QGEN

A Professional Data Manipulation Program

User's Guide

Revision: 21.1.5

QGEN is the data cleaning and manipulation component of the QUIP System

QUIP System software is available from:

www.quipsoftware.com

QGEN 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 in the most demanding production environments and handle data sets that are too large or too complicated for other tabulation programs or integrated survey packages. The QUIP System programs are entirely script-driven and run from a command prompt, permitting batch processing or execution from within other programs, as required for repetitive tasks and for building automated systems.

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 QGEN, the data recode and cleaning component of the QUIP System. It aims to teach beginners the basics of QGEN 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.

QGEN 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).

Whenever the name QGEN appears, users of the demo version should substitute QGEND.

1 HOW QGEN WORKS

QGEN is a data generation language interpreter, much like the RPG (report program generator) languages used in the early days of data processing. QGEN compiles a set of instructions and then applies them to the data from one or more input files, one record at a time, generating new records in one or more output files as it progresses sequentially through the data.

The most common uses for QGEN are to recode each record of a data file according to a given set of directives, or to list those records that break certain rules. But QGEN can be used for a wide variety of other purposes, such as generating or reformatting text files, summarizing data, collapsing hierarchical data structures or building new ones from flat files.

QGEN provides a full set of tools to control the flow of execution, cumulate data across records, insert data from one file into the matching records in a second file, assemble complete records from discrete components, create multiple records from a single record, direct output to multiple files selectively and generate printable reports from a data file.

1.1 The Work Area

The work area is a region of memory into which QGEN reads a record from the input files before executing the instructions in its script. The size of the work area is 250,000 bytes, which is also the maximum record length that QGEN can read or write

QGEN is said to be "data driven" because all of its actions are determined by the contents of the work area at the time each instruction is executed.

QGEN instructions can access any location in the work area. Input records replace any existing content, but the remainder of the work area is not cleared by default between records, allowing information to be cumulated or passed from one record to subsequent ones.

1.2 Input and Output Streams

QGEN can read records into the work area from two separate data streams simultaneously and output to many separate output streams, usually about 30 in a Windows environment.

The input stream is the primary incoming data stream, functioning as the master file when an insert stream is also specified. Each record in the input stream is read in its entirety into the work area as a case, or logical record.

The insert stream is a slave or trailer incoming data stream, relying on a case ID field in the data to distinguish between cases. Any number of physical records from the insert stream may be inserted into a single case, or logical record, in the work area, usually requiring a cardcode field (see section 4.2) to distinguish between different record types.

Either or both of these streams may be specified for a given QGEN run. It is even possible to generate output data from no input data at all, using only instructions and an end of job recycle.

Each of the two input data streams may each specify any number of files which will be read consecutively in the sequence they appear in the specifications.

Output streams are written simultaneously and each generates a single data file.

Section 5 describes the flow of data from the input streams into the work area in a QGEN run. The directives used to define the input and output streams are describe in section 3.1.

1.3 Other Input and Output

In addition to the input and output data streams, QGEN can generate a report file containing plain ASCII text produced a line at a time using special instructions. A rudimentary interactive capability is also provided in the form of SAY and GET instructions that send text to the console and retrieve input from the keyboard (see section 7.4).

The only file that is always created by a QGEN run is the listing file, which numbers the lines of instructions in the script file, displays error messages, lists cleaning rules broken and a number of summaries and statistics. ASCII text can be placed in the listing file in addition to, or instead of, the report file or any output files.

2 THE SPEC FILE

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

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 except in dictionary labels and literal strings.

Spaces are used as separators between operands and keywords. The number of spaces does not matter if less than 20. QGEN considers any text following 20 or more consecutive spaces on an instruction line to be comments and shows them in the listing while ignoring their contents. Tab characters are not allowed in specifications. Blank lines are ignored but show in the listing.

QGEN compiles a spec file in a single pass so items such as look-up tables and dictionary entries must be defined before they can be referenced elsewhere in the spec file, but tags used as program flow targets may appear anywhere in the file as they are resolved after compilation.

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

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

QGEN will compile the spec file named on the command line, checking for syntax errors and writing the listing to disk as it proceeds through the specs. If there were no errors in the specs, the input data will be processed according to the instructions, and any output files written.

2.1 Specification Types

A QGEN spec file contains several different types of specifications, which should appear in the following order in the spec file:

DATA DIRECTIVES:	Specifications that describe the listing, input and output files.
RUN LEVEL OPTIONS:	Options and parameters that apply to the entire QGEN run and describe how data will be addressed.
INSERT DEFINITIONS:	Required only if data is read from an insert file to define where the inserts are placed in the data record.
INSTRUCTIONS:	Tell QGEN what to do based on the contents of the work area for the current record.

Other specifications, such as dictionaries and includes, may appear anywhere in the spec file.

2.2 Comments and Special Commands

Any line beginning with an asterisk (*) in column 1 is treated as a comment, appearing in the listing file but otherwise ignore by QGEN.

An END statement is a line with the keyword END beginning in column one and containing no other text. END statements are used to terminate dictionaries (see section 10) and lookup tables (see section 10). An END statement anywhere else in the spec file forces the immediate end of the specifications causing QGEN to skip past all remaining lines in the spec file.

Except within lookup tables and dictionaries, a line containing only the letter "P" in column 1 causes a formfeed to be inserted at that point in the listing file. A line containing only a dollar sign ($) in column 1 forces the immediate end of the specifications, like an END statement.

3 DATA DIRECTIVES

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

3.1 File Names

QGEN accepts the following data directives:

INTAP[n]	<[path\]filename.ext (lrecl> [V]	Input data file.
INDATA[n]	<[path\]filename.ext (lrecl> [V]	Insert data file.
OUTAP[n]	<[path\]filename.ext (lrecl> [V|C]	Output data file.
REPORT	<[path\]filename.ext>	Report text output file.
LISTING	<[path\]filename.ext>	Spec listing file.

If no LISTING file is specified, QGEN 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. When no explicit path is given, QGEN always reads from and writes to the current directory. All file names must be in DOS format (8.3).

INTAP, INDATA and OUTAP directives name data files followed by a left parenthesis and the record length in bytes. Unless explicitly specified as variable or constant, the record length is assumed fixed, meaning that each consecutive block of bytes of the specified record length is considered a separate record.

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.

The letter "C" after the record length in OUTAP directives indicates constant length records, which are padded with blanks to the indicated record length, then terminated with a CR-LF pair. QGEN looks for the CR-LF when reading files specified as either variable or constant, so while a "C" may be used in input data directives without causing an error, the result is identical to "V".

3.1.1 INTAP and INDATA Files

A QGEN run may have any number of INTAP and INDATA files, which may be numbered, as INTAP1, INTAP2, INDATA23, etc., although any numbering is ignored by QGEN. Both input streams are read simultaneously, and within each stream, files are processed sequentially in the order in which they appear in the spec file, regardless of numbering. All the files in each stream should have the same record length specified.

3.1.2 OUTAP Files

Each OUTAPn data directive generates a separate output data stream. The number "n" in the keyword is required if more than output file is specified and will be used in the spec file to direct data to the output streams it identifies. If "n" is omitted, the file will be referenced as OUTAP1.

Each individual OUTAPn directive immediately opens a file for output, overwriting any existing file by that name without warning, whether or not any data is ever written to the file.

By default, QGEN writes each case to OUTAP1 when it reaches the end of the instructions. All other output activity must be explicitly specified by using the appropriate output actions to select or drop output streams for the current case (see section 7.4).

3.2 Command Line Data Directives

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

/I[n]= <[path\]filename.ext>	INTAP file name
/IL[n]= <nnn >	INTAP record length
/D[n]= <[path\]filename.ext>	INDATA file name
/DL[n]= <nnn >	INDATA record length
/O[n]= <[path\]filename.ext>	OUTAP file name
/OL[n]= <nnn >	OUTAP record length
/L=<[path\]filename.ext>	LISTING file name
/S[n]=<[path\]filename.ext>	SPEC file name
/T=<[path\]filename.ext>	REPORT 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, INDATA and OUTAP files may be specified on the command line for a single run and each must have a matching record length (e.g., /ILn=). Variable and constant length records cannot be specified using command line data directives.

4 GLOBAL OPTIONS

Global options apply to an entire QGEN run. They should be set after the data directives and before any instructions. Except for the data address mode, all run options are specified on the RUNOPS line. All options have defaults that are used when explicit settings are not specified.

4.1 Address Mode

QGEN 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 QGEN 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 6.1.1.

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, QGEN defaults to ADDRMODE B (column-binary data, card/column addressing).

4.2 The RUNOPS Line

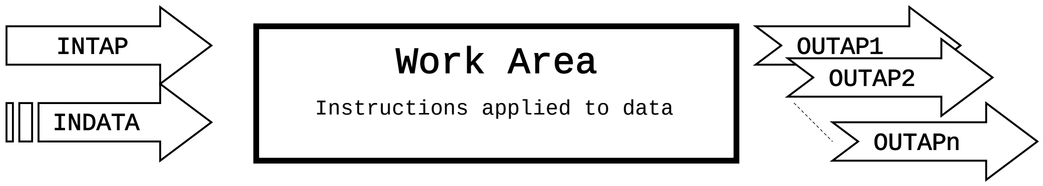
The RUNOPS line begins with the keyword RUNOPS. Options are invoked by keywords separated by spaces, some of which are followed by one or more parameters. RUNOPS line options are listed below with the applicable parameters.

RUNOPS	Indicates that this is a runops line. Must be the first item on the line.
CC datafield[,datafield]	Gives the location in the insert data file (INDATA) of the cardcode used to identify physical records that are to be inserted into a single logical record in the work area. Two fields can be specified, separated by commas, and their contents concatenated into a single cardcode. A maximum of 20 alphanumeric characters can be used. See section 5.1 below for more information on cardcodes and insertions.
CLEAR addr,addr[;addr,addr[;...]]	Areas in the data work area to be cleared between records. Each area is defined by the beginning and ending address, separated by commas. Up to 10 areas can be specified, separated by semicolons. The default is no area cleared, except for that used by the input data (INTAP).
ID datafield[,datafield]	Tells QGEN where to find the case ID in the input data file (INTAP). Two fields can be specified, separated by commas, and their contents will be concatenated into a single ID. A maximum of 20 alphanumeric characters can be used.
IDIN datafield[,datafield]	Specifies the case ID field in the insert data file (INDATA). The same rules apply as for ID.
INSTART addr	Starting location in the work area for the input data (INTAP). Defaults to the first byte of the work area.
LENGTH nnnnn	Page length (in lines) for the listing and report files. The default is no page skips, maximum is 32,767.
MAXERR nnn[,nnn][,nnn]	Maximum number of various data error messages that QGEN will accept before aborting a run. The first number is the number of broken cleaning rules. The second number is the number of file-matching error messages from an insert. The third number is the number of data conversion errors to allow. If not specified, the default for each is 1000. The maximum value for each limit is 32,767.
NOMATCH	Suppresses the error messages generated when insert records are bypassed because they do not have matching records in the input data stream.
OUTSTART addr	Starting location in the data work area from which the output data will be written. The default is the first byte of the work area.

RECYCLE	Causes QGEN to execute an additional pass of the specifications after the last input record has been processed. OUTAP1 is selected unless a DRP instruction (see section 7.4) is used to turn it off. Switch 5 is set on at the beginning of the recycle pass so that the user can test whether the current pass is on actual data or the end of job recycle.
SEQ type[,type]	Requests sequence checking for input data. The first "type" applies to the ID field in the input data and the second to the IDIN filed in the insert data. Types are A (ascending), E (equal or ascending--allowed for INTAP only), and N (none).
TEST nnnnn|ALL|SPECS	Stop after processing 'nnnnn' cases, ALL cases, or after testing the SPECS. The default is ALL.
WIDTH nnn	Maximum width of text output for both the LISTING and the REPORT files. Valid range is 80 to 250. The default is 132.
ZCOUNTER nn	Number of decimal places to use for the display of counter contents in the listing file (nn = 0 to 15). The default is 0.

5 DATA FLOW

The diagram below illustrates how the input and insert streams are combined for each logical record in a QGEN run to produce one or more output streams:



The essential difference between the input and insert streams is that each record in the input stream generates a single logical record, or case in the work area, whereas the records in the insert stream are treated as separate elements of a logical record identified by a case ID.

Under most circumstances, QGEN requires only the input stream defined by the INTAP data directives, but there are two specific situations require data to be read from the insert stream defined by the INDATA data directive.

The first of these occurs when additional data must be inserted into some or all of the records in the input file. In this situation, both input and insert streams are used.

The second occurs when the data consists of separate physical records that must be combined into a single logical record for each case. In this situation, only the insert data stream is used.

5.1 INSERT Instructions

Insert instructions tell what to read from the INDATA file and where to insert it in the work area. If the insert stream contains more than one physical record per logical record, each record must be identified by a unique code in the cardcode field (CC on the RUNOPS line, see section 4.2), and each distinct record type requires a separate insert instruction, which may specify as many as 10 different insertions from that record.

An insert instruction is written on a single line, beginning with the keyword INSERT, followed by a cardcode between single quotes if there is more than one record type, an optional 'mandatory record' flag, and up to 10 different insertion specifications. The format is:

INSERT ['code'] [M|P] aa1,bb1,cc1 [aa2,bb2.cc2 ...

The cardcode may be up to 20 columns long and may contain any characters. An asterisk (*) in any column of the cardcode is a wildcard representing any character in that position. The size of the cardcode in each insert instruction must match that given for CC in the RUNOPS line.

A mandatory record flag M tells QGEN that this record must be present if any insertions at all were made into a logical record. The flag P, which tells QGEN that this record type is optional, may be omitted.

Each insertion specification lists three addresses in the current addressing mode, separated by commas. The first is the starting address in the INDATA record for the data being inserted, the second is the starting address in the work area for the data to be placed, and the third is the end address in the work area for the inserted data. The second and third addresses determine the amount of data inserted by each specification.

5.2 IDs and Sequence Checking

While each record in the primary input stream (INTAP) always generates a case, records from the insert data stream (INDATA) can be lost for several reasons, as described in this section.

When inserting data from INDATA into INTAP, QGEN reads both files in parallel and attempts to keep them synchronized on the case ID, as defined by the RUNOPS parameters ID and IDIN (see section 4.2). If one file contains records that are not in the other, this synchronization may fail and data for subsequent matching records in INDATA may not be properly inserted. This can be avoided by sorting both files on the ID and calling for sequence checking with the SEQ parameter on the RUNOPS line. When it is known that both files contain the same cases in the same order, the IDs need not be sorted, and sequence checking may be omitted.

The order of multiple physical records within a logical record in an insert file is not important, but in the case of duplicate cardcodes, the first will be inserted and any following ones skipped.

For logical records constructed from multiple different physical records (INDATA, but no INTAP), a new logical record will be created each time a new ID is read from the insert stream and it is not necessary to check the ID sequence.

It is possible to build logical records from multiple physical records with no common ID or with no unique cardcode to differentiate between them by reading records one at a time from the primary input stream and constructing the logical record in the work area, writing it out only when fully assembled, or after the first record of the next case has been read. This can be done using the output control actions described in section 7.4, but it requires considerably more effort than using the built-in insert mechanisms.

5.3 Some Examples of Data Inserts

The following examples illustrate some ways of using inserts in QGEN.

```
    RUNOPS  IDIN 101b4   CC 180b1   CLEAR 101,280
    INSERT  '1' M 101,101,180
    INSERT  '2'   101,201,280
```

This illustrates the common case where a logical record is built from two card images, the first of which is require, the second of which may not be present in the input file. The record identified by cardcode 1 in column 80 is read into the work area as card 1, if a record with the same ID has cardcode 2, it will be read into the work area as card 2. Note that IDIN, CC and the source field in the INSERT instructions are all specified as their positions in the insert records, whereas the output field in the INSERT instructions is specified as the position in the work area. In this situation, it is essential to clear the work area where the data will be inserted, because a case that had only one card image in the insert file would otherwise inherit the data for the other from the previous case. A record will be created if either or both cards are present for a case, but if card 1 is missing, it will be listed with a "MANDATORY CARD(S) MISSING" message.

```
    RUNOPS   ID 21c8,31c2   IDIN 1c10   CC 11c3   SEQ A,A   NOMATCH
    INSERT   'A**'   21,41,43   24,51,53   27,61,63
    INSERT   'a**'   21,41,43   24,51,53   27,61,63
```

This shows a situation where data is being inserted into an existing file. Data from any record with a cardcode beginning with the letter "A" in upper or lower case will be inserted into a record with the matching ID in the primary data file. Note that the ID is constructed from two parts in the main file, while it is in a single field in the inserts, and that no messages will be generated for records in the insert file that do not have a matching record in the main file. If no insert is made for a record in the primary file, the contents of the insert areas will remain unchanged.

6 THE QUIP SPECIFICATION LANGUAGE

The QUIP specification language allows QGEN to define the tests and numeric values in the data that instructions use to perform computations and construct logical rules for decisions.

6.1 Data Specifications

When tabulating data, QGEN 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.

6.1.1 Addressing Data

QGEN addresses data using absolute column locations within each the work area. The location is specified as an offset from the beginning of the work area using direct addressing or using the card/column notation widely used in statistics and marketing research. The actual data location addressed by a specification is defined by the address mode (see section 4.1).

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.

QGEN 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.

6.1.2 Fields and Numbers

A data address followed by one of the codes listed below will be treated as a field or a number. QGEN 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. This translates to Bn if the address mode is B or E, and to Cn if the address mode is C or D.
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 comparisons. Column-binary fields (A, C) 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. To handle 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).

Data type A accepts another suffix, "X", that causes it to work like the data type X (extended column-binary) with either column-binary of ASCII data, as per the address mode. In ASCII data, X-punches are represented in data by minus signs, so for a field defined as 21AEX3, an entry of "---" would translate to 1000, "--1" to zero, and "-10" to minus 10. This suffix may be used in operands but cannot be used in result fields.

6.1.3 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.

6.1.4 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. QGEN 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. Multiple literals may be specified within a pair of single quotes for certain types of tests (see section 6.3).

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).

QGEN 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.

6.2 Counters and Switches

Outside of the work area, QGEN provides two data areas that are not cleared between cases and cannot be read directly from the input streams. These are called counters and switches and are designed to be used for counting events or flagging conditions across cases. An additional set of program switches is used to flag certain conditions within the case being processed.

6.2.1 Counters

QGEN provides 1000 double-precision floating point counters addressed as Z1 through Z1000.Counters are set to zero at the beginning of a QGEN run, not cleared between cases, and program actions can increment them to count events across cases (see section 7.3.3). They may also be addressed directly in arithmetic expressions and instructions, like any other double-precision floating point field.

Counters are stored in adjacent memory locations and can be looped through with an 8-byte increment (see section 9). Counters may also be used to provide variable numbers of passes for loop instructions and while a loop is executing, the current pass number may be read from counter Z1000, which should therefore not be used for other purposes.

6.2.2 Switches

QGEN provides 36 single-byte fields that can take only the values 1 or 0, called switches and identified by the keyword **SW** followed by a one-character alphanumeric identifier. All switches start with the value zero at the beginning of a QGEN run.

The 26 letter switches (SWA through SWZ) are called user switches and are not reset to zero between cases. They are often used to flag conditions within or between cases.

The numbered switches (SW0 through SW9) are reset to zero between logical records and set to one to flag certain conditions that could not otherwise be directly tested for in specifications. They should not normally be set by the user.

The conditions indicated for the current record by numbered switches are:

SW1	The record was out of sequence in the input data stream (INTAP).
SW2	One or more inserts were successfully made from the insert data stream into the current logical record (from INDATA).
SW3	One or more insert records were bypassed as invalid for the current logical record (from INDATA)
SW4	A mandatory record type (see section 5.1) was missing from the insert data stream for the current logical record (from INDATA)
SW5	Indicates an end-of-job recycle pass, after the last record from any input data sources has been processed. See the definition of the RECYCLE keyword in section 4.2 for details.
SW6	A cleaning rule (see section 7.6) was broken. This is set to 1 after each broken rule, so it may be reset to allow testing the current case against later rules in the spec file being broken.
SW7	This record begins a new data file in the input stream.
SW8	This record begins a new data file in the insert stream.
SW9	A conversion error or other arithmetic anomaly occurred. Set to 1 after each error, so it may be reset to allow testing later in the spec file.

6.3 Testing Data

A case is counted by QGEN 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.

6.3.1 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.

6.3.2 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 6.1.3). 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.

6.3.3 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 QGEN 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.

Compound expressions are evaluated from left to right but this will stop if any invalid values are found in a numeric field, causing the entire specification to default to false. For this reason, data in numeric fields used in compound expressions should always be tested for validity.

6.3.4 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
`Z21='100'`	Test for a value of 100 in counter Z21 (see section 6.2.1)
`SW2'1'`	Test for switch 2 being set (see section 6.2.2)

7 INSTRUCTIONS AND ACTIONS

QGEN instructions are written one to a line and are executed sequentially in the order they appear in the spec file, unless the order of execution is modified by a program flow control action (see section 7.5).

The format for instructions is:

[Tag] <Result> <Operation Code> [Operand A] [Operand B] [Operand C] [Operand D]

Except for tags, the elements of the instruction must be separated by spaces. The number of spaces does not matter, but 20 or more consecutive spaces indicate that all remaining text on that line are to be considered comments and ignored by QGEN. No spaces are allowed within the result or operand fields themselves, except within literal strings.

Depending on the nature of the instruction, as defined by the instruction code, the operands may be test conditions, data fields, literal strings or numeric constants as described in section 6. See section 8 for descriptions of the instruction codes.

7.1 Tags

A tag consists of two characters between underscores, (e.g., **_XX_**) and provides a label for an instruction line as the target of a program flow control action (see section 7.5) or as the end of a loop instruction (see section 9). A tag must always be the first item on the line it identifies.

Tags are resolved after all the instructions in a spec file are read. Actions may reference tags anywhere in the spec file, while loop instructions may only reference tags on following lines.

Tags may use any alphanumeric characters, are not case sensitive. and must be unique within a spec file, so two lines tagged as _A1_ and _a1_ in the same spec file will cause a spec error and prevent QGEN from running. References to tags need not use the underscores unless the tag label is numeric, which would cause a reference to an absolute or relative line number.

7.2 Results and Actions

For arithmetic, data conversion, table lookup and most column-binary punch instructions, the result field specifies where the result of the operation will be placed. This can be any location within the work area, including counters or switches (see section 6.2.2) when appropriate.

For arithmetic instructions, the result must be a valid numeric field definition (see section 6.1.2). Numeric results in character and column-binary fields are right-justified and left-zero-filled, with a minus-sign (-) entered in the first column of the field for negative numbers.

QGEN does not check for underflow or overflow, so it is up to the user to ensure that the field specified is adequate to contain the result of the operation. For all types except floating point, only the integer part of a decimal fraction is retained and the most significant digits are dropped if the number is too large to fit the allocated result field. In general, it is a good idea to perform computations in floating point, using counters as temporary work areas, then to test the results before converting them into the final desired format in the data area.

For data conversion and table lookup instructions, only the beginning of the output area should be specified, as the size of the output field is determined by the operands or the lookup table.

For logical, comparison and some column-binary punch instructions, the result field will be either an action or a cleaning rule name (see section 7.6).

An action is taken when the result of the instruction applied to the operands evaluates true and modifies some data in the work area, the output streams, or the flow of program control.

Multiple actions may be specified in the result field of an instruction by concatenating them with an ampersand (&). Multiple actions are executed from left to right, as in the following example.

```
    121b1*&121'Y'  AND   121t>1'1-Y'
```

This instruction will evaluate true if column 121 contains more than one punch. The first action will clear the column of all punches (121b1*), and the second action will add the Y-punch to the column (121'Y'), thus resulting in a column containing only the Y-punch.

7.3 Data Modification Actions

There are four types of data modification actions that can be specified as an instruction result:

- A character or column-binary field can be cleared.
- A constant value, string literal or punch can be entered into a field or column.
- A counter can be incremented.
- A switch can be set or cleared.

7.3.1 Clearing Fields

Character (ASCII) and column-binary fields are cleared to blanks by following the field definition in the result area with an asterisk (*). For ASCII character fields, all bytes are cleared to spaces (decimal 32/hex 20). For column-binary fields, columns are cleared to nulls (0 in both bytes).

```
    51c200*  ALL                           Clear bytes 51-250 to ASCII spaces
    213b2*  AND 211b2b                     Clear cols. 213-214 if 211-212 are blank
```

7.3.2 Entering Values Into Data

A numeric constant is entered into a numeric field (but not a counter, see section 7.3.3 below) by specifying the field in the result area followed by the desired value between single quotes. Negative and fractional values may be used, but all decimals will be discarded in integer fields.

```
15c3'1'   ALL                          Will enter "001" into bytes 15-17
15c3'-1'  ALL                          Will enter "-01" into bytes 15-17
15c3'1.5' ALL                          Will enter "001" into bytes 15-17
15d'1.5'  ALL                          Will enter the floating point value 1.5
                                          into bytes 15-22
```

A string literal may be entered into a character field by specifying the field in the result area followed by the string operator (@) and the string between single quotes. The length specified for the field must be the same as the length of the string.

```
15c3@'abc' ALL                         Will enter "abc" in bytes 15-17
15c3@'1'   ALL                         *** Will cause a spec error!
```

Punches are entered into columns by specifying the column followed by the punches between single quotes. Multiple punches may be specified in column binary data and they will be added to the column without clearing or displacing any punches already present. In ASCII data, a "punch" in a "column" is actually a character in a byte, so only one "punch" can be specified and the column must already be blank in the work area for the current record, or a run-time error will be generated and the instruction will not be executed.

```
121b'234'  ALL                         Adds punches 2,3 and 4 to column 121
121c'234'  ALL                         *** Will cause a spec error!
121'2'     ALL                         Adds punch 2 to column 121
```

In address modes C or D (see section 4.1), the last example above will fail with a run-time error message for any record in which the work area byte addressed as column 121 is not blank at the time the instruction is executed.

7.3.3 Incrementing Counters

A counter is incremented by a numeric constant by specifying the counter in the result area followed by the value of the constant between single quotes. The format is the same as when entering constants into data fields, except that the constant is added to the previous value of the counter, instead of replacing it.

```
z21'1'   ALL                          Adds 1 to the value in counter z21
z21'-1'  ALL                          Subtracts 1 from the value in counter z21
z21'1.5' ALL                          Adds 1.5 to the value in counter z21
```

7.3.4 Setting and Clearing Switches

Switches are set using the same syntax in the result area as used to test switches in operands. Setting a switch is unconditional, that is, it does not toggle the value of the switch:

```
swa'1'   all                          Set switch A to 1 (on)
SWB'0'   all                          Set switch B to 0 (off)
```

7.4 Output Actions

These actions provide a variety of methods for QGEN to send data to different output streams or files, to interact with the user while it is running, or to control subsequent actions of a batch file after the run terminates.

By default, OUTAP1 is selected for output and all other streams deselected when QGEN begins processing each case. The SEL and DRP actions allow control over which files a case will be written to, either when an explicit WRT action is issued, or by the automatic write QGEN issues when it reaches the end of the specifications for each case.

Except for the LISTING file, which is always created, no output will be written to any file that has not been specified by a data directive.

Action	Description
DRPn[,m...]	Deselects output streams numbered n, m, etc. for the current case. See the SEL action below for more information.
GET'prompt'<addr>	Displays the prompt on screen and pauses until a string is typed at the keyboard and the enter key pressed. The string is stored in the work area starting at the address specified.
LST<addr>[ctrl]	Immediately writes to the listing file the contents of the work area starting with the address specified and ending after the number of bytes specified for WIDTH in the RUNOPS line (see section 4.2). The optional print controls are listed in section below.
PRT<addr>[ctrl]	Immediately writes to the report file the contents of the work area starting with the address specified and ending after the number of bytes specified for WIDTH in the RUNOPS line (see section 4.2). This has no effect if no REPORT file was named (see section 3.1). The optional print controls are listed in section below.
SAY<addr>	Displays the contents of the work area beginning at the address specified on screen, for a length of 78 bytes, without pausing.
SELn[,m...]	Selects the current case to be written to the output streams n, m, etc. (see section 3.1.2). The case will actually be written by any subsequent WRT action (see below) or when the end of the spec file is reached for the current case, if not deselected before then. QGEN starts each case with OUTAP1 selected and all other output streams deselected, so this action must be used to send data to any other output files. If an output stream is not defined by a data directive, selecting it will have no effect.
RC'n'	Returns errorlevel 'n' at the end of the QGEN run (0 =< n =< 255) for testing in batch files. QGEN normally returns an errorlevel of 0 for a successful run and 12 if terminated with spec errors.
WRT	Immediately writes out the current record to any streams currently selected, then continues processing. QGEN writes each case to any files selected when it reaches the end of the specifications, regardless of any previous WRT commands, so streams should be deselected after this command to avoid unwanted duplicates.

7.4.1 Print Control Codes

QGEN provides a limited amount of formatting for text output sent to the REPORT or LISTING files. The spacing between lines of text can be adjusted by specifying one of the following print control codes immediately after the address of the output.

B	Write a blank line before writing this line.
C	Write a blank line after writing this line.
N	Do not issue a linefeed before writing this line, that is, overwrite the previous line written.
P	Issue a formfeed before writing this line.

7.5 Program Flow Actions

QGEN normally executes all the instructions in the spec file in sequence for each case, then writes it out and starts over at the top for the next case. The actions described below allow the user to control the order in which instructions are executed, based on the results of tests on the data for the current case.

Action	Description
BCH<tag>	Branch to subroutine named by tag, do not carry loop increments.
EOJ	Force an end of job, that is, stop the QGEN run after processing this case. A recycle pass will be executed if RECYCLE was called for in the RUNOPS (see section 4.2),
LNK<tag>	Link to subroutine named by tag, carry loop increments.
RET	Return from a subroutine.
SKP<n|tag>	Skip past the next n lines, or to the tag named.
SKPEND	Skip to the end of the spec file. No further instructions will be processed for the current case and any currently selected output streams will be written.

7.5.1 Subroutines

QGEN allows a group of instructions to be defined as a subroutine so that they can be used as many times as needed without being written out in full each time. A subroutine must begin with an instruction identified with a tag (see section 7.1) and must end with an RET action.

Program control is transferred by using the BCH or LNK actions with the tag that identifies the subroutine. When QGEN reaches a RET action, program control returns to the instruction line immediately following the BCH or LNK action that called the subroutine. Subroutines may be nested, that is, an instruction in one subroutine may call another subroutine.

QGEN does not pass arguments to subroutines, but the user can load data into locations in the work area that will be used by the subroutine. When a subroutine is called by an instruction within a loop (see section 9), the loop increments will carry through to the instructions in the subroutine if it is called by the LNK action. They will not carry to the subroutine if it is called by the BCH action, although they will still be in effect when control returns to the calling routine.

7.6 Cleaning Rules

Any instruction that evaluates logically may be used as a cleaning instruction by starting the line with a question mark (?). A cleaning instruction is given a rule name, up to 40 characters long, with no embedded spaces, entered in the result field. The question mark does not need to be in column 1 of the line and may be separated from the rule name by one or more spaces.

When the logical condition defined by a cleaning instruction evaluates false, the rule is broken and both the case ID for the current record and the rule name are written to the listing file. Breaking a rule also turns on switch 6 (see section 6.2.2) for the current record.

At the end of the listing file, QGEN provides a summary with the number of cleaning messages, the number of cases involved, and a breakdown by each rule that was actually broken during the run. If the number of cleaning messages exceed the maximum allowed by the MAXERR run option (see section 4.2), the run is terminated with the current record.

8 OPERATION CODES

Each QGEN instruction carries out some operation on the data or expressions specified in its operand fields, the exact nature of which is indicated by an operation code, or opcode.

One type of operation tests a condition among the operands and results in an action being taken if the condition tests true, or in a cleaning rule being broken if the condition tests false. Logical and comparison instructions fall into this category, as do the SPB and MP instructions on column-binary data.

A second type of operation uses the data in the locations specified in the operands as input to a procedure that results in new data being placed in a location specified in the result field of the instruction. Arithmetic, data conversion and table lookup instructions fall into this category, as do the remaining column-binary punch instructions.

Except for the program control instructions described in section 9, every QGEN instruction has a conditional form, where the first operand tests a condition, and the operation will be performed on the remaining operands only if that condition tested true. The conditional form is specified by prefixing the operation code with the letter C.

8.1 Logical Instructions

The following operations test for a condition between operands using Boolean logic. Operands must all be expressions that evaluate to true or false. The result can be either an action to be performed if the logical condition tests true, or a cleaning rule that will be broken if it tests false.

Code	# Args	Description
ALL	0	True for all cases (performs an action unconditionally).
AND	1-4	True if the conditions in all operands test true.
IF	1-4	Identical to "AND"
OR	1-4	True if the conditions in any one or more of the operands tests true.
EOR	2-4	True if one, and only one, of the operands tests true.

8.2 Arithmetic Instructions

The following operations perform arithmetic on the data specified by the operands. Operands must all be valid numeric fields or expressions that evaluate to a numeric value. The result is converted to the numeric format specified in the result field and placed there. Invalid numeric data will cause the instruction to fail, usually resulting in a "CANNOT CONVERT" error message in the listing, and no change will be made to the previous contents of the result location.

Code	# Args	Description
ADD	1-4	Add all operands: A+B[+C[+D]].
DIV	2	Divide the first operand by the second: A/B.
MPY	2-4	Multiply all operands together: A*B[*C[*D]].
MPD	3	Multiply the first two operands and divide by the third: (A*B)/C
SUB	2	Subtract the second operand from the first: A-B.
SQ	1	Compute the square root of the operand.

8.3 Comparison Instructions

The following operations test for a condition between operands by comparing the contents of the fields specified. The result can be either an action to be performed if the logical condition tests true, or a cleaning rule that will be broken if it tests false.

Opcodes beginning with F compare the contents of fields directly and all operands must specify the same data type and be of the same size.

Opcodes beginning with N compare numeric values and operands must be either valid numeric fields or expressions that evaluate to a numeric value. Operands do not need to be the same type or size as they are converted internally for comparison. Invalid numeric data will cause the instruction to fail, usually resulting in a "CANNOT CONVERT" error message in the listing, and no change will be made to the previous contents of the result location.

Code	# Args	Description
FEQ/**NEQ**	2-4	True if all operands are equal: A = B [= C [= D]]
FNE/**NNE**	2-4	True if no two operands are equal: A # B [# C [# D]]
FHE/**NHE**	2-4	True if the first operand is higher or equal to all remaining operands: A>=B[,C[,D]].
FHI/**NHI**	2-4	True if the first operand is higher than all remaining operands: A>B[,C[,D]]
FLE/**NLE**	2-4	True if the first operand is lower or equal to all remaining operands: A=<B[,C[,D]].
FLO/**NLO**	2-4	True if the first operand is lower than all remaining operands: A=<B[,C[,D]].
FRN/**NRN**	3	True if the first operand falls in the range defined by the remaining two operands, inclusive: B=<A=<C, where B<C.
FNR/**NNR**	3	True if the first operand does not fall in the range defined by the remaining two operands, inclusive: A<B or A>C, where B<C.

8.4 Column Binary Punch Instructions

The following operations require column-binary data in all operand fields, either by default in address modes B or E (see section 4.1) or by explicit specification in address modes C or D. Any reference to other data types will generate a spec error. MP and SPB test the number of punches contained in the data locations specified by the operands and result in an action if true. TAL counts the punches and returns a numeric value in the format specified in the result field. The remaining opcodes result in punch modifications to the locations specified in the result field.

Code	# Args	Description
MP	1-4	True if more than one punch is present among those specified in the operands (requires both column and punches in operands).
NEN	2-4	Performs a bitwise AND, one punch at a time, of the columns specified in the operands. Multi-column fields of equal length may be specified, generating a result field of that length, with matching columns in each field ANDed. No punches should be specified in either result or operands.
NET	2-4	Performs a bitwise OR, one punch at a time, of the columns specified in the operands. Multi-column fields of equal length may be specified, generating a result field of that length, with matching columns in each field ORed. No punches should be specified in either results or operands.
NEX	2	Performs a bitwise EOR (Exclusive OR), one punch at a time, of the columns specified in the operands. Multi-column fields of equal length may be specified, generating a result field of that length, with matching columns in each field ORed. No punches should be specified in either results or operands.
PMV	1	Move a set of punches in one column to a different set of punches in another column. Result and operand must be single columns, with the same number of punches specified for both.
SHI	1	Forces a column to single punch, with only highest punch kept. Both result and operand must be single columns, with no punches specified in either.
SLO	1	Forces a column to single punch, with only the lowest punch kept. Both result and operand must be single columns, with no punches specified in either.
SPB	1-4	True if no more than one punch is present among those specified in the operands (requires both column and punches in operands).
TAL	1-4	Tally (count) the punches specified across all operands and put the number found in the result field (requires both column and punches in operands).

8.5 Data Conversion Instructions

The following instructions take data from the locations specified by the operands and move it to the location specified in the result field, converting it from one format to another as indicated by the opcode. No testing is performed to verify that originating data are of the type specified.

The length of the data being converted is determined by the input, and only the starting location of the output should be specified in the result field. Note that in conversions between ASCII and column binary formats, an ASCII byte converts to a 2 byte column, and vice-versa.

Code	# Args	Description
ATB	1	Convert from ASCII to column-binary. Each ASCII byte converts to a 2-byte column. Maximum input 80 bytes.
BSA	1	Convert from column-binary to ASCII, keeping only alphanumeric characters, punches and blanks only. Each valid 2-byte column converts to one ASCII byte. Maximum input 80 columns.
BTA	1	Convert from column-binary to ASCII. Each valid 2-byte column converts to one ASCII byte and invalid multi-punch combinations generate an asterisk (*). Maximum input 80 columns.
DEC	1-4	Converts a list of column-binary punches to ASCII 1's and 0's in a series of successive bytes. Operands must be single columns with punches specified. The number of bytes output will be equal to the total number of punches specified across all operands.
ED	1	Formats an ASCII numeric field using a mask (see section 8.5.1)
ATE	1	Converts from ASCII to EBCDIC. Maximum input 80 bytes.
ETA	1	Converts from EBCDIC to ASCII. Maximum input 80 bytes.
MOV	1	Copies data from one location to another. Maximum input 250 bytes or 80 column-binary columns.
STR	1-4	Converts a list of column-binary punches to their ASCII values in a series of successive bytes. Operands must be single columns with punches specified. The number of bytes output will be equal to the total number of punches specified across all operands.

8.5.1 Formatting With Edit Masks

An edit mask is a literal string used with the ED operation code to format an ASCII numeric field. Within the mask, the letter X is used to mark the position of each digit. Standard numeric period and comma punctuation may be specified and leading zeroes are suppressed, except after a decimal point. Other characters will be placed as they appear in the mask. For example:

```
1001   ED   901c6'XXX,XXX'
```

The following examples illustrate the effect of various edit masks on numeric fields:

```
INPUT          MASK          OUTPUT
009999         'XXX,XXX'       9,999
-00999         'XXX,XXX'        -999
10000          'XXX.XX%'     100.00%
```

9 PROGRAM CONTROL INSTRUCTIONS

QGEN provides two instructions that do not interact with the data in the work area, but are used instead to control the execution of other instructions in a spec file. They are:

Code	# Args	Description
LOOP R-N	1-4	Repeat a range R of instructions N times. The range may be given as a number of lines to be included or as a tag identifying the last line of the range. Operands for a loop instruction are increments applied to the corresponding operands in the looped instructions at each new pass through the loop.
NOP	0	No Operation: a dummy instruction used to provide a target line for a program flow action or to end a loop range.

9.1 Loop Instructions

A loop instruction tells QGEN to perform a group of instructions (the range of the loop), then to return to the beginning of the range and repeat the same group of instructions with a specific increment applied to each operand and result field, and to do this a specified number of times.

The format of the loop instruction is:

<increment> LOOP <RR>-<nn|Zn> [increment] [increment] [increment] [increment]

The range always begins with the line immediately following the loop instruction. "RR" can be a number, in which case it specifies how many instruction lines are included in the range, or it can be a tag labeling the last instruction line to be included in the range. The number of times to loop can be a positive integer, or it can be a counter (see section 6.2.1).

Loops can increment either columns or punches using positive or negative integer values.

Column increments are integer numbers that tell by how many columns the location specified in an operand will be incremented in each pass of the loop. By default, each column translates to one byte in address modes C and D and to two bytes in address modes B and E (see section 6.1.1). Appending the letter "C" to a column increment forces it to count single bytes and appending the letter "B" forces it to count two bytes per column, regardless of address mode.

Punches may also be incremented in both column-binary and ASCII data. Punch increments are specified by the expression **PI**<n> where n indicates the number of punches to increment. Punches are always incremented in the sequence 1,2,3,4,5,6,7,8,9,0,X,Y (see section 6.3.2). Looping past the end of a column carries into the next column.

Increments are applied to any result fields or actions that address the work area, counters or switches. They are ignored for operands that contain numeric constants or lookup table names, and for program flow and output actions that do not address locations.

Here are some examples of loops:

`002 1-3  LOOP  001  PI1`	Loop 1 instruction 3 times. The result field is incremented by 2 columns at a time, the first operand by 1 column, and the second operand by one punch.
`0  A1-24 LOOP  -1b   8c`	Loop the block of instructions through the line tagged as _A1_ 24 times. The result field is not incremented, the first operand is decremented by 2 bytes, and the second incremented by 8 bytes at a time.

9.2 Loops and Program Flow

Loops may be nested, with inner loops being executed in full for each pass of the outer loop, and inner loop operands starting over from the values reached through outer loop increments.

QGEN allows skipping within the range of a loop, and a loop can always be terminated by skipping out of its range, but skipping into the range of a loop will cause a spec error.

QGEN provides two actions that call subroutines (see section 7.5.1). The BCH action calls a subroutine without carrying loop increments through to the instructions in the subroutine. This permits the user to load data into fixed locations under a loop, branch out of the loop to perform actions on that data, and then continue with the loop. The LNK also calls a subroutine, but it carries all loop increments through to the instructions in the subroutine, so that the subroutine functions exactly as if it appeared within the range of the loop.

While in loops, including nested loops, the pass count for the current execution level is copied to counter Z1000 (see section 6.2.1) so that its value is available to the user.

10 TABLE LOOKUPS

Table lookup instructions allow the contents of data locations to be compared against tables of pre-defined strings or values (arguments), returning a character string (function) when a match is found. Any number of tables may appear anywhere in a spec file, as long as they are defined before they are referenced in instructions. The function returned is always a character string.

10.1 Tables

The first line of every table is identified by the **TABLE** keyword and specifies the name by which the table will be referenced in lookup instructions, the type (character or numeric), the length of the function and argument fields, and the maximum number of elements the table may contain. An END statement (see section 2.2) signals the end of the table. Each line between the TABLE and END statements specifies a table entry.

The format of the TABLE statement is:

TABLE <name> [NUM] A=nn F=mm E=pp

The table name may be up to 8 characters in length and must be unique within a spec file. The keyword NUM is used to specify a numeric table, in which case all table arguments must be valid numeric values in strict ascending order. The remaining parameters specify the number of characters in the **A**rgument, the number of characters returned by the **F**unction, and the **E**stimated size, which sets the maximum number of entries allowed for the table.

If nn is the argument size and mm the function size specified for a table, the first nn characters of each table entry line are the argument. One character is skipped, then the following mm characters are the function returned by the entry.

In character type tables, any character string may be used for an argument. Case is preserved ("A" and "a" are not the same). Tables are checked from top to bottom. Duplicate arguments are not flagged as errors, but only the first occurrence will be found by the lookup instruction.

For tables identified as numeric by use of the NUM keyword, arguments are converted to floating point for comparison purposes and the TLN instruction must be used for lookups.

10.2 Table Lookup Instructions

Table lookup instructions always specify the name of a previously defined table as the second operand. The first operand specifies the argument that will be used to perform the lookup.

Code	# Args	Description
TLU	2	Looks up the character string in the location specified by the first operand in the character type table named in the second operand. If a matching entry is found, its function is returned in the location specified in the result field.
TLN	2	Looks up the exact value of the numeric field specified by the first operand in the numeric type table named in the second operand. If a matching entry is found, its function is returned in the location specified in the result field. The table must be defined as numeric.
TLR	2	Compares the character string in the location specified by the first operand to the ranges between the arguments of the character type table named in the second operand, using ASCII sequence, excluding lower bounds and including upper bounds. The function of the upper bound argument of the matching range is returned in the location specified in the result field.

The length of the ASCII string used as input to character table lookups is determined solely by the size of the argument specified for the table named in the instruction. The operand points to the location of the start of the string, and any ASCII field length specified there is ignored. Column binary fields can be specified directly as input to character table lookup instructions and QGEN will convert the contents of the field to ASCII internally before performing the lookup. In this case, the length specified in the TLU or TLR operand determines the number of characters converted and must match the argument size for the table, or results may not be as expected.

10.3 Examples of Table Lookups

The following example compares the results of TLR and TLU instructions using the same lookup table on the same argument. The result field is assumed blank before the lookup.

```
TABLE  ARANGE  A=5 F=1 E=5
00001 1
00010 2
00100 3
01000 4
99999 5
END
121   TLR   111   ARANGE
122   TLU   111   ARANGE
```

Input	TLR	TLU
00000	1	
00001	1	1
00005	2	
00010	2	2
00015	3	
00100	3	3
01000	4	4
01500	5	

The next example illustrates the conditional form of a numeric table lookup instruction. The first operand is the condition and tests for a numeric value in 491c4 in the range from 1 to 100. This test will fail if there is no valid number in that field, or if there is one and it is not in that range. The table lookup will only occur if the condition tests true, in which case the value will be looked up in the numeric type table named NTAB, and if a match is found, the function returned in 501.

```
501   CTLN   491c4r'1;100'    491c4    NTAB
```

11 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 QGEN spec file may contain any number of dictionaries anywhere before the table definition section.

11.1 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 anywhere in a QGEN spec file. QGEN does not use the second definition, which is used for text area substitution in those programs in the QUIP System that show annotated tables.

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. QGEN 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.

11.2 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
period      "121c4"     ""         Location of period code
current     "9803"      ""         Code for current period
prior       "9801"      ""         Code for prior period
END

SEL2&DRP1   AND  ##period@'##prior'
```

The result of this would be exactly as the instruction had been written:

```
SEL2&DRP1   AND  121C4@'9801'
```

QGEN parses the text immediately following the ## keyword for a valid label and continues until it finds the longest label already defined in any 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. See section 11.4 for an example of concatenation.

11.3 Built-in Dictionary Labels

QGEN 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 QGEN began execution, as it appears at the top of the listing, in the format: hh:mm (e.g., 15:30)

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

11.4 Command Line Parameters

QGEN 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 the definition for that dictionary entry. The format is:

QGEN 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 QGEN run:

```
QGEND QTDEMO.QGS  /# WAVE P /#                    <--- Command line
...
DICT
FILE_C  "3c3='710'"   ""   Current wave
FILE_P  "3c3='708'"   ""   Prior wave
END
...
? WRONG_WAVE   AND  ##FILE_##WAVE                     Check for correct wave
...
```

In this example, the argument for the command line parameter "WAVE" is "P" which will be substituted for ##WAVE in "##FILE_##WAVE" in the cleaning instruction to check the wave. QGEN will recognize this expression as the dictionary label FILE_P, and the spec definition for that dictionary entry will be substituted into the cleaning instructions. 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 spec error.

12 FLOW CONTROL AND BATCH PROCESSING

QGEN 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. QGEN can also be used as a data manipulation 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, 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 individual runs. QGEN also provides flow control features to assist with batch processing and process automation situations, and allow building modular libraries of spec files that can be selected and included into other files at run-time.

12.1 Includes

The include statement allows a spec file to call another spec file. QGEN 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>

Include statements may not appear within lookup tables or a dictionaries.

12.2 Stopping and Starting Specs and Listings

QGEN 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 take effect immediately and unconditionally, except for STARTSPEC IF, which applies only if the condition evaluates true. While STARTSPEC IF may be used with any dictionary entry, it is most effective when used with command line parameters (see section 11.4) for controlling the inclusion or exclusion of subsections of a spec file 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.qgs
END
STOPSPEC
STARTSPEC IF ##RUNCODE=B
#INCLUDE subfile1.qgs
END
STARTSPEC
```

13 WHERE TO GET ADDITIONAL INFORMATION

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

Jan Werner Data Processing

www.jwdp.com