CHAPTER

22

Reading Raw Data

Definition

raw data
is unprocessed data that has not been read into a SAS data set. You can use a DATA step to read raw data into a SAS data set from two sources:

- instream data
- an external file.

CAUTION:
Raw data does not include Database Management System (DBMS) files. You must license SAS/ACCESS software to access data stored in DBMS files. See Chapter 33, "Accessing Data in a DBMS," on page 487 for more information about SAS/ACCESS features. △

Ways to Read Raw Data

You can read raw data by using:

- SAS statements
- SAS functions
- External File Interface (EFI)
- Import Wizard.

When you read raw data with a DATA step, you can use a combination of the INPUT, DATALINES, and INFILE statements. SAS automatically reads your data when you use these statements. For more information on these statements, see "Reading Raw Data with the INPUT Statement" on page 290.

You can also use SAS functions to manipulate external files and to read records of raw data. These functions provide more flexibility in handling raw data. For a description of available functions, see the SAS functions listed under the "External File" and "SAS File I/O" categories in the functions category table in *SAS Language Reference: Dictionary*. For further information about how statements and functions manipulate files differently, see Chapter 6, "Functions and CALL Routines," on page 43.

If your operating environment supports a graphical user interface, you can use the EFI or the Import Wizard to read raw data. The EFI is a point-and-click graphical interface that you can use to read and write data that is not in SAS software's internal format. By using EFI, you can read data from an external file and write it to a SAS data set, and you can read data from a SAS data set and write it to an external file. See the SAS online Help for more information about EFI.

The Import Wizard guides you through the steps to read data from an external data source and write it to a SAS data set. As a wizard, it is a series of windows that present simple choices to guide you through a process. See the SAS online Help for more information on the wizard.

Operating Environment Information: Using external files with your SAS jobs requires that you specify file names with syntax that is appropriate to your operating environment. See the SAS documentation for your operating environment for more information. △

Kinds of Data

Definitions

data values
are character or numeric values.

numeric value
contains only numbers, and sometimes a decimal point and/or minus sign. When they are read into a SAS data set, numeric values are stored in the floating-point

format native to the operating environment. Nonstandard numeric values can contain other characters as numbers; you can use formatted input to enable SAS to read them.

character value
is a sequence of characters.

standard data
are character or numeric values that can be read with list, column, formatted, or named input. Examples of standard data include:

- **ARKANSAS**
- **1166.42**

nonstandard data
is data that can be read only with the aid of informats. Examples of nonstandard data include numeric values that contain commas, dollar signs, or blanks; date and time values; and hexadecimal and binary values.

Numeric Data

Numeric data can be represented in several ways. SAS can read standard numeric values without any special instructions. To read nonstandard values, SAS requires special instructions in the form of informats. Table 22.1 on page 287 shows standard, nonstandard, and invalid numeric data values and the special tools, if any, that are required to read them. For complete descriptions of all SAS informats, see *SAS Language Reference: Dictionary*.

Table 22.1 Reading Different Types of Numeric Data

Example of Numeric Data	Description	Solution Required to Read
Standard Numeric Data		
23	input right aligned	None needed
23	input not aligned	None needed
23	input left aligned	None needed
00023	input with leading zeroes	None needed
23.0	input with decimal point	None needed
2.3E1	in E-notation, 2.30 (ss1)	None needed
230E-1	in E-notation, 230x10 (ss-1)	None needed
-23	minus sign for negative numbers	None needed
Nonstandard Numeric Data		
2 3	embedded blank	COMMA. or BZ. informat
- 23	embedded blank	COMMA. or BZ. informat
2,341	comma	COMMA. informat
(23)	parentheses	COMMA. informat
C4A2	hexadecimal value	HEX. informat

Example of Numeric Data	Description	Solution Required to Read
`1MAR90`	date value	DATE. informat
	Invalid Numeric Data	
`23 -`	minus sign follows number	Put minus sign before number or solve programmatically.[1]
`..`	double instead of single periods	Code missing values as a single period or use the ?? modifier in the INPUT statement to code any invalid input value as a missing value.
`J23`	not a number	Read as a character value, or edit the raw data to change it to a valid number.

1 It might be possible to use the S370FZDT*w.d* informat, but positive values require the trailing plus sign (+).

Remember the following when reading numeric data:

- Parentheses or a minus sign preceding the number (without an intervening blank) indicates a negative value.
- Leading zeros and the placement of a value in the input field do not affect the value assigned to the variable. Leading zeros and leading and trailing blanks are not stored with the value. Unlike some languages, SAS does not read trailing blanks as zeros by default. To cause trailing blanks to be read as zeros, use the BZ. informat described in *SAS Language Reference: Dictionary*.
- Numeric data can have leading and trailing blanks but cannot have embedded blanks (unless they are read with a COMMA. or BZ. informat).
- To read decimal values from input lines that do not contain explicit decimal points, indicate where the decimal point belongs by using a decimal parameter with column input or an informat with formatted input. See the full description of the INPUT statement in *SAS Language Reference: Dictionary* for more information. An explicit decimal point in the input data overrides any decimal specification in the INPUT statement.

Character Data

A value that is read with an INPUT statement is assumed to be a character value if one of the following is true:

- A dollar sign ($) follows the variable name in the INPUT statement.
- A character informat is used.
- The variable has been previously defined as character: for example, in a LENGTH statement, in the RETAIN statement, by an assignment statement, or in an expression.

Input data that you want to store in a character variable can include any character. Use the guidelines in the following table when your raw data includes leading blanks and semicolons.

Table 22.2 Reading Instream Data and External Files Containing Leading Blanks and Semicolons

If your data contains ...	then use ...	because ...
leading or trailing blanks that you want to preserve	formatted input and the $CHAR*w*. informat	list input trims leading and trailing blanks from a character value before the value is assigned to a variable.
semicolons in instream data	DATALINES4 or CARDS4 statements and four semicolons (;;;;) to mark the end of the data	with the normal DATALINES and CARDS statements, a semicolon in the data prematurely signals the end of the data.
delimiters, blank characters, or quoted strings	DSD option, with DELIMITER= option on the INFILE statement	it enables SAS to read a character value that contains a delimiter within a quoted string; it can also treat two consecutive delimiters as a missing value and remove quotation marks from character values.

Remember the following when reading character data:

- In a DATA step, when you place a dollar sign ($) after a variable name in the INPUT statement, character data that is read from data lines remains in its original case. If you want SAS to read data from data lines as uppercase, use the CAPS system option.
- If the value is shorter than the length of the variable, SAS adds blanks to the end of the value to give the value the specified length. This process is known as padding the value with blanks.

Sources of Raw Data

Instream Data

The following example uses the INPUT statement to read in instream data:

```
data weight;
   input PatientID $ Week1 Week8 Week16;
   loss=Week1-Week16;
   datalines;
2477 195 177 163
2431 220 213 198
2456 173 166 155
2412 135 125 116
;
```

Note: A semicolon appearing alone on the line immediately following the last data line is the convention that is used in this example. However, a PROC statement, DATA statement, or global statement ending in a semicolon on the line immediately following the last data line also submits the previous DATA step. △

Instream Data Containing Semicolons

The following example reads in instream data containing semicolons:

```
data weight;
   input PatientID $ Week1 Week8 Week16;
   loss=Week1-Week16;
   datalines4;
24;77 195 177 163
24;31 220 213 198
24;56 173 166 155
24;12 135 125 116
;;;;
```

External Files

The following example shows how to read in raw data from an external file using the INFILE and INPUT statements:

```
data weight;
   infile file-specification or path-name;
   input PatientID $ Week1 Week8 Week16;
   loss=Week1-Week16;
run;
```

Note: See the SAS documentation for your operating environment for information on how to specify a file with the INFILE statement. △

Reading Raw Data with the INPUT Statement

Choosing an Input Style

The INPUT statement reads raw data from instream data lines or external files into a SAS data set. You can use the following different input styles, depending on the layout of data values in the records:

- list input
- column input
- formatted input
- named input.

You can also combine styles of input in a single INPUT statement. For details about the styles of input, see the INPUT statement in *SAS Language Reference: Dictionary*.

List Input

List input uses a scanning method for locating data values. Data values are not required to be aligned in columns but must be separated by at least one blank (or other

defined delimiter). List input requires only that you specify the variable names and a dollar sign ($), if defining a character variable. You do not have to specify the location of the data fields.

An example of list input follows:

```
data scores;
   length name $ 12;
   input name $ score1 score2;
   datalines;
Riley 1132 1187
Henderson 1015 1102
;
```

List input has several restrictions on the type of data that it can read:

- Input values must be separated by at least one blank (the default delimiter) or by the delimiter specified with the DELIMITER= option in the INFILE statement. If you want SAS to read consecutive delimiters as though there is a missing value between them, specify the DSD option in the INFILE statment.
- Blanks cannot represent missing values. A real value, such as a period, must be used instead.
- To read and store a character input value longer than 8 bytes, define a variable's length by using a LENGTH, INFORMAT, or ATTRIB statement prior to the INPUT statement, or by using modified list input, which consists of an informat and the colon modifier on the INPUT statement. See "Modified List Input" on page 291 for more information.
- Character values cannot contain embedded blanks when the file is delimited by blanks.
- Fields must be read in order.
- Data must be in standard numeric or character format.

Note: Nonstandard numeric values, such as packed decimal data, must use the formatted style of input. See "Formatted Input" on page 293 for more information △

Modified List Input

A more flexible version of list input, called modified list input, includes format modifiers. The following format modifiers enable you to use list input to read nonstandard data by using SAS informats:

- The & (ampersand) format modifier enables you to read character values that contain embedded blanks with list input and to specify a character informat. SAS reads until it encounters multiple blanks.
- The : (colon) format modifier enables you to use list input but also to specify an informat after a variable name, whether character or numeric. SAS reads until it encounters a blank column.
- The ~ (tilde) format modifier enables you to read and retain single quotation marks, double quotation marks, and delimiters within character values.

The following is an example of the : and ~ format modifiers:

```
data scores;
   infile datalines dsd;
   input Name : $9. Score1-Score3 Team ~ $25. Div $;
```

```
   datalines;
Smith,12,22,46,"Green Hornets, Atlanta",AAA
Mitchel,23,19,25,"High Volts, Portland",AAA
Jones,09,17,54,"Vulcans, Las Vegas",AA
;

proc print data=scores noobs;
run;
```

Output 22.1 Output from Example with Format Modifiers

Name	Score1	Score2	Score3	Team	Div
Smith	12	22	46	"Green Hornets, Atlanta"	AAA
Mitchel	23	19	25	"High Volts, Portland"	AAA
Jones	9	17	54	"Vulcans, Las Vegas"	AA

Column Input

Column input enables you to read standard data values that are aligned in columns in the data records. Specify the variable name, followed by a dollar sign ($) if it is a character variable, and specify the columns in which the data values are located in each record:

```
data scores;
   infile datalines truncover;
   input name $ 1-12 score2 17-20 score1 27-30;
   datalines;
Riley           1132       987
Henderson       1015      1102
;
```

Note: Use the TRUNCOVER option on the INFILE statement to ensure that SAS handles data values of varying lengths appropriately. △

To use column input, data values must be:

- in the same field on all the input lines
- in standard numeric or character form.

Note: You cannot use an informat with column input. △

Features of column input include the following:

- Character values can contain embedded blanks.
- Character values can be from 1 to 32,767 characters long.
- Placeholders, such as a single period (.), are not required for missing data.
- Input values can be read in any order, regardless of their position in the record.

- Values or parts of values can be reread.
- Both leading and trailing blanks within the field are ignored.
- Values do not need to be separated by blanks or other delimiters.

Formatted Input

Formatted input combines the flexibility of using informats with many of the features of column input. By using formatted input, you can read nonstandard data for which SAS requires additional instructions. Formatted input is typically used with pointer controls that enable you to control the position of the input pointer in the input buffer when you read data.

The INPUT statement in the following DATA step uses formatted input and pointer controls. Note that $12. and COMMA5. are informats and +4 and +6 are column pointer controls.

```
data scores;
   input name $12. +4 score1 comma5. +6 score2 comma5.;
   datalines;
Riley           1,132      1,187
Henderson       1,015      1,102
;
```

Note: You also can use informats to read data that is not aligned in columns. See "Modified List Input" on page 291 for more information. △

Important points about formatted input are:

- Characters values can contain embedded blanks.
- Character values can be from 1 to 32,767 characters long.
- Placeholders, such as a single period (.), are not required for missing data.
- With the use of pointer controls to position the pointer, input values can be read in any order, regardless of their positions in the record.
- Values or parts of values can be reread.
- Formatted input enables you to read data stored in nonstandard form, such as packed decimal or numbers with commas.

Named Input

You can use named input to read records in which data values are preceded by the name of the variable and an equal sign (=). The following INPUT statement reads the data lines containing equal signs.

```
data games;
   input name=$ score1= score2=;
   datalines;
name=riley score1=1132 score2=1187
;

proc print data=games;
run;
```

Note: When an equal sign follows a variable in an INPUT statement, SAS expects that data remaining on the input line contains only named input values. You cannot switch to another form of input in the same INPUT statement after using named input.

Also, note that any variable that exists in the input data but is not defined in the INPUT statement generates a note in the SAS log indicating a missing field. △

Additional Data-Reading Features

In addition to different styles of input, there are many tools to meet the needs of different data-reading situations. You can use options in the INFILE statement in combination with the INPUT statement to give you additional control over the reading of data records. Table 22.3 on page 294 lists common data-reading tasks and the appropriate features available in the INPUT and INFILE statements.

Table 22.3 Additional Data-Reading Features

If you want to read data that has ...	while ...	then use ...
multiple records	creating a single observation	#n or / line pointer control in the INPUT statement with a DO loop.
a single record	creating multiple observations	trailing @@ in the INPUT statement.
		trailing @ with multiple INPUT and OUTPUT statements.
variable-length data fields and records	reading delimited data	list input with or without a format modifier in the INPUT statement and the TRUNCOVER, DELIMITER= and/or DSD options in the INFILE statement.
	reading non-delimited data	$VARYING*w*. informat in the INPUT statement and the LENGTH= and TRUNCOVER options in the INFILE statement.
a file with varying record layouts		IF-THEN statements with multiple INPUT statements, using trailing @ or @@ as necessary.
hierarchical files		IF-THEN statements with multiple INPUT statements, using trailing @ as necessary.
more than one input file or to control the program flow at EOF		EOF= or END= option in an INFILE statement.
		multiple INFILE and INPUT statements.

If you want to read data that has ...	while ...	then use ...
		FILEVAR=option in an INFILE statement.
		FILENAME statement with concatentation, wildcard, or piping.
only part of each record		LINESIZE=option in an INFILE statement.
some but not all records in the file		FIRSTOBS=and OBS= options in an INFILE statement; FIRSTOBS= and OBS= system options; #n line pointer control.
instream datalines	controlling the reading with special options	INFILE statement with DATALINES and appropriate options.
starting at a particular column		@ column pointer controls.
leading blanks	maintaining them	$CHAR*w*. informat in an INPUT statement.
a delimiter other than blanks (with list input or modified list input with the colon modifier)		DELIMITER= option and/ or DSD option in an INFILE statement.
the standard tab character		DELIMITER= option in an INFILE statement; or the EXPANDTABS option in an INFILE statement.
missing values (with list input or modified list input with the colon modifier)	creating observations without compromising data integity; protecting data integrity by overriding the default behavior	TRUNCOVER option in an INFILE statement; DSD and/or DELIMITER= options might also be needed.

For further information on data-reading features, see the INPUT and INFILE statements in *SAS Language Reference: Dictionary*.

How SAS Handles Invalid Data

An input value is invalid if it:

- requires an informat that is not specified
- does not conform to the informat specified
- does not match the input style used; for example, if it is read as standard numeric data (no dollar sign or informat) but does not conform to the rules for standard SAS numbers
- is out of range (too large or too small).

Operating Environment Information: The range for numeric values is operating environment-specific. See the SAS documentation for your operating environment for more information. △

If SAS reads a data value that is incompatible with the type specified for that variable, SAS tries to convert the value to the specified type, as described in "How SAS Handles Invalid Data" on page 295. If conversion is not possible, an error occurs, and SAS performs the following actions:

- sets the value of the variable being read to missing or to the value specified with the INVALIDDATA= system option
- prints an invalid data note in the SAS log
- sets the automatic variable _ERROR_ to 1 for the current observation.
- prints the input line and column number containing the invalid value in the SAS log. If a line contains unprintable characters, it is printed in hexadecimal form. A scale is printed above the input line to help determine column numbers

Reading Missing Values in Raw Data

Representing Missing Values in Input Data

Many collections of data contain some missing values. SAS can recognize these values as missing when it reads them. You use the following characters to represent missing values when reading raw data:

numeric missing values
: are represented by a single decimal point (.). All input styles except list input also allow a blank to represent a missing numeric value.

character missing values
: are represented by a blank, with one exception: list input requires that you use a period (.) to represent a missing value.

special numeric missing values
: are represented by two characters: a decimal point (.) followed by either a letter or an underscore (_).

For more information about missing values, see Chapter 11, "Missing Values," on page 123.

Special Missing Values in Numeric Input Data

SAS enables you to differentiate among classes of missing values in numeric data. For numeric variables, you can designate up to 27 special missing values by using the letters A through Z, in either upper- or lowercase, and the underscore character (_).

The following example shows how to code missing values by using a MISSING statement in a DATA step:

```
data test_results;
   missing a b c;
   input name $8. Answer1 Answer2 Answer3;
   datalines;
Smith    2 5 9
```

```
Jones    4 b 8
Carter   a 4 7
Reed     3 5 c
;

proc print;
run;
```

Note that you must use a period when you specify a special missing numeric value in an expression or assignment statement, as in the following:

```
x=.d;
```

However, you do not need to specify each special missing numeric data value with a period in your input data. For example, the following DATA step, which uses periods in the input data for special missing values, produces the same result as the input data without periods:

```
data test_results;
   missing a b c;
   input name $8. Answer1 Answer2 Answer3;
   datalines;
Smith    2 5 9
Jones    4 .b 8
Carter   .a 4 7
Reed     3 5 .c
;

proc print;
run;
```

Output for both examples is shown in Output 22.2 on page 297.

Output 22.2 Output of Data with Special Missing Numeric Values

```
                        The SAS System

 Obs     name      Answer1    Answer2    Answer3

  1     Smith        2          5          9
  2     Jones        4          B          8
  3     Carter       A          4          7
  4     Reed         3          5          C
```

Note: SAS displays and prints special missing values that use letters in uppercase. △

Reading Binary Data

Definitions

binary data
is numeric data that is stored in binary form. Binary numbers have a base of two and are represented with the digits 0 and 1.

packed decimal data
are binary decimal numbers that are encoded by using each byte to represent two decimal digits. Packed decimal representation stores decimal data with exact precision; the fractional part of the number must be determined by using an informat or format because there is no separate mantissa and exponent.

zoned decimal data
are binary decimal numbers that are encoded so that each digit requires one byte of storage. The last byte contains the number's sign as well as the last digit. Zoned decimal data produces a printable representation.

Using Binary Informats

SAS can read binary data with the special instructions supplied by SAS informats. You can use formatted input and specify the informat in the INPUT statement. The informat you choose is determined by the following factors:

- the type of number being read: binary, packed decimal, zoned decimal, or a variation of one of these
- the type of system on which the data was created
- the type of system that you use to read the data.

Different computer platforms store numeric binary data in different forms. The ordering of bytes differs by platforms and is referred to as either "big endian" or "little endian." For a list of platforms considered big endian and little endian, see "Byte Ordering on Big Endian and Little Endian Platforms" on page 31.

SAS provides a number of informats for reading binary data and corresponding formats for writing binary data. Some of these informats read data in native mode, that is, by using the byte-ordering system that is standard for the system on which SAS is running. Other informats force the data to be read by the IBM 370 standard, regardless of the native mode of the system on which SAS is running. The informats that read in native or IBM 370 mode are listed in the following table.

Table 22.4 Informats for Native or IBM 370 Mode

Description	Native Mode Informats	IBM 370 Mode Informats
ASCII Character	$*w.*	$ASCII*w.*
ASCII Numeric	*w.d*	$ASCII*w.*
EBCDIC Character	$*w.*	$EBCDIC*w.*
EBCDIC Numeric (Standard)	*w.d*	S370FF*w.d*
Integer Binary	IB*w.d*	S370FIB*w.d*
Positive Integer Binary	PIB*w.d*	S370FPIB*w.d*
Real Binary	RB*w.d*	S370FRB*w.d*
Unsigned Integer Binary	PIB*w.d*	S370FIBU*w.d*, S370FPIB*w.d*
Packed Decimal	PD*w.d*	S370FPD*w.d*
Unsigned Packed Decimal	PK*w.d*	S370FPDU*w.d* or PK*w.d*
Zoned Decimal	ZD*w.d*	S370FZD*w.d*
Zoned Decimal Leading Sign	S370FZDL*w.d*	S370FZDL*w.d*

Description	Native Mode Informats	IBM 370 Mode Informats
Zoned Decimal Separate Leading Sign	S370FZDS*w.d*	S370FZDS*w.d*
Zoned Decimal Separate Trailing Sign	S370FZDT*w.d*	S370FZDT*w.d*
Unsigned Zoned Decimal	ZD*w.d*	S370FZDU*w.d*

If you write a SAS program that reads binary data and that will be run on only one type of system, you can use the native mode informats and formats. However, if you want to write SAS programs that can be run on multiple systems that use different byte-storage systems, use the IBM 370 informats. The IBM 370 informats enable you to write SAS programs that can read data in this format and that can be run in any SAS environment, regardless of the standard for storing numeric data.* The IBM 370 informats can also be used to read data originally written with the corresponding native mode formats on an IBM mainframe.

For complete descriptions of all SAS formats and informats, including how numeric binary data is written, see *SAS Language Reference: Dictionary*.

Reading Column-Binary Data

Definition

column-binary data storage
is an older form of data storage that is no longer widely used and is not needed by most SAS users. Column-binary data storage compresses data so that more than 80 items of data can be stored on a single punched card. The advantage is that this method enables you to store more data in the same amount of space. There are disadvantages, however; special card readers are required and difficulties are frequently encountered when this type of data is read. Because multi-punched decks and card-image data sets remain in existence, SAS provides informats for reading column-binary data. See “Description of Column-Binary Data Storage” on page 300 for a more detailed explanation of column-binary data storage.

How to Read Column-Binary Data

To read column-binary data with SAS, you need to know:

- how to select the appropriate SAS column-binary informat
- how to set the RECFM= and LRECL= options in the INFILE statement
- how to use pointer controls.

The following table lists and describes SAS column-binary informats.

* For example, using the IBM 370 informats, you could download data that contain binary integers from a mainframe to a PC and then use the S370FIB informats to read the data.

Table 22.5 SAS Informats for Reading Column-binary Data

Informat Name	Description
$CB*w*.	reads standard character data from column-binary files
CB*w*.	reads standard numeric data from column-binary files
PUNCH.*d*	reads whether a row is punched
ROW*w*.*d*	reads a column-binary field down a card column

To read column-binary data, you must set two options in the INFILE statement:

- Set RECFM= to F for fixed.
- Set the LRECL= to 160, because each card column of column-binary data is expanded to two bytes before the fields are read.

For example, to read column-binary data from a file, use an INFILE statement in the following form before the INPUT statement that reads the data:

```
infile file-specification or path-name
  recfm=f lrecl=160;
```

Note: The expansion of each column of column-binary data into two bytes does *not* affect the position of the column pointer. You use the absolute column pointer control @, as usual, because the informats automatically compute the true location on the doubled record. If a value is in column 23, use the pointer control @23 to move the pointer there. △

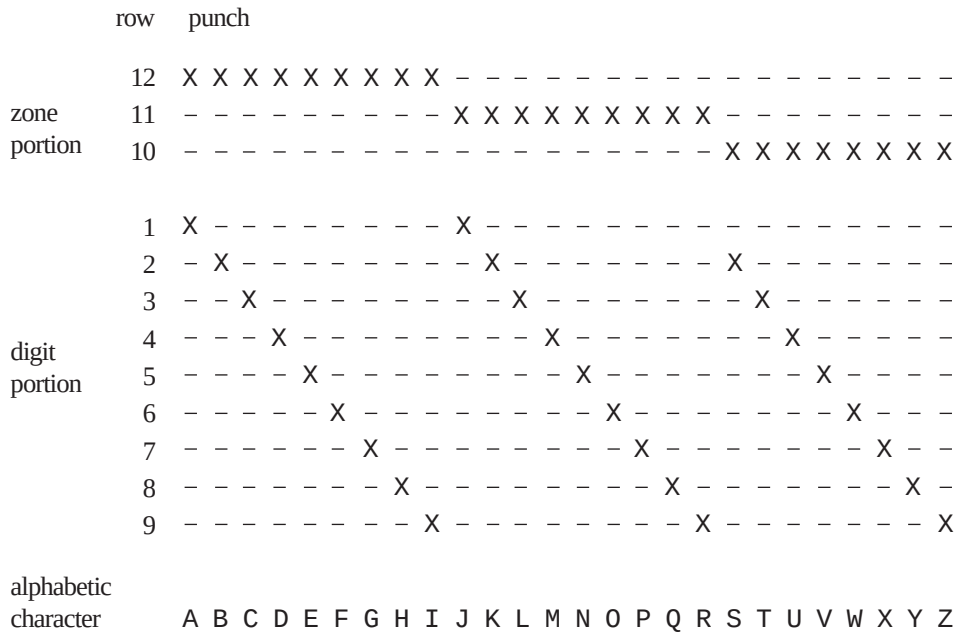
Description of Column-Binary Data Storage

The arrangement and numbering of rows in a column on punched cards originated with the Hollerith system of encoding characters and numbers. It is based on using a pair of values to represent either a character or a numeric digit. In the Hollerith system, each column on a card has a maximum of two punches, one punch in the zone portion, and one in the digit portion. These punches correspond to a pair of values, and each pair of values corresponds to a specific alphabetic character or sign and numeric digit.

In the zone portion of the punched card, which is the first three rows, the zone component of the pair can have the values 12, 11, 0 (or 10), or not punched. In the digit portion of the card, which is the fourth through the twelfth rows, the digit component of the pair can have the values 1 through 9, or not punched.

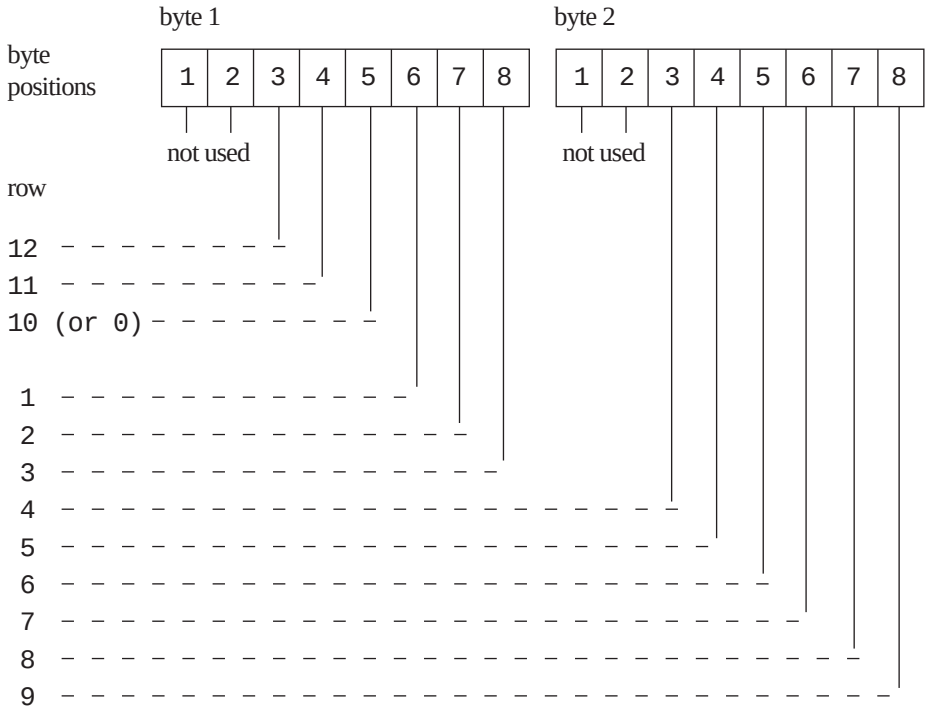
The following figure shows the multi-punch combinations corresponding to letters of the alphabet.

Figure 22.1 Columns and Rows in a Punched Card



SAS stores each column of column-binary data in two bytes. Since each column has only 12 positions and since 2 bytes contain 16 positions, the 4 extra positions within the bytes are located at the beginning of each byte. The following figure shows the correspondence between the rows of a punched card and the positions within 2 bytes that SAS uses to store them. SAS stores a punched position as a binary 1 bit and an unpunched position as a binary 0 bit.

Figure 22.2 Column-Binary Representation on a Punched Card



The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS Language Reference: Concepts*, Cary, NC: SAS Institute Inc., 1999. 554 pages.

SAS Language Reference: Concepts

ISBN 1–58025–441–1

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, November 1999