CHAPTER

7

DBF Chapter

Note to UNIX and OS/390 Users

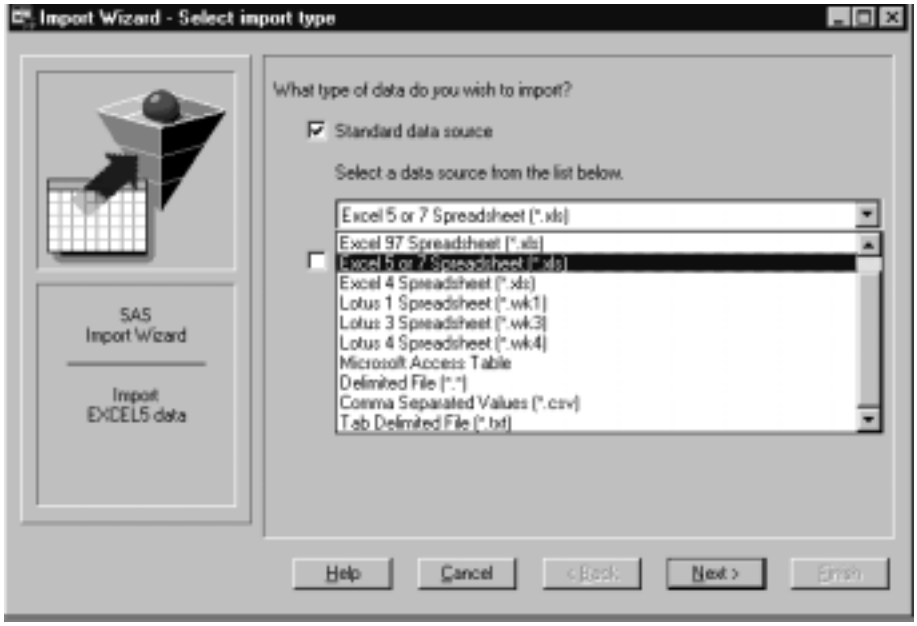
If you are running this SAS/ACCESS interface under the UNIX or OS/390 operating environment, this chapter does not apply to you. Instead, see Chapter 3, "DBF and DIF Procedures," on page 33. Under UNIX and PC hosts, you can use these procedures to convert a DBF or DIF file to a SAS data set or a SAS data set to a DBF or DIF file. Under OS/390, you can use PROC DBF *only* to convert a DBF file to a SAS data set or a SAS data set to a DBF file.

Import/Export Facility

UNIX and PC users can access DBF data through the Import/Export facility or by using the IMPORT and EXPORT procedures. An overview is included in Chapter 5, "Import/Export Facility and Procedures," on page 51.

To use the point-and-click interface from a SAS PROGRAM EDITOR window, select the **File** menu and then select the **Import Data** or **Export Data** item. Information about how to import or export DBF data is available from the Help button. The following is a sample Import window:

Display 7.1 Import Window



To write code to import or export DBF data, refer to the detailed descriptions of the IMPORT and EXPORT procedures in the *SAS Procedures Guide*. This documentation also includes several examples.

Understanding DBF Essentials

This chapter introduces SAS System users to DBF files, which can be created using a variety of microcomputer software programs. DBF files are a file format created by dBASE, a relational database management system for microcomputer systems.

This chapter focuses on the terms and concepts that help you access DBF files with SAS/ACCESS software. Then it describes DBF-specific statements you use in the ACCESS and DBLOAD procedures. Finally, it also contains a section on how the SAS/ACCESS interface works.

Note: The SAS/ACCESS interface cannot access DBF files created by Visual dBASE 7. △

The SAS/ACCESS interface works with DBF files that are created by dBASE (II, III, III PLUS, IV, and 5.0) and with DBF files that are created by other software products.

As an introduction to DBF files, this chapter describes DBF files that are created using dBASE 5.0, rather than describing each version of dBASE and the differences among them.* If you want more information on a dBASE concept or term, see the dBASE documentation packaged with your system.

DBF Files

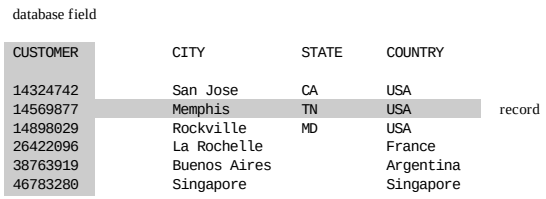
A DBF file contains data that are organized in a tabular format of database fields and records. Each *database field* can contain one type of data, and each *record* can hold

* The term dBASE refers to dBASE 5.0 for Windows unless otherwise noted.

one data value for each field. Figure 7.1 on page 99 illustrates four database fields from CUSTOMER.DBF and highlights a database field and a record.

The SAS/ACCESS interface uses database files that have a .DBF extension. A DBF file consists of a specific number of database fields and some number of records. DBF files are one kind of file that you can select in a catalog. You can create DBF files in a number of ways in dBASE, including using the CREATE command. See your dBASE or other software products' documentation for information about creating DBF files and assigning field names, field types, and other attributes.

Figure 7.1 DBF File



The ACCESS procedure uses SAS/ACCESS descriptor files to reference DBF files for reading or extracting data. It cannot use any dBASE indexes or indexes created by other software products that are defined on the fields in a DBF file. You can use the view descriptors you create to update DBF data. You can use the DBLOAD procedure to create and load DBF files.

The ACCESS procedure cannot reference DBF files that are secured through encryption. Like other files, DBF files are subject to any security restrictions imposed by the operating system or network (if applicable).

DBF File Naming Conventions

Filenames must also follow operating-system specific conventions, so check the documentation that comes with your dBASE product or other software products for further information. The following conventions apply to DBF filenames and field names:

- Under Windows 95, Windows 98, Windows NT, and OS/2, the ACCESS and DBLOAD procedures support long names that are specified in the PATH= statement (such as **`path='c:\sasdemo\library\customer99.dbf';`**) However, some applications that support dBASE files might not accept files with long names.
- Filenames or field names start with a letter, and they can contain any combination of the letters A through Z, the digits 0 through 9, the colon (:) (in dBASE II field names only), and the underscore (_).
- Database field names can be from one to ten characters long. Each field in a DBF file has a unique name.
- Filenames or field names are not case sensitive; that is, CUSTOMER is the same as Customer. Field names typed in lowercase are changed to uppercase on the display.

DBF File Data Types

Every field in a DBF file has a name and a data type. The data type tells how much physical storage to set aside for the database field and the form in which the data are stored. The following section lists and describes each data type.

Character(*N*)
specifies a field for character string data. The maximum length of *N* is 254 characters. Characters can be letters, digits, spaces, or special characters. You can abbreviate *character* to *char* in your programs.

Numeric(*N*,*n*)
specifies a packed decimal number, that is, a Binary Coded Decimal number. The *N* value is the total number of digits (precision), and the *n* value is the number of digits to the right of the decimal point (scale). The maximum values allowed depend on the software product you are using. For dBASE products, the maximum values allowed are

dBASE Version	*N*,*n*
dBASE II	16,14
dBASE III	19,15
dBASE III PLUS	19,15
dBASE IV	20,18
dBASE 5.0	20,18

Numeric field types always preserve the precision of their original numbers. However, the SAS System stores all numbers internally as double-precision, floating-point numbers, so their precision is limited to 16 digits.

Note: If every available digit in a DBF file field is filled with a **9**, the value of the field is interpreted as missing by the SAS System. If a field in the SAS System indicates a missing value (represented by a period), the SAS System writes a 9 for each available digit in the corresponding DBF file database field. While in a SAS session, if you fill every available digit in a DBF file field with 9s, scroll from the field, and return to the field, the value is represented as missing. △

Float(*N*,*n*)
specifies a floating-point binary number that is available in dBASE IV and later versions. The maximum *N*,*n* value for Float is 20,18. Check with the documentation that comes with other software products you may be using to create DBF files to determine if those products support floating-point binary numbers.

Date
specifies a date value in a format that has numbers and a character value to separate the month, day, and year. The default format is *mm*/*dd*/*yy*, for example, 02/20/95 for February 20, 1995.

Dates in DBF files can be subtracted from one another, with the result being the number of days between the two dates. A number (of days) can also be added to a date, with the result being a date.

Logical
specifies a type that answers a yes/no or true/false question for each record in a file. This type is 1 byte long and accepts the following character values: Y, y, N, n, T, t, F, and f.

dBASE also has data types called Memo, General, binary, and OLE, which are stored in an associated memo text file (called a DBT file), but these data types are not supported in the SAS/ACCESS interface to PC file formats.

"How the SAS/ACCESS Interface Works" on page 104 describes how the DBLOAD procedure determines data types when creating DBF files.

ACCESS Procedure Data Conversions

The table below shows the default SAS System variable formats that the ACCESS procedure assigns to each DBF file data type. If DBF file data fall outside of the valid SAS data ranges, you get an error message in the SAS log when you try to read the data.

DBF File Data Type	SAS Variable Format
Character(*n*)	$*n* .(*n* <= 200) $200. (*n* > 200)
Numeric(*N*,*n*)	(*N*,*n*)
Float(*N*,*n*)*	(*N*,*n*)
Date	MMDDYY8.
Logical	$1.

* This data type applies to dBASE IV and later. Check with other software products' documentation to see if this data type applies.

Handling Missing Values

Missing numeric values are filled in with 9s by default. The DBFMISCH is used to change the default by specifying the character that the interface to DBF files uses to fill missing numeric fields. For example, if you try to write a SAS file with a missing numeric variable to a DBF file, the corresponding field in the DBF file would be filled with the DBFMISCH character. Conversely, any numeric or float field in a DBF file that is filled with the DBFMISCH character is treated as missing when read by the SAS System.

You set the DBMISCH environment variable in the SAS configuration file using the following syntax:

-set DBFMISCH <*value*>

Valid values are

<*any single character*>
Type in any single character. For example, to fill missing numeric values with the zero character (0), enter **-set DBFMISCH 0**.

NULLS
To replace missing numeric values with binary zeros, enter **-set DBFMISCH NULLS**.

BLANKS
To replace missing numeric values with blanks, enter **-set DBFMISCH BLANKS**.

ACCESS Procedure: DBF Specifics

Chapter 2, "ACCESS Procedure Reference," on page 11 describes the generic options and procedure statements that enable you to create access descriptors, view descriptors, and SAS data files from PC file format data. The following section describes the PC file-specific statements that you use in the SAS/ACCESS interface to DBF files.

ACCESS Procedure Statements for DBF Files

To create an access descriptor, you use the DBMS=DBF option and the database-description statement PATH=. This PATH= statement supplies DBF-specific information to the SAS System and must immediately follow the CREATE statement. In addition to the database-description statements, you can use optional editing statements when you create an access descriptor. These editing statements must follow the database-description statements.

Database-description statements are only required when you create access descriptors. Because the DBF information is stored in an access descriptor, you do not need to repeat this information when you create view descriptors.

The SAS/ACCESS interface to DBF allows the following procedure statements:

Note: The SAS/ACCESS interface cannot read DBF files that are encrypted. Therefore, you cannot define an access descriptor based on these files. △

PROC ACCESS *options*;
CREATE *libref.name*.ACCESS|VIEW;
UPDATE *libref.name*.ACCESS|VIEW;

PATH= *'path-and-filename*<.DBF>'|<'>*filename*<'>| *fileref*;
ASSIGN | AN <=> YES | NO;
DROP <'>*column-identifier-1*<'> <...<'>*column-identifier-n*<'>>;
FORMAT <'>*column-identifier-1*<'><=>*SAS-format-name-1*
<...<'>*column-identifier-n*<'> <=>*SAS-format-name-n*>;
LIST <ALL | VIEW | <'>*column-identifier*<'>>;
RENAME <'>*column-identifier-1*<'><=>*SAS-variable-name-1*
<...<'>*column-identifier-n*<'><=>*SAS-variable-name-n*>;
RESET ALL | <'>*column-identifier-1*<'>
<...<'>*column-identifier-n*<'>>;
SELECT ALL | <'>*column-identifier-1*<'>
<...<'>*column-identifier-n*<'>>;
SUBSET *selection criteria*;
UNIQUE <=> YES | NO;

RUN;

The QUIT statement is also available in PROC ACCESS. However, its use causes the procedure to terminate. QUIT is used most often in the interactive line and noninteractive modes to exit the procedure without exiting SAS.

The following example creates an access descriptor and a view descriptor based on DBF file data.

```
options linesize=80;
libname dbfliba 'SAS-data-library';
libname dbflibv 'SAS-data-library';

proc access dbms=dbf;
/* create access descriptor */
   create adlib.custs.access;
   path='c:\dbfiles\dbcusts.dbf';
   assign=yes;
   rename customer = custnum;
   format firstorder date9.;
```

```
   list all;

/* create usacust view      */
   create vlib.usacust.view;
   select customer state zipcode name
          firstorder;
run;
```

DBLOAD Procedure: DBF Specifics

Chapter 4, "DBLOAD Procedure Reference," on page 41 describes the generic options and procedure statements that enable you to create a PC data file. The following section describes the file-specific statements you use in the SAS/ACCESS interface to DBF files.

DBLOAD Procedure Statements for DBF Files

To create and load a DBF table, the SAS/ACCESS interface to PC file formats uses the following statements:

PROC DBLOAD <DBMS=DBF> <DATA=<*libref.*>*SAS-data-set*>;
PATH=*'path-and-filename*<.DBF>' | <'>*filename*<'>|*fileref*;
VERSION= *dBASE-product-number*;
ACCDESC=<*libref.*>*access-descriptor*;
DELETE *variable-identifier-1* <*...variable-identifier-n*>;
ERRLIMIT= *error-limit*;
LABEL;
LIMIT= *load-limit*;
LIST <ALL | FIELDS | *variable-identifier*>;
LOAD;
RENAME *variable-identifier-1*= <'>*database-field-name-1*<'>
<*...variable-identifier-n* = <'>*database-field-name-n*<'>>;
RESET ALL | *variable-identifier-1* <*...variable-identifier-n*>;
TYPE *variable-identifier-1='database-field-type-1'*
<*...variable-identifier-n* = *'database-field-type-n*'>;
WHERE *SAS-where-expression*;
RUN;

The QUIT statement is also available in PROC DBLOAD. However, its use causes the procedure to terminate. QUIT is used most often in the interactive line and noninteractive modes to exit the procedure without exiting SAS.

VERSION= *dBASE-product-number*
specifies the number of the dBASE product you are using, such as dBASE IV. The *dBASE-product-number* argument can be one of the following values: II, III, IIIP, IV, V, 2, 3, 3P, 4, 5. The statement's default value is V.

Specify VERSION= before the TYPE statement in order to get the correct data types for your new .DBF table.

The following example creates a new .DBF table, EXCHANGE.DBF, from the data file DLIB.RATEOFEX . An access descriptor DBFLIBA.EXCHANGE is also created,

based on the new table. You must be granted the appropriate privileges in order to create new DBF tables.

```
libname dbfliba 'SAS-data-library';
libname dbflibv 'SAS-data-library';

proc dbload dbms=dbf data=dlib.rateofex;
   path='c:\dbfiles\sasdemo\exchange.dbf';
   accdesc=adlib.exchange;
   rename fgnindol=fgnindolar 4=dolrsinfgn;
   type country='char(25)';
   load;
run;
```

TYPE *variable-identifier-1 = 'database-field-name-1'*
<... variable-identifier-n = 'database-field-name-n'>;

specifies a DBF file data type, which is based on the SAS variable format. The database field name must be enclosed in quotation marks.

The following example defines the data types for several database fields. Notice that you can specify the length of the data type.

```
proc dbload dbms=dbf data=employee;
  path='c:\sasdemo\employee.dbf';
  rename firstname = fname;
  type   empid     = 'numeric(6)'
         hiredate  = 'date'
         salary    = 'numeric(10,2)'
         jobcode   = 'numeric(5)';
  run;
```

How the SAS/ACCESS Interface Works

For DBF files, the SAS/ACCESS interface is a *read-write interface*. When you use the ACCESS procedure to create an access descriptor, the SAS System retrieves descriptive information about the database fields directly from the DBF file. When you create a view descriptor, the SAS System retrieves information from the access descriptor without reading the DBF file again.

If the *structure* of a DBF file changes—for example, database fields are deleted—these changes do not appear in the access descriptor that you created with the ACCESS procedure. The changes also are not reflected in any view descriptors that created previously on that access descriptor and, therefore, invalidate the view descriptors.

However, if the *data* in the DBF file change, the updated data do appear when they are retrieved by a view descriptor. Suppose, for example, you have a view descriptor defined on a DBF file, and you add 30 records to that file. When you perform a SAS PRINT procedure using that view descriptor, both the old and new records are displayed.

To perform data manipulation tasks, the interface uses SAS commands and statements. For example, in the ACCESS procedure, you use the SAS WHERE statement to retrieve a subset of records from a DBF file. To sort DBF data, you must first extract the data into a SAS data file, unless you are using the SQL procedure. (The SQL procedure enables you to present output data in a sorted order with the ORDER BY clause in the SELECT statement without extracting the data.) You can extract and sort the DBF file data in one step using the OUT= option in the SORT procedure.

The SAS System does not use dBASE indexes or indexes created by other software products that are defined on fields in a DBF file. However, once you have extracted DBF file data with a view descriptor, you can use the SQL or DATASETS procedure to define SAS indexes on variables in the new SAS data file. Using SAS indexes often enhances the performance of data manipulation and retrieval tasks.

When you use the DBLOAD procedure to create and load a DBF file from a SAS data set, the procedure translates the SAS variable formats into field types that can be used in dBASE or other software products. It stores the file in the path specified by the PATH= statement so that dBASE and other software products can then read data from the newly created DBF file.

When you use a view descriptor in a DATA step to display or edit DBF file data, the SAS System's DBF file interface view engine reads from or writes to the DBF file that is stored in the path you specified.

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS/ACCESS® Software for PC File Formats: Reference, Version 8*, Cary, NC: SAS Institute Inc., 1999.

SAS/ACCESS® Software for PC File Formats: Reference, Verison 8

ISBN 1-58025–544–2

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

1st printing, October 1999