro Templates

A variable is defined from a macro template by a macro equate **Mnn**, where nn is a column offset applied to all specifications defined in the template. Post processing may be called for after the macro equate, but no other equates, specifications or label text except for substitution parameters are allowed anywhere in the variable. Each substitution parameter must appear on a separate line, with the place marker label in column one, followed after one or more spaces by the replacement text in double quotes.

QTAB treats a variable defined from a macro template just as if the contents of the template had been entered for the variable definition, with the contents of each substitution parameter beginning exactly where the corresponding place marker appears. The only exception to this rule is that substitutions in the spec area may be longer than would fit in the space if the variable were specified directly. This is not true for substitutions in the text label areas, where care must be taken to preserve columnar alignments.

For stub row labels only, place markers may start in the spacer column. This allows templates with what appear to be variable numbers of rows, using spacer codes at the beginning of label text substitutions to suppress the display of specific rows in the output (see section 4.2.2).

Here following is an example of a simple macro template:

```
MACRO   M1     L S36,16
               THave you ever purchased ^A
101'1,2'        Yes
                No
^1             ^X
END
```

Here is an example of a variable built from the preceding template:

```
VAR     V1     L M00;M1
A  "Brand X"
1  "stk0"
X  "D Not used in this table"
END
```

In this example, the third line of the table which could have been used for another response, was defined as a dummy value (a subtotal containing the constant value zero), and the row label text suppressed by use of the "D" space code.

10.2 Using Macro Templates With Dictionaries

Dictionary entries may be used in macro templates and in substitution parameters and function the same way as in any other variable, with the first definition substituted for the label when it appears in the spec area and the second definition substituted everywhere else. This means that a single dictionary entry can be used to provide both specifications and label text for a row or column, or for a table title.

The following example illustrates the use of dictionary entries and parameter substitution to generate tables from a very generic template:

```
*** Dictionary definitions
DICT
BR_A    "112"   "ALPHA"
BR_B    "113"   "BETA"
BR_C    "114"   "GAMMA"
TOP1    "1"     "FIRST"
TOP2    "12"    "FIRST OR SECOND"
END
*** Template
MACRO   SUMTAB                L S24,32
                              TBRANDS RANKED ^1 FOR
                              T- ^2 -
##BR_A'^1'                     ##BR_A
##BR_B'^1'                     ##BR_B
##BR_C'^1'                     ##BR_C
END
*** Variables
VAR     Q2S1                  L M0;SUMMARY
1  "##TOP1"
2  "TOTAL SALES VOLUME"
VAR     Q2S2                  L M0;SUMMARY
1  "##TOP2"
2  "TOTAL SALES VOLUME"
VAR     Q3S1                  L M7;SUMMARY
1  "##TOP1"
2  "QUALITY OF CUSTOMER SERVICE"
VAR     Q3S2                  L M7;SUMMARY
1  "##TOP2"
2  "QUALITY OF CUSTOMER SERVICE"
END
```

11 FLOW CONTROL AND BATCH PROCESSING

QTAB is strictly script-driven, as are most of the programs in the QUIP System. It is designed for a production environment and optimized for repetitive processing tasks. QTAB can also be used as a tabulation engine for "user friendly" systems built with environments or languages that can generate text instructions and run external processes, such as FoxPro, Delphi, Visual Basic, and most standard programming languages.

Through judicious use of dictionaries and templates, it is possible to design "generic" spec files that can be used for a number of similar projects, needing only minimal information to be added or modified for each individual set of tabulations. QTAB provides flow control features to assist with batch processing and process automation situations, and to allow building modular libraries of spec files that can be selected and included into other files at run-time.

11.1 Includes

The include statement allows a spec file to call another spec file. QTAB processes an included spec file as if its entire contents appeared in the calling file beginning at the line containing the include statement, and then returns to processing the calling file at the next line. Included spec files can in turn call other spec files, but the flow of specs will always return through all nested levels to the original file.

An include statement begins with the keyword #INCLUDE in column 1, followed by the name of the spec file to be included. The format is:

#INCLUDE <[path]specfile.ext>

An include statement must always follow an END statement (see section 1.2), so it cannot appear within a variable, macro template, dictionary or the table definition section, all of which would be terminated by the END statement. Likewise, any spec file that will be called by an include statement must also be terminated by an END statement.

11.2 Stopping and Starting Specs and Listings

QTAB provides special statements that allow the flow of spec processing to be turned on or off when reading a spec file, and to suppress the output of those specs processed to the listing file. The flow control statements all consist of a single beginning in column 1 with a keyword and, if allowed, its parameter, with no other text on the line. They are:

STOPSPEC	Stop processing the spec file from this point until a STARTSPEC statement is read.
STARTSPEC	Resume processing the spec file if it has been stopped by a STOPSPEC statement.
STARTSPEC IF <##X=Y>	Resume processing the spec file if it has been stopped by a STOPSPEC statement and if the definition of dictionary entry X matches the character string "Y".
STOPLIST	Stop sending any output to the listing file from this point until a STARTLIST statement is read.
STARTLIST	Resume sending output to the listing file if it has been stopped by a STOPLIST statement.

All of these statements must appear after an END statement and take effect immediately and unconditionally, except for STARTSPEC IF, which applies only if the condition evaluates true. STARTSPEC IF may be used with any dictionary entry, however it is most effective when used with command line parameters (see section 9.4) to control inclusion or exclusion of sections of spec files at run time.

In the following example, one or the other of two spec files (or neither) will be included in the calling spec file depending on the value of the dictionary entry labeled RUNCODE:

```
    STOPSPEC
    STARTSPEC IF ##RUNCODE=A
    #INCLUDE subfile1.qts
    END
    STOPSPEC
    STARTSPEC IF ##RUNCODE=B
    #INCLUDE subfile1.qts
    END
    STARTSPEC
```

12 INTERFACING WITH OTHER SOFTWARE

QTAB normally reads data from flat files and outputs tables in a single plain ASCII report file. These formats provide the widest level of compatibility, allowing almost any data to be readily converted to a file QTAB can tabulate, and making it possible for QTAB's output to be imported into most other analytical software, either directly or through format conversion programs.

QTAB provides a few special options designed to make it easier to use directly with several other types of programs commonly used to process or analyze data in a PC environment.

12.1 Tabulating Data from .DBF Files

QTAB can tabulate dBASE type (.DBF) files directly. The keyword "DBASE" is used in the INTAP statement instead of a record length to indicate that the data file is in this format:

```
    INTAP datafile.dbf ( DBASE.
```

Addressing is implicit in a DBASE type run, so no SETUP ADDRMODE statement should be entered in the spec file. Data locations are addressed as dBASE field names surrounded by dollar signs. DBASE fields are actually ASCII character fields starting at that address, from which up to the 15 leftmost characters can be accessed using normal QTAB specs.

The following example could be used to sum sales volume within region:

```
         $REGION$C4@='EAST';$SALES$C10      Eastern Region
         $REGION$C4@='CENT';$SALES$C10      Central Region
         $REGION$C4@='WEST';$SALES$C10      Western Region
```

In this case, the field named "REGION" could be of any size in the .DBF file, as long as the contents are left justified, because only the first 4 bytes will be tested. The field "SALES" should be numeric with a width of 10 bytes. Explicit decimals (with a period) will be read properly.

QTAB does not generate database reports, but it provides far greater flexibility and speed in cross-tabulating data than the report generator in any database program. QTAB operates in batch mode, so many database programs that support DBASE type files and have procedural languages can be used to build spec files and run QTAB on their data.

12.2 Importing Tables into Spreadsheets

Two global options may be specified on the RUNOPS line (see section 3.2) to facilitate the conversion of QTAB tables into popular spreadsheet formats by writing individual tables to separate files and by adding tab character delimiters between table cells. These options, which may be used individually, or together, are:

EXCEL	Adds a tab character immediately following each table cell and immediately after the stub label area for each table row that contains printing cells. While this may cause the table columns to appear out of kilter with the banner annotation when printed or viewed on screen, it allows many spreadsheets that can import tables in "tab delimited" format to read the table cells into separate spreadsheet cells.
LOTUS	Writes out each individual table as a separate file. The file name of each file will be as given in the REPORT data directive (see section 2.1), but file extensions will be used to number the files sequentially, starting with 001.

Some tips for successfully setting up tables to be read into spreadsheets are:

- Set page length to 999 lines in the RUNOPS to prevent page skips and repeating banners.
- Keep stub labels on a single line and let the spreadsheet wrap annotation.
- Show frequencies or percents, but not both. The FPRINT table option (see section 7.4) can be used to show the frequencies in the base row or column of percent only tables.

13 WHERE TO GET ADDITIONAL INFORMATION

For information about QTAB and other QUIP System programs, contact:

Jan Werner Data Processing

www.jwdp.com