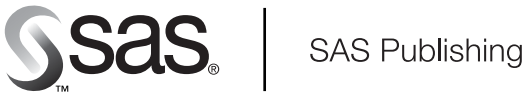


SAS/ACCESS® 9.1 Interface to PC Files

Reference

The Power to Know®

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2004. *SAS/ACCESS® 9.1 Interface to PC Files: Reference*. Cary, NC: SAS Institute Inc.

SAS/ACCESS® 9.1 Interface to PC Files: Reference

ISBN 1-59047-221-7

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

1st printing, January 2004

SAS Publishing provides a complete selection of books and electronic products to help customers use SAS software to its fullest potential. For more information about our e-books, e-learning products, CDs, and hard-copy books, visit the SAS Publishing Web site at **support.sas.com/pubs** or call 1-800-727-3228.

Contents

What's New

Overview

The differences between the PC and UNIX features for SAS/ACCESS for PC files have been significantly decreased. The documentation has also been enhanced with new sections added for these two operating environments.

Note:

- This section describes the features of SAS/ACCESS for PC files that are new or enhanced since SAS 8.2.

△

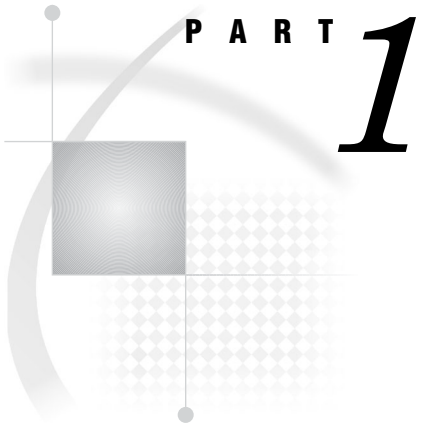
Windows Details

- The LIBNAME statement for PC files (Chapter 2, "The LIBNAME Statement for PC Files on Windows," on page 5) , new for SAS 9, provides direct, transparent access to Microsoft Access (97, 2000, and 2002) and Microsoft Excel (5, 95, 97, 2000, and 2002).
- PROC IMPORT and PROC EXPORT, using the SAS/ACCESS engine for PC files, provide direct access to JMP data files.
- PROC SQL, using capabilities of the new PC files engine, enables you to communicate with Microsoft Access and Microsoft Excel.
- Enhancements to the Import/Export wizard and procedures (Chapter 4, "The Import/Export Wizard and Procedures," on page 49) enable you to interact with JMP data files and to access multiple Microsoft Excel worksheets.

UNIX Details

- The SAS/ACCESS interface to PC Files on UNIX communicates with the PC files server (Chapter 9, "The PC Files Server," on page 105) . The server enables you to process requests for PC data.

- A new engine (Chapter 10, "The LIBNAME Statement for PC Files on UNIX," on page 109), called pcfiles, is available in SAS 9.1. This engine enables you to access data stored on network accessible PCs. This engine, working with new UNIX features, also enables you to directly access Microsoft Access (97, 2000, and 2002), Microsoft Excel (5, 95, 97, 2000, and 2002), and data from ODBC data sources on the PC (for example, the Microsoft SQL Server).
- Enhancements to PROC IMPORT and PROC EXPORT (Chapter 11, "The Import/Export Wizard and Procedures on UNIX," on page 117) enable you to access local JMP data files and remote JMP data files that are stored on the PC via the client/server model, in addition to Microsoft Access, Microsoft Excel, and ODBC data sources.
- The Pass-Through Facility for PC files (Chapter 12, "The Pass-Through Facility for PC Files on UNIX," on page 129) uses the pcfiles engine to communicate directly with Microsoft Access, Microsoft Excel, and ODBC data sources.



PART 1

Accessing PC Files

CHAPTER

1

Overview of the SAS/ACCESS Interface to PC Files

Methods for Accessing PC Files Data

SAS/ACCESS for PC files enables you to read data from PC files, to use that data in SAS reports or applications, and to use SAS data sets to create PC files in various formats. SAS/ACCESS for PC files includes the following features:

LIBNAME statement (UNIX and Windows operating environments)
provides direct, transparent access to Microsoft Access (97, 2000, or 2002) and Microsoft Excel (5, 95, 97, 2000, or 2002) data.

Pass-Through Facility (UNIX and Windows operating environments)
enables you to interact with Microsoft Access (97, 2000, or 2002) and Microsoft Excel (5, 95, 97, 2000, or 2002) data using the data source's SQL syntax without leaving your SAS session. The SQL statements are passed directly to the data source for processing.

Import/Export wizard and procedures (OpenVMS, UNIX, and Windows operating environments)
enable you to transfer data between SAS and several PC file formats including Microsoft Access, Microsoft Excel, Lotus 1-2-3, and DBF. Not every PC file format is available under every operating environment. See "Import/Export Overview for PC Files" on page 49 for a list of file formats supported under your operating environment.

DBF and DIF procedures (UNIX, Windows, and OS/390 operating environments)
enable you to convert between dBASE (DBF) files and SAS data sets and between data interchange format (DIF) files and SAS data sets. The DIF procedure is not available under OS/390.

ACCESS procedure (Windows operating environments)
creates descriptor files that describe data in a PC file to SAS, enabling you to directly read, update, or extract PC files data into a SAS data file. You can use the ACCESS procedure with the following file formats: Microsoft Excel (4, 5, 95), Lotus 1-2-3 (WK1, WK3, or WK4), DBF, and DIF.

DBLOAD procedure (Windows operating environments)
creates PC files and loads them with data from a SAS data set. You can use the DBLOAD procedure with any of the file formats that are supported by the ACCESS procedure.

Using This Document

This document is intended for applications programmers and users who know how to use their operating environment, and basic SAS commands and statements, and who are familiar with their PC file format.

This document provides both general reference and file format specific details about how to access data in PC file formats from SAS. It includes examples that demonstrate how you can use SAS/ACCESS software to read and write PC files data directly from SAS programs. The sample data that is used in the examples is provided in an appendix.

Sample Data in This Document

This document uses sample PC files that show you how to use the SAS/ACCESS interface to PC files. The PC files were created for a fictitious international textile manufacturer. This company's product line includes some special fabrics that are made to precise specifications. All the data in the files is fictitious.

Note: The files are designed to show how the SAS/ACCESS interface treats data stored in PC files. They are not meant as examples for you to follow in designing files for any purpose. △

Appendix 1, "Sample Data," on page 243 shows you the data in these sample PC files. The SAS/ACCESS software sample library contains the following files for your use. These files enable you to create the PC files and SAS/ACCESS descriptors and to run the examples.

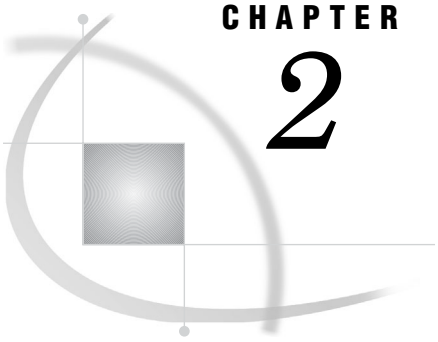
PcfFdbl.sas
: contains the DATA steps and PROC DBLOAD statements to create the PC files.

PcfFsamp.sas
: contains the SAS code of the examples in "Examples" on page 76.

PcfFmacs.sas
: contains macros that enable any SAS/ACCESS interface for a PC file format to create database description statements; these statements are used in the PROC DBLOAD and the PROC ACCESS code in PCDBL.SAS and PCSAMP.SAS files.

PcfFscl.sas
: contains the SAS Component Language (SCL) examples used in this document to create SAS/AF software examples.

These files are shipped with your SAS/ACCESS software. Check with your SAS system administrator or SAS Software Consultant for access to these files.



CHAPTER 2

The LIBNAME Statement for PC Files on Windows

Overview of the LIBNAME Statement for PC Files on Windows

The SAS/ACCESS LIBNAME statement extends the SAS global LIBNAME statement to support assigning a libref to Microsoft Excel and Microsoft Access files. This enables you to reference spreadsheets and databases directly in a DATA step or SAS procedure, and to read from and write to a Microsoft Access or Excel object as though it were a SAS data set.

Sorting PC Files Data

When you use the LIBNAME statement to associate a libref with PC files data, you might observe some behavior that differs from that of normal SAS librefs. Because these librefs refer to database and workbook objects, such as tables, they are stored in a format that differs from the format of normal SAS data sets. This is helpful to remember when you access and work with PC files data.

For example, you can sort the observations in a normal SAS data set and store the output to another data set. However, in a Microsoft Access database, sorting data has no effect on how it is stored. Because your data might not be sorted in the external file, you must sort the data at the time of query. Furthermore, when you sort PC files data, the results might vary depending on whether the external spreadsheet or database places data with NULL values (which are translated in SAS to missing values) at the beginning or the end of the result set.

Using SAS Functions with PC Files Data

When you use librefs that refer to PC files data with SAS functions, some functions might return a value that differs from what is returned when you use the functions with normal SAS data sets. For example, the PATHNAME function might return a Microsoft Excel filename assigned for the libref. For a normal SAS libref, it returns the pathname for the assigned libref.

Usage of some functions might also vary. For example, the LIBNAME function can accept an optional *SAS-data-library* argument. When you use the LIBNAME function to

assign or deassign a libref that refers to PC files data, you omit this argument. For full details about how to use SAS functions, see the *SAS Language Reference: Dictionary*.

Assigning a Libref Interactively

An easy way to associate a libref with PC files data is to use the New Library window. To open this window, issue the LIBASSIGN command from your SAS session's command box or command line. You can also access the New Library window by right-clicking on the libraries icon in the Explorer window and selecting New.

The following list describes how to use the New Library window:

- **`Name`**: enter the libref that you want to assign to a SAS data library or an external data source.
- **`Engine`**: click the down arrow to select a name from the pull-down listing.
- **`Enable at startup`**: click this if you want the specified libref to be assigned automatically when you open a SAS session.
- **`Library Information`**: these fields represent the SAS/ACCESS connection options and vary according to the SAS/ACCESS engine that you specify. Enter the appropriate information for your PC file format.
- OK: click this button to assign the libref, or click Cancel to exit the window without assigning a libref.

LIBNAME Statement Syntax for PC Files on Windows

Associates a SAS libref with a workbook or database

Valid in: anywhere

Syntax

❶ **LIBNAME** *libref* <*engine-name*> <*physical-file-name*>
<*SAS/ACCESS-engine-connection-options*>
<*SAS/ACCESS-libname-options*>;

❷ **LIBNAME** *libref* CLEAR | _ALL_ CLEAR;

❸ **LIBNAME** *libref* LIST | _ALL_ LIST;

Arguments

libref
is any SAS name that serves as an alias to associate SAS with a spreadsheet or database. Like the global SAS LIBNAME statement, the SAS/ACCESS LIBNAME statement creates shortcuts or nicknames for data storage locations. While a SAS libref is an alias for a virtual or physical directory, a SAS/ACCESS libref for PC files is an alias for the spreadsheet or database where your data is stored.

engine-name
is the SAS/ACCESS engine name for your PC file format. The engine name is optional if *physical-file-name* is specified. The SAS/ACCESS LIBNAME statement

associates a libref with a SAS/ACCESS engine that supports connections to a particular PC file. The following are the valid values for *engine-name*:

EXCEL for Microsoft Excel data (5, 95, 97, 2000, or 2002).

ACCESS for Microsoft Access data (97, 2000, or 2002).

physical-file-name
is the path and filename, including extension (.xls or .mdb), of the data source.

Note: If you omit *physical-file-name*, your engine connection options should identify the data source or you will be prompted for a filename, unless PROMPT=NO or NOPROMPT is indicated in the engine connection options. △

CLEAR
disassociates one or more currently assigned librefs.

Specify *libref* to disassociate a single libref. Specify _ALL_ to disassociate all currently assigned librefs.

ALL
specifies that the CLEAR or LIST argument applies to all currently-assigned librefs.

LIST
writes the attributes of one or more SAS/ACCESS libraries or SAS data libraries to the SAS log.

Specify *libref* to list the attributes of a single SAS/ACCESS library or SAS data library. Specify _ALL_ to list the attributes of all libraries that have librefs in your current session.

SAS/ACCESS-engine-connection-options
provide connection information to SAS/ACCESS to connect to your PC files. If the connection options contain characters that are not allowed in SAS names, enclose the values of the arguments in quotation marks. In some instances, if you specify the appropriate system options or environment variables for your data source, you can omit the connection options.

See “Connection Options” on page 7 for detailed information about connection options.

SAS/ACCESS-libname-options
define how SAS interacts with your data source, providing enhanced control of the way that SAS processes data source objects. For example, some LIBNAME options can improve performance. For many tasks, you do not need to specify any of these advanced options.

See “LIBNAME Statement Syntax for PC Files on Windows” on page 6 for detailed information about LIBNAME options.

Connection Options

SAS/ACCESS provides many ways to connect to your PC files.

DBPASSWORD="*database-file-password*"
enables you to access your file if you have database-level security set in your MDB file. A database password is case-sensitive and is defined in addition to user-level security.

Aliases: DBPWD=, DBPW=

Note: This connection option is only for Microsoft Access. Use of this option does not change your current security setting for your MDB file. △

DBSYSFILE="*workgroup-information-file*"
contains information about the users in a workgroup based on information that you define for you Microsoft Access database. Any user and group accounts or passwords you create are saved in the workgroup information file.

Alias: DBSYS=, WGDB=

Note: This connection option is only for Microsoft Access. Use of this option does not change your current security setting for your MDB file. △

HEADER=YES | NO
determines whether the first row of data in a Microsoft Excel range (or spreadsheet) are column names when SAS is reading data from a Microsoft Excel file.

Aliases: HDR=, GETNAMES=

YES — specifies to use the first row of data in an Excel range (or spreadsheet) as column names when SAS is reading data from an Excel file.

NO — specifies not to use the first row of data as column names in an Excel range (or spreadsheet) when SAS is reading data from an Excel file. SAS generates and uses the variable names F1, F2, F3 and so on.

Note: This connection option is only for Microsoft Excel. △

Note: This option is ignored when you are writing data to an Excel file. △

INIT= "*connection-string*"
specifies an initialization string that SAS uses when connecting to a data source. For example:

```
libname db ACCESS init="Provider=Microsoft.Jet.OLEDB.4.0
Data Source=c:\temp\sasdemo.mdb"
```

Alias: INIT_STRING=

Note: This option should not be used with a physical filename or other connection options, such as PATH= and UDL=. △

MIXED=YES | NO
specifies whether to convert numeric data values into character data values for a column with mixed data types. This option is valid only while you are importing data from Excel.

The default is NO, which means that numeric data will be imported as missing values in a character column. If MIXED=YES, the engine assigns a SAS character type for the column and convert all numeric data values to character data.

Alias: MIXED_DATA=, MIXED_DATATYPE=.

Note: The use of MIXED= option causes the Excel workbook to be locked in READONLY mode. No update is possible until the libref is deassigned. This option is not valid for accessing data in Microsoft Access database. △

PASSWORD="*user-password*"
specifies a password for the user account. A password can be 1 to 14 characters long and can include any characters except ASCII character 0 (null). Passwords are case-sensitive.

Aliases: PWD=, PW=

Note: This connection option is only for Microsoft Access. Use of this option does not change your current security setting for your MDB file. △

PATH="*path-for-file*"

specifies the data source file. This is the full path and filename for your Microsoft Access database file or Microsoft Excel workbook file. This option value is treated the same as the physical filename and should only be used when the physical filename is not specified in the LIBNAME statement. However, use of this option requires the engine name to be specified. Always enter file extension .mdb for Microsoft Access and .xls for Excel.

Alias: DATASRC=, DS=

PROMPT=YES | NO |REQUIRED | NOPROMPT | PROMPT | UDL
determines whether you are prompted for connection information that supplies the data source information.

YES — enables you to be prompted with a Data Link Properties dialog box. To write the initialization string to the SAS log, submit the following code immediately after connecting to the data source:

```
%put %superq (SYSDBMSG);
```

NO — does not allow you to be prompted with a Data Link Properties dialog box, and you are required to specify the data source (physical filename or PATH=).

REQUIRED — enables you to connect without prompting for more information only if a valid physical filename is specified for a successful connection. Otherwise, you are prompted for the connection options with a dialog box that enables you to change the data source file and other properties.

NOPROMPT — disables the prompt of the Data Link Properties dialog box.

PROMPT — enables you to be prompted for connection information that supplies the data source information.

UDL — enables you to browse and select an existing data link file (.udl).

UDL="*path-for-udl-file*"
specifies the path and filename for a UDL file (a Microsoft data link file). For example, you could specify

```
UDL=''C:\WinNT\profiles\me\desktop\MyDBLink.UDL'';
%put %superq(SYSDBMSG);
```

This option does not support SAS filerefs. Macro variable SYSDBMSG is set on successful completion. For more information, see Microsoft's documentation about using data link.

Alias: UDL_FILE=

Note: This option should not be used with a physical file name or other connection options, such as PATH= and INIT=. △

USER="*user-ID*"
specifies a user account name. User names can be 1 to 20 characters long and can include alphabetic characters, accented characters, numbers, and spaces. If you have user-level security set in your .mdb file, you need to use this option and the PASSWORD= option to be able to access your file.

Alias: UID=, USERID=

Note: This connection option is only for Microsoft Access. Use of this option does not change your current security setting for your MDB file. △

VERSION=2002 | 2000 | 97 | 95 | 5
sets the version of Microsoft Excel. The default value is 97.

Alias: VER=

Note: This connection option is only for Microsoft Excel. △

Note: You do not need to specifiy this option if you do not know the version of your Microsoft Excel file. However, if you want to create a new Microsoft Excel file, you can use this option to specify the version you want to create. △

2002	sets the version of Microsoft Excel to 2002.
2000	sets the version of Microsoft Excel to 2000.
97	sets the version of Microsoft Excel to 97.
95	sets the version of Microsoft Excel to 95.
5	sets the version of Microsoft Excel to 5.

The following example assigns the libref Db for an Excel file:

```
libname db 'c:\demo.xls';
```

The following example prompts you for data source information:

```
libname db excel;
libname db excel prompt= yes;
```

The following example prompts you for the UDL file:

```
libname db excel prompt=udl;
```

The following example uses the connection string to connect to the data source:

```
libname db excel init=''connection_string'';
```

Details

❶ Using Data from a PC File You can use a LIBNAME statement to read from and write to a data source table or view as though it were a SAS data set. The LIBNAME statement associates a libref with a SAS/ACCESS engine to access tables or views in a spreadsheet or database. The SAS/ACCESS engine enables you to connect to a particular data source and to specify an external data object name in a two-level SAS name.

For example, in MyPCLib.Employees_Q2, MyPCLib is a SAS libref that points to a particular group of external data objects, and Employees_Q2 is a table name. When you specify MyPCLib.Employees_Q2 in a DATA step or procedure, you dynamically access the external data object. SAS supports reading, updating, creating, and deleting external data objects dynamically.

❷ Disassociating a Libref from a SAS Data Library To disassociate or clear a libref, use a LIBNAME statement, specifying the libref (for example, MyPCLib) and the CLEAR option as follows:

```
libname mypclib CLEAR;
```

You can clear a single specified libref or all current librefs.

SAS/ACCESS disconnects from the data source and closes any free threads or resources that are associated with that libref's connection.

❸ Writing SAS Data Library Attributes to the SAS Log Use a LIBNAME statement and the LIST option to write the attributes of one or more SAS/ACCESS libraries or SAS data libraries to the SAS log. Specify *libref* to list the attributes of a single SAS/ACCESS library or SAS data library, as follows:

```
libname mypclib LIST;
```

Specify _ALL_ to list the attributes of all libraries that have librefs in your current session.

Examples

Assigning a Libref with a SAS/ACCESS LIBNAME Statement The following statement creates a libref, mymdb, as a Microsoft Access database file:

```
libname mymdb "c:\demo.mdb";
```

The Demo.mdb database contains a number of objects, including several tables, such as Staff. After you assign the libref, you can reference the Microsoft Access table like a SAS data set and use it as a data source in any DATA step or SAS procedure. In the following PROC SQL statement, MyMdb.Staff is the two-level SAS name for the Staff table in the Microsoft Access database Demo.

```
proc sql;
  select idnum, lname
    from mymdb.staff
    where state='NY'
    order by lname;
quit;
```

You can use the Microsoft Access data to create a SAS data set:

```
data newds;
   set mymdb.staff(keep=idnum lname fname);
run;
```

You can also use the libref and data set with any other SAS procedure. This statement prints the information in the Staff table:

```
proc print data=mymdb.staff;
run;
```

This statement lists the database objects in the MyMdb library:

```
proc datasets library=mymdb;
quit;
```

The following statement associates the SAS libref MYXLS with an Excel workbook:

```
libname myxls "c:\demo.xls";
```

See Also

"Overview of the LIBNAME Statement for PC Files on Windows" on page 5
"LIBNAME Statement Data Conversions for MDB Files" on page 169
"LIBNAME Statement Data Conversions for XLS Files" on page 152

LIBNAME Options for PC Files on Windows

The following LIBNAME statement options provide enhanced control over the way that SAS processes PC files data. For many tasks, you do not need to specify any of these advanced options.

Many of these options are also available as data set options.

ACCESS=READONLY
indicates that tables and views can be read but not updated.

AUTOCOMMIT=YES | NO
determines whether the ACCESS engine commits (saves) updates as soon as the user submits them.

YES
specifies that updates are committed to a table as soon as they are submitted, and no rollback is possible.

NO
specifies that the SAS/ACCESS engine automatically performs the commit when it reaches the end of the file.

Default: NO

COMMAND_TIMEOUT=*number-of-seconds*
specifies the number of seconds that pass before a data source command times out.

Default: 0 (no timeout)

Alias: TIMEOUT=

CONNECTION= SHAREDREAD | UNIQUE |GLOBALREAD
determines whether operations against a single libref share a connection to the data source. Also determines whether operations against multiple librefs share a connection to the data source.

SHAREDREAD
specifies that all READ operations that access data source tables *in a single libref* share a single connection. A separate connection is established for each table that is opened for update or output operations.

Where available, this is usually the default value because it offers the best performance and it guarantees data integrity.

UNIQUE
specifies that a separate connection is established every time a data source table is accessed by your SAS application.

Use UNIQUE if you want each use of a table to have its own connection.

GLOBALREAD
specifies that all READ operations that access data source tables *in multiple librefs* share a single connection if the following conditions are met:

- the participating librefs are created by LIBNAME statements that specify identical values for the CONNECTION= and CONNECTION_GROUP= options
- the participating librefs are created by LIBNAME statements that specify identical values for any data source connection options.

A separate connection is established for each table that is opened for update or output operations.

GLOBALREAD is the default value for CONNECTION= when you specify CONNECTION_GROUP=.

Default: SHAREDREAD

CONNECTION_GROUP= *connection-group*
causes operations against multiple librefs to share a connection to the data source. Also causes operations against multiple Pass-Through Facility CONNECT statements to share a connection to the data source.

CURSOR_TYPE=KEYSET_DRIVEN | STATIC
specifies the cursor type for read-only and updatable cursors. If CURSOR_TYPE= is not set, then the default cursor type is determined by the Jet provider you are using.

KEYSET_DRIVEN
specifies that the cursor determines which rows belong to the result set when the cursor is opened. However, changes that are made to these rows are reflected as you move the cursor. The OLE DB property DBPROP_OTHERUPDATEDELETE is set as TRUE for keyset cursors and FALSE for static cursors.

STATIC
specifies that the complete result set is built when the cursor is opened, but no changes made to the result set will be reflected in the cursor. Static cursors are read-only.

Default: none

Alias: CURSOR=

DBCOMMIT=*number-of-rows*
affects update, delete, and insert processing. The number of rows that are processed includes rows that are not processed successfully. If you set DBCOMMIT=0, a commit is issued only once (after the procedure or DATA step completes). If the DBCOMMIT= option is explicitly set, SAS/ACCESS fails any update that has a WHERE clause.

Note: If you specify both DBCOMMIT= and ERRLIMIT=, and these options collide during processing, then the commit is issued first and the rollback is issued second. Because the commit (caused by the DBCOMMIT= option) is issued prior to the rollback (caused by the ERRLIMIT= option), the DBCOMMIT= option is said to override the ERRLIMIT= option in this situation. △

Default: 1,000 (inserting) or 0 (updating; commit occurs when data set or procedure completes)

DBGEN_NAME=DBMS | SAS
specifies that the data source columns are renamed and the format the names will follow.

DBMS
specifies that the data source columns are renamed to valid SAS variable names. Disallowed characters are converted to underscores. If a column is converted to a name that already exists, then a sequence number is appended to the end of the new name.

SAS
specifies that data source columns are renamed to the format _COL*n*, where *n* is the column number (starting with zero).

Default: DBMS

DBMAX_TEXT=*n*
specifies an integer between 1 and 32,767 that indicates the maximum length for a character string. Longer character strings are truncated. This option only applies when you are reading, appending, and updating character data in a Microsoft Access database or Excel workbook from SAS.

Note: Although you may specify a value less than 256, it is not recommended for reading data from Microsoft Access Database. △

Default: 1,024

DBNULLKEYS=YES | NO
specifies column definitions.

YES
If there might be NULL values in the transaction table or the master table for the columns that you specify in the DBKEY= option, use DBNULLKEYS=YES. When you specify DBNULLKEYS=YES and specify a column that is not defined as NOT NULL in the DBKEY= data set option, SAS generates a WHERE clause that can find NULL values. For example, if you specify DBKEY=COLUMN and COLUMN is not defined as NOT NULL, SAS generates a WHERE clause with the following syntax:

```
WHERE ((COLUMN = ?) OR ((COLUMN IS NULL) AND (? IS NULL)))
```

This syntax enables SAS to prepare the statement once and use it for any value (NULL or NOT NULL) in the column. Note that this syntax has the potential to be much less efficient than the shorter form of the WHERE clause (presented below).

NO
When you specify DBNULLKEYS=NO or specify a column that is defined as NOT NULL in the DBKEY= option, SAS generates a simple WHERE clause. If you know that there are no NULL values in the transaction table or the master table for the columns that you specify in the DBKEY= option, then you can use DBNULLKEYS=NO. If you specify DBNULLKEYS=NO and specify DBKEY=COLUMN, SAS generates a shorter form of the WHERE clause (regardless of whether or not the column specified in DBKEY= is defined as NOT NULL):

```
WHERE (COLUMN = ?)
```

Default: YES

DBSASLABEL=COMPAT | NONE
specifies whether SAS/ACCESS saves the data source column names as SAS label names. This option is valid only when you are reading data into SAS from the data source.

COMPAT
specifies that the data source column names are saved as SAS label names. This is compatible to the previous SAS releases.

NONE
specifes that the data source column names are not saved as SAS label names. SAS label names are left as NULLs.

Default: COMPAT

DEFER=NO | YES
enables you to specify when the connection to the data source occurs.

NO
specifies that the connection to the data source occurs when the libref is assigned by a LIBNAME statement.

YES
specifies that the connection to the data source occurs when a table in the data source is opened.

Default: NO

DIRECT_SQL=YES | NO | NONE | *specific-functionality*

enables you to specify whether generated SQL is passed to the data source for processing.

YES
specifies that, whenever possible, generated SQL, except multiple outer joins, is passed to the data source for processing. This includes SQL that is generated from PROC SQL, SAS functions that can be converted into data source functions, joins, and WHERE clauses.

NO
specifies that generated SQL from PROC SQL is not passed to the data source for processing. This is the same as specifying the *specific-functionality* value NOGENSQL.

NONE
specifies that generated SQL is not passed to the data source for processing. This includes SQL that is generated from PROC SQL, SAS functions that can be converted into data source functions, joins, and WHERE clauses.

specific-functionality
identifies types of processing to be handled by SAS instead of the data source. You can specify the following values:

NOFUNCTIONS
causes SAS to handle all SAS functions. The SAS functions are not converted into data source functions and are not passed to the data source for processing.

NOMULTOUTJOINS
causes SAS to process outer joins that involve more than two tables. This option does not affect outer joins of two tables.

Note: This option is always turned ON for the Jet libname engine. △

NOGENSQL
prevents PROC SQL from generating SQL to be passed to the data source for processing.

NOWHERE
prevents WHERE clauses from being passed to the data source for processing. This includes SAS WHERE clauses and PROC SQL generated or PROC SQL specified WHERE clauses.

Default: YES

INSERTBUFF=*number-of-rows*
specifies the number of rows for a multiple-rows insert. The value for INSERTBUFF= must be a positive number. If the INSERTBUFF= value is greater than the DBCOMMIT= value, the DBCOMMIT= value will override it.

Note: When you assign a value that is greater than INSERTBUFF=1, the SAS application notes that indicate the success or failure of the insert operation might be incorrect because these notes only represent information for a single insert, even when multiple inserts are performed. △

Default: 1

READBUFF=*number-of-rows*
specifies the number of rows to use when you are reading data from a data source. Setting a higher value for this option reduces I/O and increases performance, but also increases memory usage. Additionally, if too many rows are read at once, values returned to SAS might be out of date.

Default: 1

Alias: ROWSET=
ROWSET_SIZE=

SCAN_TEXTSIZE= YES | NO
specifies whether to scan the length of text data for a data source column and use the length of the longest string data found as the SAS column width.

YES
scans the length of text data for a data source column and use the length of the longest string data found as the SAS variable width. However, if the maximum length found is greater than what is specified in the DBMAX_TEXT= option, the smaller value specified in DBMAX_TEXT= will be applied as the SAS variable width.

For Microsoft Excel, this option applies to all character data type columns. For Microsoft Access, this only applies to the MEMO data type field and does not apply to the TEXT (less than 256 characters long) field.

NO
does not scan the length of text data for a data source column. The column length returned from Microsoft Jet provider will be used as the SAS variable width. However, if the returned column width is greater than what is specified in the DBMAX_TEXT= option, the smaller value specified in DBMAX_TEXT= will be applied as the SAS variable width.

Note: Specify SCANTEXT=NO when you need to update data in the Microsoft Access database or Excel workbook. △

Default: YES for Microsoft Excel workbook
NO for Microsoft Access database

Alias: SCAN_TEXT=, SCANTEXT=, SCANMEMO=

SCAN_TIMETYPE=YES | NO
specifies whether to scan all row values for a DATETIME data type field and automatically determine the TIME data type if only time values (that is, no date or datetime values) exist in the column.

YES
specifies that a column with only time values will be assigned a TIME8. format.

NO
specifies that a column with only time values will be assigned a DATE9. format or DATETIME19 format. Please refer to USE_DATETYPE= option for more information.

Default: NO

Alias: SCAN_TIME=, SCANTIME=

SPOOL=YES | NO
specifies whether SAS creates a utility spool file during read transactions that read data more than once.

YES
specifies that SAS creates a utility spool file into which it writes the rows that are read the first time. For subsequent passes through the data, the rows are read from the utility spool file rather than being reread from the data source table. This guarantees that the row set is the same for every pass through the data.

NO

specifies that the required rows for all passes of the data are read from the data source table. No spool file is written. There is no guarantee that the row set is the same for each pass through the data.

Default: YES

STRINGDATES=YES | NO
specifies whether datetime values are read from the data source as character strings or as numeric date values. STRINGDATES= is not available as a data set option.

YES
specifies that SAS/ACCESS reads datetime values as character strings.

NO
specifies that SAS/ACCESS reads datetime values as numeric date values.

Default: NO

Alias: STRDATES=

USE_DATETYPE=YES | NO
specifies whether to use DATE. format for datetime columns in the data source table while importing data from Microsoft Access database or Excel workbook.

YES
specifies that the SAS DATE format is assigned for datetime columns in the data source table.

NO
specifies that the SAS DATETIME format is assigned for datetime columns in the data source table.

Default: YES for Microsoft Excel workbook
NO for Microsoft Access database

Alias: USE_DATE=, USEDATE=

Data Set Options for PC Files on Windows

You can specify SAS/ACCESS data set options on a SAS data set when you access PC files data with the LIBNAME statement. A data set option applies only to the data set on which it is specified, and it remains in effect for the duration of the DATA step or procedure.

The following generic example illustrates the format of data set options:

```
LIBNAME libref engine-name;
PROC PRINT libref.data-set-name(DATA_SET_OPTION=value)
```

You can use the CNTLLEV=, DROP=, FIRSTOBS=, IN=, KEEP=, OBS=, RENAME=, and WHERE= SAS data set options when you access PC files data. The REPLACE= SAS data set option is not supported by SAS/ACCESS interfaces. For information about using SAS data set options, refer to the *SAS Language Reference: Dictionary*.

Note: Specifying data set options in PROC SQL might reduce performance, because it prevents operations from being passed to the data source for processing. △

COMMAND_TIMEOUT=

Specifies the number of seconds to wait before a command times out

Valid in: DATA and PROC steps (when accessing PC files data using SAS/ACCESS software)

Default value: LIBNAME option setting

Syntax

COMMAND_TIMEOUT=*number-of-seconds*

See Also

To assign this option to a group of tables, use the COMMAND_TIMEOUT= option specified in "LIBNAME Options for PC Files on Windows" on page 11.

CURSOR_TYPE=

Specifies the cursor type for read-only and updatable cursors

Valid in: DATA and PROC steps (when accessing PC files data using SAS/ACCESS software)

Default value: LIBNAME option setting

Syntax

CURSOR_TYPE=KEYSET_DRIVEN |STATIC

Syntax Description

KEYSET_DRIVEN
specifies that the cursor determines which rows belong to the result set when the cursor is opened. However, changes that are made to these rows are reflected as you move the cursor.

STATIC
specifies that the cursor builds the complete result set when the cursor is opened. No changes made to the rows in the result set after the cursor is opened are reflected in the cursor. Static cursors are read-only.

Details

By default, this option is not set and the Microsoft Jet provider uses a default. The OLE DB properties applied to an open row set are as follows:

CURSOR_TYPE=	OLE DB Properties Applied
KEYSET_DRIVEN	DBPROP_OTHERINSERT=FALSE, DBPROP_OTHERUPDATEDELETE=TRUE
STATIC	DBPROP_OTHERINSERT=FALSE, DBPROP_OTHERUPDATEDELETE=FALSE

See your OLE DB programmer reference documentation for details about these properties.

See Also

To assign this option to a group of tables, use the CURSOR_TYPE= option specified in "LIBNAME Options for PC Files on Windows" on page 11.

DBCOMMIT=

Enables you to issue a commit statement automatically after a specified number of rows have been processed

Valid in: DATA and PROC steps (when accessing PC files data using SAS/ACCESS software)

Default value: LIBNAME option setting

Syntax

DBCOMMIT=*number-of-rows*

Syntax Description

number-of-rows
is an integer greater than or equal to 0.

Details

DBCOMMIT= affects update, delete, and insert processing. The number of rows processed includes rows that are not processed successfully. When DBCOMMIT=0, a commit is issued only once (after the procedure or DATA step completes).

If the DBCOMMIT= option is explicitly set, SAS/ACCESS fails any update that has a WHERE clause.

Note: If you specify both DBCOMMIT= and ERRLIMIT=, and these options collide during processing, then the commit is issued first and the rollback is issued second. Because the commit (caused by the DBCOMMIT= option) is issued prior to the rollback (caused by the ERRLIMIT= option), the DBCOMMIT= option is said to override the ERRLIMIT= option in this situation. △

Example

In the following example, a commit is issued after every 10 rows are inserted:

```
data myxls.dept(dbcommit=10);
   set mysas.staff;
run;
```

See Also

To assign this option to a group of tables, use the DBCOMMIT= option specified in "LIBNAME Options for PC Files on Windows" on page 11.

DBCONDITION=

Specifies criteria for subsetting and ordering data

Valid in: DATA and PROC steps (when accessing PC files data using SAS/ACCESS software)

Default value: none

Syntax

DBCONDITION="*SQL-query-clause*"

Syntax Description

SQL-query-clause
is a data source specific SQL query clause, such as WHERE, GROUP BY, HAVING, or ORDER BY.

Details

This option enables you to specify selection criteria in the form of data source specific SQL query clauses, which the SAS/ACCESS engine passes directly to the data source for processing. When selection criteria are passed directly to the data source for processing, performance is often enhanced. The data source checks the criteria for syntax errors when it receives the SQL query.

The DBKEY= option is ignored when you use DBCONDITION=.

DBCREATE_TABLE_OPTS=

Specifies data source specific syntax to be added to the CREATE TABLE statement

Valid in: DATA and PROC steps (when accessing PC files data using SAS/ACCESS software)

Default value: LIBNAME option setting

Syntax

DBCREATE_TABLE_OPTS=*'SQL-clauses'*

Syntax Description

SQL-clauses
are one or more data source specific clauses that can be appended to the end of an SQL CREATE TABLE statement.

Details

This option enables you to add data source specific clauses to the end of the SQL CREATE TABLE statement. The SAS/ACCESS engine passes the SQL CREATE TABLE statement and its clauses to the data source, which executes the statement and creates the table.

See Also

To assign this option to a group of tables, use the DBCREATE_TABLE_OPTS= option specified in "LIBNAME Options for PC Files on Windows" on page 11.

DBFORCE=

Specifies whether to force the truncation of data during insert processing

Valid in: DATA and PROC steps (when accessing PC files data using SAS/ACCESS software)

Default value: NO

Syntax

DBFORCE=YES | NO

Syntax Description

YES
specifies that the rows that contain data values that exceed the length of the column are inserted, and the data values are truncated to fit the column length.

NO
specifies that the rows that contain data values that exceed the column length are not inserted.

Details

This option determines how the SAS/ACCESS engine handles rows that contain data values that exceed the length of the column.

The SAS data set option FORCE= overrides this option when it is used with PROC APPEND or the PROC SQL UPDATE statement. The PROC SQL UPDATE statement does not provide a warning before truncating the data.

DBGEN_NAME=

Specifies whether to rename columns automatically when they contain characters that SAS does not allow

Valid in: DATA and PROC steps (when accessing PC files data using SAS/ACCESS software)

Default value: LIBNAME option setting

Syntax

DBGEN_NAME=DBMS | SAS

Syntax Description

DBMS
specifies that disallowed characters are converted to underscores.

SAS
specifies that columns that contain disallowed characters are converted into valid SAS variable names, using the format _COL*n*, where *n* is the column number (starting with zero). If a name is converted to a name that already exists, a sequence number is appended to the end of the new name.

Details

SAS retains column names when reading data, unless a column name contains characters that SAS does not allow, such as $ or @. SAS allows alphanumeric characters and the underscore (_).

This option is intended primarily for National Language Support, notably the conversion of Kanji to English characters because the English characters converted from Kanji are often those that are not allowed in SAS. If you specify DBGEN_NAME=SAS, a column named DEPT$AMT is renamed to _COL*n* where *n* is the column number. If you specify DBGEN_NAME=DBMS, a column named DEPT$AMT is renamed to DEPT_AMT.

See Also

To assign this option to a group of tables, use the DBGEN_NAME= option specified in “LIBNAME Options for PC Files on Windows” on page 11.

DBKEY=

Improves performance when you are processing a join that involves a large data source table and a small SAS data set (by specifying a column to use as an index)

Valid in: DATA and PROC steps (when accessing PC files data using SAS/ACCESS software)

Default value: none

Syntax

DBKEY=*(<'>column-1<'> <... <'>column-n<'>>)*

Syntax Description

column
is the name of the column that forms the index on the data source table.

Details

When processing a join that involves a large data source table and a relatively small SAS data set, you might be able to use DBKEY= to improve performance.

CAUTION:
Improper use of this option can harm performance. △

DBLABEL=

Specifies whether to use SAS variable labels as data source column names during output processing

Valid in: DATA and PROC steps (when accessing PC files data using SAS/ACCESS software)

Default value: NO

Syntax

DBLABEL=YES | NO

Syntax Description

YES
specifies that SAS variable labels are used as data source column names during output processing.

NO
specifies that SAS variable names are used as data source column names.

Details

This option is valid only for creating data source tables.

Note: Only up to 64 characters of SAS variable labels are written to Microsoft Access or Microsoft Excel files. △

Example

In the following example, the SAS data set New is created with one variable C1. This variable is assigned a label of DeptNum. In the second DATA step, the MyDBLib.MyDept table is created by using DeptNum as the data source column name. Setting DBLABEL=YES enables the label to be used as the column name.

```
data new;
   label c1='deptnum';
   c1=001;
run;

data mydblib.mydept(dblabel=yes);
   set new;
run;

proc print data=mydblib.mydept;
run;
```

DBMAX_TEXT=

Determines the length of a very long data source character data type that is read into SAS or written from SAS when you are using a SAS/ACCESS engine

Valid in: DATA and PROC steps (when accessing PC files data using SAS/ACCESS software)

Default value: LIBNAME option setting

Syntax

DBMAX_TEXT= *integer*

Syntax Description

integer
is a number between 1 and 32,767.

Details

This option applies to reading, appending, and updating rows in an existing table. It does not apply when you are creating a table.

DBMAX_TEXT= is usually used with a very long character data type.

Note: Although you can specify a value less than 256, it is not recommended for reading data from Microsoft Access Database. △

See Also

To assign this option to a group of tables, use the DBMAX_TEXT= option specified in "LIBNAME Options for PC Files on Windows" on page 11.

DBNULL=

Indicates whether NULL is a valid value for the specified columns when a table is created

Valid in: DATA and PROC steps (when accessing PC files data using SAS/ACCESS software)

Default value: YES

Syntax

DBNULL= (*column-name-1*=YES | NO <...*column-name-n*=YES | NO > |_ALL_=YES | NO)

Syntax Description

YES
specifies that the NULL value is valid for the specified columns.

NO
specifies that the NULL value is not valid for the specified columns.

Details

This option is valid only for creating data source tables. If you specify more than one column name, the names must be separated with spaces.

The DBNULL= option processes values from left to right, so if you specify a column name twice, or if you use the _ALL_ value, the last value overrides the first value specified for the column.

Note: This option is only supported by the Access engine and is not supported by the Excel engine. △

Examples

In the following example, by using the DBNULL= option, the EmpId and Jobcode columns in the new MyDBLib.MyDept2 table are prevented from accepting null values.

If the Employees table contains null values in the EmpId or Jobcode columns, the DATA step fails.

```
data mydblib.mydept2(dbnull=(empid=no jobcode=no));
   set mydblib.employees;
run;
```

In the following example, all columns in the new MyDBLib.MyDept3 table except for the Jobcode column are prevented from accepting null values. If the Employees table contains null values in any column other than the Jobcode column, the DATA step fails.

```
data mydblib.mydept3(dbnull=(_ALL_=no jobcode=YES));
   set mydblib.employees;
run;
```

DBNULLKEYS=

Controls the format of the WHERE clause when you use the DBKEY= data set option

Valid in: DATA and PROC steps (when accessing PC files data using SAS/ACCESS software)

Default value: LIBNAME setting

Syntax

DBNULLKEYS= YES | NO

Details

If there might be NULL values in the transaction table or the master table for the columns that you specify in the DBKEY= option, then use DBNULLKEYS=YES. When you specify DBNULLKEYS=YES and specify a column that is not defined as NOT NULL in the DBKEY= data set option, SAS generates a WHERE clause that can find NULL values. For example, if you specify DBKEY=COLUMN and COLUMN is not defined as NOT NULL, SAS generates a WHERE clause with the following syntax:

```
WHERE ((COLUMN = ?) OR ((COLUMN IS NULL) AND (? IS NULL)))
```

This syntax enables SAS to prepare the statement once and use it for any value (NULL or NOT NULL) in the column. Note that this syntax has the potential to be much less efficient than the shorter form of the WHERE clause (presented below). When you specify DBNULLKEYS=NO or specify a column that is defined as NOT NULL in the DBKEY= option, SAS generates a simple WHERE clause.

If you know that there are no NULL values in the transaction table or the master table for the columns that you specify in the DBKEY= option, you can use DBNULLKEYS=NO. If you specify DBNULLKEYS=NO and specify DBKEY=COLUMN, SAS generates a shorter form of the WHERE clause (regardless of whether or not the column specified in DBKEY= is defined as NOT NULL):

```
WHERE (COLUMN = ?)
```

See Also

To assign this option to a group of tables, use the DBNULLKEYS= option specified in "LIBNAME Options for PC Files on Windows" on page 11.

DBSASLABEL=

specifies whether SAS/ACCESS saves the data source's column names as SAS label names

Valid in: DATA and PROC steps (when accessing PC files data using SAS/ACCESS software)

Default value: COMPAT

Syntax

DBSASLABEL= COMPAT | NONE

Syntax Description

COMPAT
specifies that SAS/ACCESS saves the data source's column names as SAS label names. This is compatible to the previous SAS releases.

NONE
specifies that SAS/ACCESS does not save the data source's column names as SAS label names. SAS label names are left as NULLs.

Details

This option is valid only while you are reading data into SAS from the data source.

DBSASTYPE=

Specifies data type(s) to override the default SAS data type(s) during input processing of data

Valid in: DATA and PROC steps (when accessing PC files data using SAS/ACCESS software)

Default value: none

Syntax

DBSASTYPE=*(column-name-1=<'>SAS-data-type<'>*
<...column-name-n=<'>SAS-data-type<'>>)

Syntax Description

column-name
specifies a data source column name.

SAS-data-type
specifies a SAS data type. SAS data types include the following: CHAR(*n*), NUMERIC, DATETIME, DATE, TIME.

Details

By default, SAS/ACCESS converts each data source data type to a SAS data type during input processing. When you need a different data type, you can use this option to override the default and assign a SAS data type to each specified data source column. Some conversions might not be supported. If a conversion is not supported, SAS prints an error to the log.

DBTYPE=

Specifies a data type to use instead of the default data source data type when SAS creates a data source table

Valid in: DATA and PROC steps (when accessing PC files data using SAS/ACCESS software)

Default value: none

Syntax

DBTYPE=*(column-name-1=<'>data-source-type<'>*
<...column-name-n=<'>data-source-type<'>>)

Syntax Description

column-name
specifies a data source column name.

data-source-type
specifies a data source data type. See the documentation for your SAS/ACCESS interface for the default data types for your data source.

Details

By default, SAS/ACCESS converts each SAS data type to a predetermined data source data type when outputting data to your data source. When you need a different data type, use DBTYPE= to override the default data type chosen by the SAS/ACCESS engine.

Examples

In the following example, DBTYPE= specifies the data types that are used when you create columns in the table.

```
data mydblib.newdept(dbtype=(deptno='double' city='char(25)'));
   set mydblib.dept;
run;
```

See Also

"LIBNAME Statement Data Conversions for MDB Files" on page 169

"LIBNAME Statement Data Conversions for XLS Files" on page 152

ERRLIMIT=

Specifies the number of errors that are allowed before SAS stops processing and issues a rollback

Valid in: DATA and PROC steps (when accessing PC files data using SAS/ACCESS software)

Default value: 1

Syntax

ERRLIMIT=*integer*

Syntax Description

integer
is a positive integer that represents the number of errors after which SAS stops processing and issues a rollback.

Details

SAS calls the data source to issue a rollback after a specified number of errors occurs during the processing of inserts, deletes, updates, and appends. If ERRLIMIT= is set to 0, SAS processes all rows, regardless of the number of errors that occur. The SAS log displays the total number of rows processed and the number of failed rows, if applicable.

The DBCOMMIT= option overrides the ERRLIMIT= option. If you specify a value for DBCOMMIT= other than zero, then rollbacks affected by the ERRLIMIT= option might not include records that are processed unsuccessfully because they were already committed by DBCOMMIT=.

Note: This option cannot be used from a SAS client session in a SAS/SHARE environment. △

Example

In the following example, SAS stops processing and issues a rollback to the data source at the occurrence of the tenth error. The MyDBLib libref was assigned in a prior LIBNAME statement.

```
data mydblib.employee3 (errlimit=10);
   set mydblib.employees;
   where salary > 40000;
run;
```

INSERT_SQL=

Determines the method that is used to insert rows into a data source

Valid in: DATA and PROC steps (when accessing PC files data using SAS/ACCESS software)

Default value: LIBNAME option setting

Syntax

INSERT_SQL=YES | NO

Syntax Description

YES
specifies that the SAS/ACCESS engine uses the data source's SQL insert method to insert new rows into a table.

NO
specifies that the SAS/ACCESS engine uses an alternate (data source specific) method to add new rows to a table.

See Also

To assign this option to a group of tables, use the INSERT_SQL= option specified in "LIBNAME Options for PC Files on Windows" on page 11.

INSERTBUFF=

Specifies the number of rows in a single insert

Valid in: DATA and PROC steps (when accessing PC files data using SAS/ACCESS software)

Default value: LIBNAME option setting

Syntax

INSERTBUFF=*number-of-rows*

Syntax Description

number-of-rows
specifies the number of rows to insert. The value must be a positive integer.

Details

SAS allows the maximum number of rows that is allowed by the data source. The optimal value for this option varies with factors such as network type and available memory. You might need to experiment with different values to determine the best value for your site.

When you assign a value that is greater than INSERTBUFF=1, the SAS application notes that indicate the success or failure of the insert operation might be incorrect because these notes only represent information for a single insert, even when multiple inserts are performed.

If the DBCOMMIT= option is specified with a value that is less than the value of INSERTBUFF=, then DBCOMMIT= overrides INSERTBUFF=.

Note: When you are inserting with the VIEWTABLE window or the FSEDIT or FSVIEW procedure, use INSERTBUFF=1 to prevent the data source interface from trying to insert multiple rows. These features do not support inserting more than one row at a time. △

See Also

To assign this option to a group of tables, use the INSERTBUFF= option specified in "LIBNAME Options for PC Files on Windows" on page 11.

NULLCHAR=

Indicates how SAS character missing values are handled during insert, update, and DBKEY= processing

Valid in: DATA and PROC steps (when accessing PC files data using SAS/ACCESS software)

Default value: SAS

Syntax

NULLCHAR= SAS | YES | NO

Syntax Description

SAS
indicates that character missing values in SAS data sets are treated as NULL values if the data source allows them. Otherwise, character missing values are treated as the NULLCHARVAL= value.

YES
indicates that character missing values in SAS data sets are treated as NULL values if the data source allows them. Otherwise, an error is returned.

NO
indicates that character missing values in SAS data sets are treated as the NULLCHARVAL= value (regardless of whether the data source allows NULLs for the column).

Details

This option affects insert and update processing and also applies when you use the DBKEY= option.

This option works in conjunction with the NULLCHARVAL= data set option, which determines what is inserted when NULL values are not allowed.

All SAS numeric missing values (represented in SAS as '.') are treated by the data source as NULLs.

NULLCHARVAL=

Defines the character string that replaces SAS character missing values during insert, update, and DBKEY= processing

Valid in: DATA and PROC steps (when accessing PC files data using SAS/ACCESS software)

Default value: a blank character

Syntax

NULLCHARVAL='*character-string*'

Details

This option affects insert and update processing and also applies when you use the DBKEY= option.

This option works with the NULLCHAR= option, which determines whether or not a SAS character NULL value is treated as a NULL value.

If NULLCHARVAL= is longer than the maximum column width, one of the following occurs:

- The string is truncated if DBFORCE=YES.
- The operation fails if DBFORCE=NO.

READBUFF=

Specifies the number of rows of data to read into the buffer

Valid in: DATA and PROC steps (when accessing PC files data using SAS/ACCESS software)

Default value: LIBNAME option setting

Syntax

READBUFF=*number-of-rows*

Syntax Description

number-of-rows
is the maximum value that is allowed by the data source.

Details

This option improves performance by specifying a number of rows that can be held in memory for input into SAS. Buffering data reads can decrease network activities and increase performance. However, because SAS stores the rows in memory, higher values for READBUFF= use more memory. In addition, if too many rows are selected at once, then the rows that are returned to the SAS application might be out of date.

When READBUFF=1, only one row is retrieved at a time. The higher the value for READBUFF=, the more rows the SAS/ACCESS engine retrieves in one fetch operation.

ROWSET_SIZE is an alias for this option.

See Also

To assign this option to a group of tables, use the READBUFF= option specified in "LIBNAME Options for PC Files on Windows" on page 11.

SASDATEFMT=

Changes the SAS date format of a data source column

Valid in: DATA and PROC steps (when accessing PC files data using SAS/ACCESS software)

Default value: none

Syntax

SASDATEFMT=*(data-source-date-column-1='SAS-date-format'*
<... data-source-date-column-n='SAS-date-format'>)

Syntax Description

data-source-date-column
specifies the name of a date column in a data source table.

SAS-date-format
specifies a SAS date format that has an equivalent (like-named) informat. For example, DATETIME21.2 is both a SAS format and a SAS informat, so it is a valid value for the *SAS-date-format* argument.

Details

If the date format of a SAS column does not match the date format of the corresponding data source column, you must convert the SAS date values to the appropriate data source date values. The SASDATEFMT= option enables you to convert date values from the default SAS date format to another SAS date format that you specify.

Use the SASDATEFMT= option to prevent date type mismatches in the following circumstances:

- during input operations to convert data source date values to the correct SAS DATE, TIME, or DATETIME values
- during output operations to convert SAS DATE, TIME, or DATETIME values to the correct data source date values.

If the SAS date format and the data source date format match, this option is not needed.

The default SAS date format is data source specific and is determined by the data type of the data source column. See the documentation for your SAS/ACCESS interface.

Note: For non-English date types, SAS automatically converts the data to the SAS type of NUMBER. The SASDATEFMT= option does not currently handle these date types, but you can use a PROC SQL view to convert the data source data to a SAS date format as you retrieve the data, or use a format statement in other contexts. △

CHAPTER

3 The Pass-Through Facility for PC Files on Windows

Overview of the Pass-Through Facility for PC Files

The SQL procedure implements the Structured Query Language (SQL) for SAS. See the SQL procedure topic in *Base SAS Procedures Guide* for information about PROC SQL. You can send data source specific SQL statements directly to a data source using an extension to the SQL procedure called the Pass-Through Facility.

This facility uses SAS/ACCESS to connect to a data source and to send statements directly to the data source for execution. This facility is an alternative to the SAS/ACCESS LIBNAME statement. It enables you to use the SQL syntax of your data source, and it supports any non-ANSI standard SQL that is supported by your data source.

The Pass-Through Facility enables you to do the following:

- establish and terminate connections with a data source using the facility's CONNECT and DISCONNECT statements
- send dynamic, non-query, data source specific SQL statements to a data source using the facility's EXECUTE statement
- retrieve data directly from a data source using the facility's CONNECTION TO component in the FROM clause of a PROC SQL SELECT statement.

You can use Pass-Through Facility statements in a PROC SQL query or you can store them in a PROC SQL view. When you create a PROC SQL view, any arguments that you specify in the CONNECT statement are stored with the view. Therefore, when the view is used in a SAS program, SAS can establish the appropriate connection to the data source.

Syntax for the Pass-Through Facility for PC Files

This section presents the syntax for the Pass-Through Facility statements and the CONNECTION TO component, which can be used in conjunction with the PROC SQL SELECT statement to query data from a data source.

PROC SQL <*options-list*>;

CONNECT TO *data-source-name* <AS *alias*> <(<*connect-statement-arguments*> <*database-connection-arguments*>)>;

DISCONNECT FROM *data-source-name* | *alias*;

EXECUTE (*data-source-specific-SQL-statement*) BY *data-source-name* | *alias*;

SELECT *column-list* FROM **CONNECTION TO** *data-source-name* | *alias* *(data-source-query)*

Return Codes

As you use the PROC SQL statements that are available in the Pass-Through Facility, any error conditions are written to the SAS log. The Pass-Through Facility generates return codes and messages that are available to you through the following two SAS macro variables:

SQLXRC
contains the data source return code that identifies the data source error.

SQLXMSG
contains descriptive information about the data source error that is generated by the data source.

The contents of the SQLXRC and SQLXMSG macro variables are printed in the SAS log using the %PUT macro. They are reset after each Pass-Through Facility statement has been executed.

Example

To connect to an Excel file and query the INVOICE table (range) within the Excel workbook:

```
PROC SQL DQUOTE=ANSI;
CONNECT TO EXCEL (PATH="c:\sasdemo\sasdemo.xls");
SELECT * FROM CONNECTION TO EXCEL
   (SELECT * FROM INVOICE);
DISCONNECT FROM EXCEL;
QUIT;
```

CONNECT Statement

Establishes a connection with the data source

Valid in: PROC SQL steps

Syntax

CONNECT TO *data-source-name* <AS *alias*> <(<*connect-statement-arguments*> <*database-connection-arguments*>)>;

Arguments

data-source-name
identifies the data source to which you want to connect, such as ACCESS for Microsoft Access or EXCEL for Microsoft Excel. You may also specify an optional alias in the CONNECT statement.

alias
specifies an optional alias for the connection that has 1 to 32 characters. If you specify an alias, the keyword AS must appear before the alias. If an alias is not specified, the data source name is used as the name of the Pass-Through connection.

connect-statement-arguments
specifies arguments that indicate whether you can make multiple connections, shared or unique connections, and so on to the database. These arguments are optional.

database-connection-arguments
specifies the data source specific arguments that are needed by PROC SQL to connect to the data source. These arguments are not required and the default behavior opens a dialog box.

Database Connection Arguments

The arguments that are listed below are available with the Pass-Through Facility for PC files. These arguments extend some of the LIBNAME statement connection management features to the Pass-Through Facility.

DBPASSWORD="*database-file-password*"
enables you to access your file if you have database-level security set in your MDB file. A database password is case-sensitive and is defined in addition to user-level security.

Note: This connection option is only for Microsoft Access. △

DBSYSFILE="*workgroup-information-file*"
contains information about the users in a workgroup based on information that you define for you Microsoft Access database. Any user and group accounts or passwords you create are saved in the new workgroup information file.

Note: This connection option is only for Microsoft Access. △

HEADER=YES | NO
determines whether the first row of data in an Excel range (or spreadsheet) is column names when you are reading data from the Excel file.

YES — specifies to use the first row of data in an Excel range (or spreadsheet) as column names when you are reading data from the Excel file.

NO — specifies to not use the first row of data as column names in an Excel range (or spreadsheet) when you are reading data from the Excel file.

Note: This connection option is only for Microsoft Excel. △

INIT= "*connection-string*"
specifies an initialization string (that is, a connection string) when connecting to a data source.

MIXED=YES | NO
specifies whether to convert numeric data values into character data values for a column with mixed data types. This option is valid only when you are importing data from Excel.
Aliases: MIXED_DATA=, MIXED_DATATYPE=.

YES — specifies that the engine assigns a SAS character type for the column and convert all numeric data values to character data.

NO — specifies that numeric data is imported as missing values in a character column (default).

Note: The use of MIXED= option causes the Excel workbook to be locked in READONLY mode. No update is possible until the libref is deassigned. This option is not valid for accessing data in Microsoft Access database. △

PASSWORD="*user-password*"
specifies a password for the user account. A password can be 1 to 14 characters long and can include any characters except ASCII character 0 (null). Passwords are case-sensitive.

Note: This connection option is only for Microsoft Access. △

PATH="*path-for-file*"
specifies the data source file location for the Microsoft Access database file or Microsoft Excel workbook file.

PROMPT=YES | NO |REQUIRED | NO PROMPT | PROMPT | UDL
determines whether you will be prompted for connection information to supply to the data source information.

YES — enables you to be prompted with the Data Link Properties window.

NO — does not enable you to be prompted with a window, and requires you to specify the *physical-filename*.

REQUIRED — enables you to connect without prompting for more information only if a valid physical filename is specified for a successful connection. Otherwise, you are prompted for the connection options with a window that enables you to change the data source file and other properties.

NOPROMPT — disables the prompt of the Data Link Properties window.

PROMPT
: enables you to be prompted for connection information to supply the data source information.

UDL
: enables you to browse and select an existing data link file (.udl).

Note: This statement also applies to the INIT= and UDL= options. △

UDL="*path-for-udl-file*"
: specifies the path and filename for a UDL (a Microsoft data link file). For example, you could specify

```
UDL_FILE="C:\WinNT\profiles\me\desktop\MyDBLink.udl'';
%put %superq(SYSDBMSG);
```

This option does not support SAS filerefs. The macro variable SYSDBMSG is set on successful completion. For more information, see Microsoft's documentation on the data link API.

USER="*user-ID*"
: specifies a default user account name. The default value is Admin. User names can be 1 to 20 characters long and can include alphabetic characters, accented characters, numbers, and spaces. If you have user-level security set in your MDB file, you need to use this option and the PASSWORD= option to be able to access your file.

Note: This connection option is only for Microsoft Access. △

VERSION=2002 | 2000 | 97 | 95 | 5
: sets the version of Microsoft Excel. The default value is 97.

Note: This connection option is only for Microsoft Excel. △

2002
: sets the version of Microsoft Excel to 2002.

2000
: sets the version of Microsoft Excel to 2000.

97
: sets the version of Microsoft Excel to 97.

95
: sets the version of Microsoft Excel to 95.

5
: sets the version of Microsoft Excel to 5.

CONNECT Statement Arguments

The arguments that are listed below are available with the Pass-Through Facility CONNECT statement for PC files. These arguments extend some of the LIBNAME statement connection management features to the Pass-Through Facility.

AUTOCOMMIT=YES | NO
: determines whether the ACCESS engine commits (saves) updates as soon as the user submits them.

YES
: specifies that updates are committed (that is, saved) to table as soon as they are submitted, and no rollback is possible.

NO
: specifies that the SAS/ACCESS engine automatically performs the commit when it reaches the end of the file.

Default: YES

Note: The default value for this option is different from the LIBNAME option. △

COMMAND_TIMEOUT=*number-of-seconds*
specifies the number of seconds that pass before a data source command times out.

Default: 0 (no timeout)

Alias: TIMEOUT=

CONNECTION= SHARED | GLOBAL
specifies whether multiple CONNECT statements for a data source can use the same connection. The CONNECTION= option enables you to control the number of connections, and therefore transactions, that your SAS/ACCESS engine executes and supports for each CONNECT statement.

SHARED
specifies that the CONNECT statement makes one connection to the DBMS. Only Pass-Through statements that use this alias share the connection.

GLOBAL
specifies that multiple CONNECT statements can share the same connection to the DBMS if they use identical values for CONNECTION=, CONNECTION_GROUP=, and any database connection arguments.

Default: SHARED

CONNECTION_GROUP= *connection-group*
causes operations against multiple librefs to share a connection to the data source. Also causes operations against multiple Pass-Through Facility CONNECT statements to share a connection to the data source.

CURSOR_TYPE=KEYSET_DRIVEN | STATIC
specifies the cursor type for read-only and updatable cursors.

KEYSET_DRIVEN
specifies that the cursor determines which rows belong to the result set when the cursor is opened. However, changes that are made to these rows are reflected as you move the cursor. The OLE DB property DBPROP_OTHERUPDATEDELETE is set as TRUE for keyset cursors and FALSE for static cursors.

STATIC
specifies that the complete result set is built when the cursor is opened, but no changes made to the result set will be reflected in the cursor. Static cursors are read-only.

Default: none

Alias: CURSOR=

DBGEN_NAME=DBMS | SAS
specifies that the data source columns are renamed, and specifies the format that the new names will follow.

DBMS
specifies that the data source columns are renamed to valid SAS variable names. Disallowed characters are converted to underscores. If a column is converted to a name that already exists, then a sequence number is appended to the end of the new name.

SAS
specifies that data source columns are renamed to the format _COL*n*, where *n* is the column number (starting with zero).

Default: DBMS

DBMAX_TEXT=*n*

specifies an integer between 1 and 32,767 that indicates the maximum length for a character string. Longer character strings are truncated. This option only applies when you are reading, appending, and updating Microsoft Access or Excel character data from SAS.

Note: Although you can specify a value less than 256, it is not recommended. △

Default: 1,024

DEFER=NO | YES
enables you to specify when the connection to the data source occurs.

NO
specifies that the connection to the data source occurs when the libref is assigned by a LIBNAME statement.

YES
specifies that the connection to the data source occurs when a table in the data source is opened.

Default: NO

READBUFF=*number-of-rows*
specifies the number of rows to use when you are reading data from a data source. Setting a higher value for this option reduces I/O and increases performance, but also increases memory usage. Additionally, if too many rows are read at once, values returned to SAS might be out of date.

Default: 1

Alias: ROWSET=
ROWSET_SIZE=

STRINGDATES=YES | NO
specifies whether datetime values are read from the data source as character strings or as numeric date values. STRINGDATES= is not available as a data set option.

YES
sepcifies that SAS/ACCESS reads datetime values as character strings.

NO
specifies that SAS/ACCESS reads datetimes values as numeric date values.

Default: NO

Alias: STRDATES

USE_DATETYPE=YES | NO
specifies whether to use DATE. format for date/time columns/fields in the data source table while importing data from Microsoft Access database or Excel workbook.

YES
specifies that SAS DATE format is assigned for datetime columns in the data source table.

NO
specifies SAS DATETIME format is assigned for datetime columns in the data source table.

Default: NO

Alias: STRDATES

Details

The CONNECT statement establishes a connection with the data source. You establish a connection to send data source specific SQL statements to the data source or to

retrieve data source data. The connection remains in effect until you issue a DISCONNECT statement or terminate the SQL procedure.

To connect to a data source using the Pass-Through Facility, complete the following steps:

1. Initiate a PROC SQL step.
2. Use the Pass-Through Facility's CONNECT statement, identify the data source (such as Microsoft Access or Excel), and (optionally) assign an alias.
3. Specify any arguments needed to connect to the database.
4. Specify any attributes for the connection.

The CONNECT statement is optional for some data sources. However, if it is not specified, the default values for all of the database connection arguments are used.

Any return code or message that is generated by the data source is available in the macro variables SQLXRC and SQLXMSG after the statement executes. See "Return Codes" on page 36 for more information about these macro variables.

Example

The following example uses the CONNECT statement with PATH= option to connect to the Microsoft Access database file, **`c:/demo.mdb`**:

```
proc sql;
connect to access as db (path="c:\demo.mdb");
```

DISCONNECT Statement

Terminates the connection to the data source

Valid in: PROC SQL steps

Syntax

DISCONNECT FROM *data-source-name* | *alias*

Arguments

data-source-name
specifies the data source from which you want to disconnect. You can use an alias in the DISCONNECT statement. The DISCONNECT statement's data source name or alias must match the name or alias that you specified in the CONNECT statement.

alias
specifies an alias that was defined in the CONNECT statement.

Details

The DISCONNECT statement ends the connection with the data source. If the DISCONNECT statement is omitted, an implicit DISCONNECT is performed when

PROC SQL terminates. The SQL procedure continues to execute until you submit a QUIT statement, another SAS procedure, or a DATA step.

Any return code or message that is generated by the data source is available in the macro variables SQLXRC and SQLXMSG after the statement executes. See "Return Codes" on page 36 for more information about these macro variables.

Example

The following example, after the connection and SQL processing uses the DISCONNECT statement to disconnect the connection from the database, and uses the QUIT statement to quit the SQL procedure:

```
disconnect from db;
quit;
```

EXECUTE Statement

Sends data source specific, non-query SQL statements to the data source

Valid in: PROC SQL steps

Syntax

EXECUTE (*data-source-specific-SQL-statement*) BY *data-source-name* | *alias*;

Arguments

(*data-source-specific-SQL-statement*)
a dynamic nonquery, data source specific SQL statement. This argument is required and must be enclosed in parentheses. However, the SQL statement cannot contain a semicolon because a semicolon represents the end of a statement in SAS. The SQL statement can be case-sensitive, depending on your data source, and it is passed to the data source exactly as you type it.

Any return code or message that is generated by the data source is available in the macro variables SQLXRC and SQLXMSG after the statement executes. See "Return Codes" on page 36 for more information about these macro variables.

data-source-name
identifies the data source to which you direct the data source specific SQL statement. The keyword BY must appear before the *data-source-name* argument. You must specify either the data source name or an alias.

alias
specifies an alias that was defined in the CONNECT statement. (You cannot use an alias if the CONNECT statement was omitted.)

Details

The EXECUTE statement sends dynamic nonquery, data source specific SQL statements to the data source and processes those statements.

The EXECUTE statement cannot be stored as part of a Pass-Through Facility query in a PROC SQL view.

Useful Statements to Include in EXECUTE Statements

You can pass the following statements to the data source by using the Pass-Through Facility's EXECUTE statement.

CREATE
creates a data source table, view, index, or other data source object, depending on how the statement is specified.

DELETE
deletes rows from a data source table.

DROP
deletes a data source table, view, or other data source object, depending on how the statement is specified.

INSERT
adds rows to a data source table.

UPDATE
modifies the data in the specified columns of a row in a data source table.

For more information about these and other SQL statements, see the SQL documentation for your data source.

Example

The following example, after the connection, uses the EXECUTE statement to drop a table, create a table, and insert a row of data:

```
execute(drop table 'My Invoice') by db;
execute(create table 'My Invoice'(
'Invoice Number' LONG not null,
'Billed To' VARCHAR(20),
'Amount' CURRENCY,
'BILLED ON' DATETIME)) by db;
execute(insert into 'My Invoice'
values( 12345, 'John Doe', 123.45, #11/22/2003#)) by db;
```

CONNECTION TO Component

Retrieves and uses data source data in a PROC SQL query or view

Valid in: PROC SQL step SELECT statements

Syntax

CONNECTION TO *data-source-name*<*AS alias*><*(database-connection-options)*>

Arguments

data-source-name

identifies the data source (Microsoft Access or Excel) to which you direct the data source specific SQL statement.

alias
specifies an alias, if one was defined in the CONNECT statement.

Details

The CONNECTION TO component specifies the data source connection that you want to use or that you want to create (if you have omitted the CONNECT statement). CONNECTION TO then enables you to retrieve data source data directly through a PROC SQL query.

You use the CONNECTION TO component in the FROM clause of a PROC SQL SELECT statement:

SELECT *column-list*
FROM CONNECTION TO *data source-name (data source-query);*

CONNECTION TO can be used in any FROM clause, including those in nested queries (that is, in subqueries).

You can store a Pass-Through Facility query in a PROC SQL view and then use that view in SAS programs. When you create a PROC SQL view, any options that you specify in the corresponding CONNECT statement are stored too. Thus, when the PROC SQL view is used in a SAS program, SAS can establish the appropriate connection to the data source.

Because external data sources and SAS have different naming conventions, some data source column names might be changed when you retrieve data source data through the CONNECTION TO component.

Example

The following example, after the connection, uses the CONNECTION TO component to query a table or a subtable:

```
select * from connection to db(select * from 'my invoice');
select * from connection to db
(select 'Invoice Number', Amount from 'my invoice');
```

Special Jet Queries

SAS/ACCESS software for PC Files supports a number of special queries that return information such as available tables, columns, and procedures.

The general format of the special queries is the following:

JET::*schema-rowset*<"*parameter-1*",.....,"*parameter-n*">

where

JET::
is required to distinguish special queries from regular queries.

schema-rowset
is the specific schema rowset that is being called. The valid schema rowsets are listed below.

"*parameter-n*"
is a quoted string. Parameters are separated from one another by commas. All parameters are optional, but the parentheses must be included. If you specify some, but not all, parameters within an argument, use commas to indicate the omitted parameters.

The following special queries are supported:

JET::CHECK_CONSTRAINTS
returns the check constraints that are defined in the database file.

JET::COLUMNS <*"table-name", "column-name"*>
returns the columns of the tables that are defined in the database file.

JET::CONSTRAINT_COLUMN_USAGE <*"table-name", "column-name"*>
returns the columns that are used by referential constraints, unique constraints, check constraints, and assertions that are defined in the database file.

JET::FOREIGN_KEYS <*"primary-key-table-name", "foreign-key-table-name"*>
returns the foreign key columns that are defined in the database file.

JET::INDEXES <*"index-name", "table-name"*>
returns the indexes that are defined in the database file.

JET::KEY_COLUMN_USAGE <*"constraint-name", "table-name", "column-name"*>
returns the key columns that are defined in the database file.

JET::PRIMARY_KEYS <*"table-name"*"'>
returns the primary key columns that are defined in the database file.

JET::PROCEDURES <*"procedure-name"*>
returns the procedures that are defined in the database file.

JET::PROVIDER_TYPES
returns information on the base data types that are supported by the Jet data provider.

JET::REFERENTIAL_CONSTRAINTS <*"constraint-name"*>
returns the referential constraints that are defined in the database file.

JET::STATISTICS <*"table-name"*>
returns the statistics that are defined in the database file.

JET::TABLE_CONSTRAINTS <*"constraint-name", "table-name", "constraint-type"*>
returns the table constraints that are defined in the database file.

JET::TABLES <*"table-name", "table-type"*>
returns the tables that are defined in the database file.

JET::VIEWS <*"table-name"*>
returns the viewed tables that are defined in the database file.

Examples

The following example retrieves a rowset that displays all of the tables in the NorthWind database:

```
proc sql;
  connect to access (path="c:\NorthWind.mdb");
  select * from connection to access(jet::tables);
quit;
```

In the following example, you retrieve the information of all the data types supported by the Jet provider for Microsoft Access:

```
proc sql;
  connect to access (path="c:\NorthWind.mdb");
  select * from connection to access(jet::provider_types);
quit;
```

Special Jet Commands

Microsoft Access and Microsoft Excel engines support several special commands in the Pass-Through Facility.

The general format of the special command is the following:

JET::*command*

where

JET::
is required to distinguish special queries from regular queries.

The following special commands are supported:

JET::COMMIT
commits the transaction.

JET::ROLLBACK
causes a rollback in the transaction.

JET::AUTOCOMMIT
sets the COMMIT mode to AUTO and commits the transaction immediately.

JET::NOAUTOCOMMIT
sets the COMMIT mode to MANUAL. When the COMMIT mode is set to MANUAL, you must issue a COMMIT or ROLLBACK command to commit or rollback the transaction.

Examples

The following example specifies the AUTOCOMMIT=NO connection option.

Note: Although the examples below state that they are Microsoft Access, the syntax is the same for both Microsoft Access and Microsoft Excel. △

```
proc sql;
connect to access( path="d:\dbms\access\test.mdb" autocommit= no );

execute(create table x (c1 int) ) by access;
execute(insert into x values( 1 ) ) by access;
   /* To commit the table create and insert ; */
execute(jet::commit) by access;

execute(insert into x values( 2 ) ) by access;
   /* To rollback the previous insert ; */
execute(jet::rollback) by access;

execute(jet::autocommit) by access;
   /* the insert is automatically committed, you cannot rollback the insert. */
execute(insert into x values( 3 ) ) by access;


   /* you should have a table created with 2 rows. */
disconnect from access; quit;
```

CHAPTER

4

The Import/Export Wizard and Procedures

Import/Export Overview for PC Files

This section introduces the Import/Export wizard and procedures for PC Files. For comprehensive documentation about these features, see *Base SAS Procedures Guide*.

The "Import/Export Wizard" on page 50 and the "IMPORT and EXPORT Procedures" on page 54 enable you to read and write data between SAS data sets and external PC files. The wizard and procedures have similar capabilities; the wizard is a point-and-click interface and the procedures are code-based. The wizard does not provide the ability to specify data set options (for example, DROP, KEEP, and WHERE.)

These wizard and procedures are available under the following operating environments:

Table 4.1 Availability of the Import/Export Wizard and Procedures

Operating Environment	File Formats
Windows 2000, XP, NT	dBASE DBF (III, III PLUS, IV, or 5.0)
	Microsoft Access (97, 2000, or 2002)
	Microsoft Excel (4, 5, 95, 97, 2000, or 2002)
	Lotus 1-2-3 (1, 3, or 4)
	delimited
	JMP
OpenVMS Alpha	delimited

Note: The Import/Export wizard and procedures are part of Base SAS software. If you do not have a license to SAS/ACCESS software for PC Files, however, you can only access CSV, TXT, and delimited files. △

Import/Export Wizard

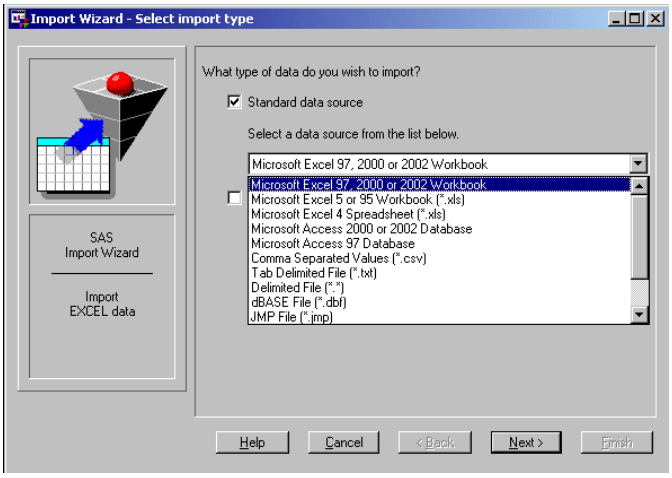
The Import/Export wizard guides you through the importing or exporting process. See Table 4.1 on page 49 for a list of file formats supported under your operating environment.

To invoke the Import/Export wizard, from the SAS windowing environment, select **`File`** and then either **`Import Data`** or **`Export Data`**. Detailed information about using the wizard is available from the Help button.

The Import wizard enables you to read data from an external data source and write it to a SAS data set. External data sources can include Microsoft Access files, Microsoft Excel files, DIF files, DBF files, JMP files, or delimited files, which are files containing columns of data values that are separated by a delimiter such as a blank or a comma. The following displays show the steps of the Import wizard under Windows NT.

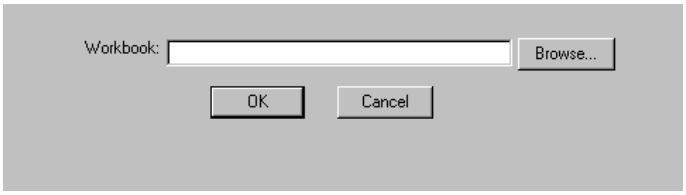
1 Select the type of files you are importing.

Display 4.1 Import Wizard: Select Import Type



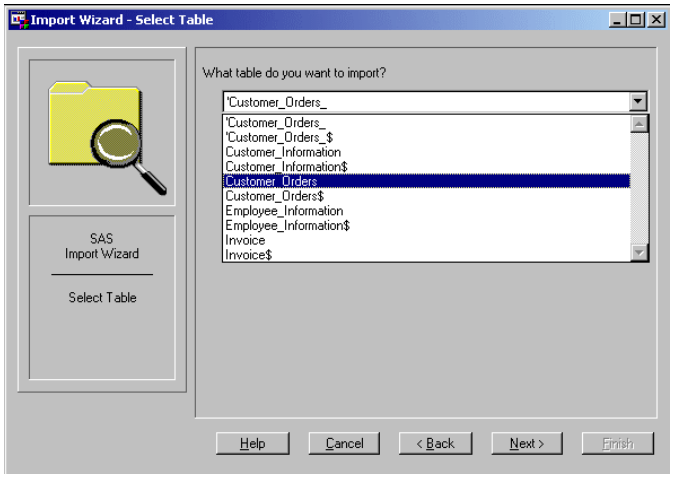
2 Locate the Input File (Excel workbook in this case).

Display 4.2 Import Wizard: Import an Excel File



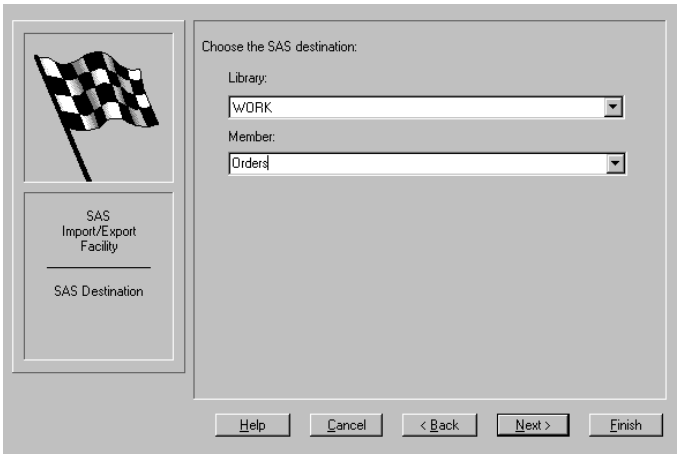
3 Select the table range or worksheet from which to import data.

Display 4.3 Import Wizard: Select Table



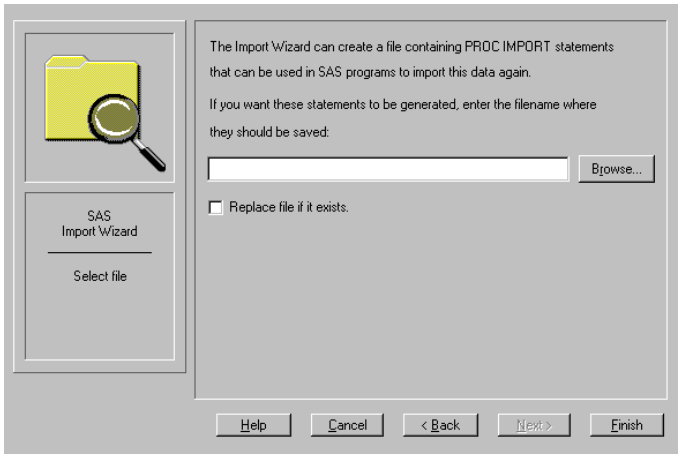
4 Select a location to store the imported file.

Display 4.4 Import Wizard: SAS destination



5 Save the generated PROC IMPORT code. (Optional)

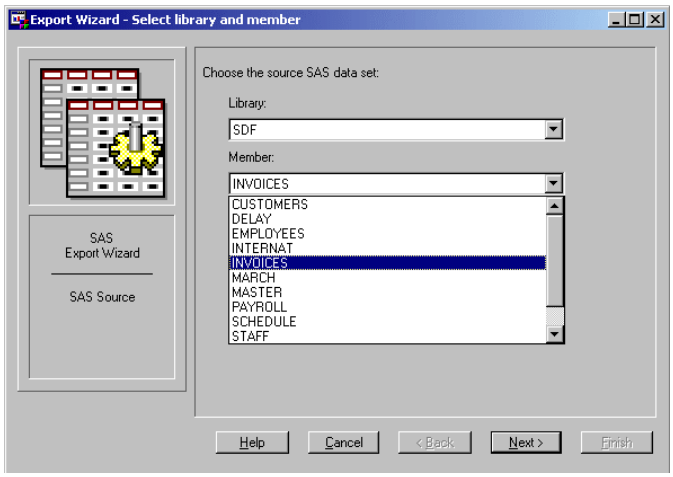
Display 4.5 Import Wizard: Save Generated Code



The Export wizard reads data from a SAS data set and writes it to an external file source. The following display shows an example of the Export wizard under Windows.

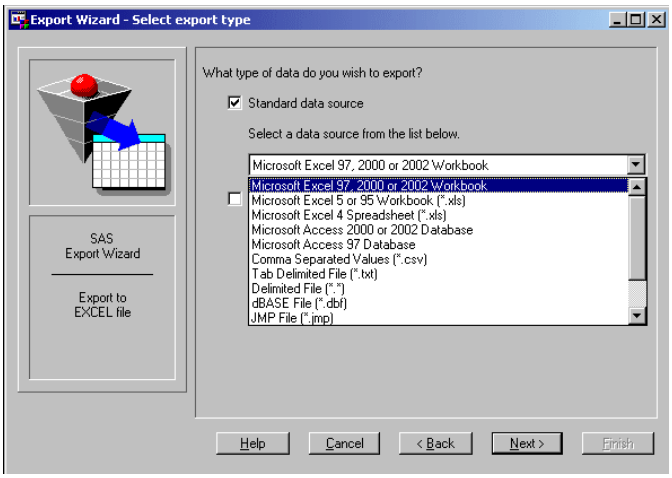
1 Select the SAS data set from which you want to export data.

Display 4.6 Export Wizard: Select Library and Member



2 Select the type of data source to which you want to export files.

Display 4.7 Export Wizard: Select Export Type

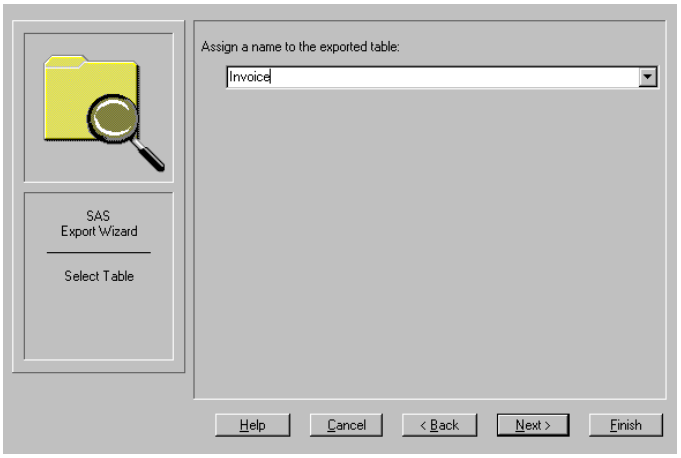


3 Assign the output file.

Display 4.8 Export Wizard: Assign Output File (Excel Workbook)

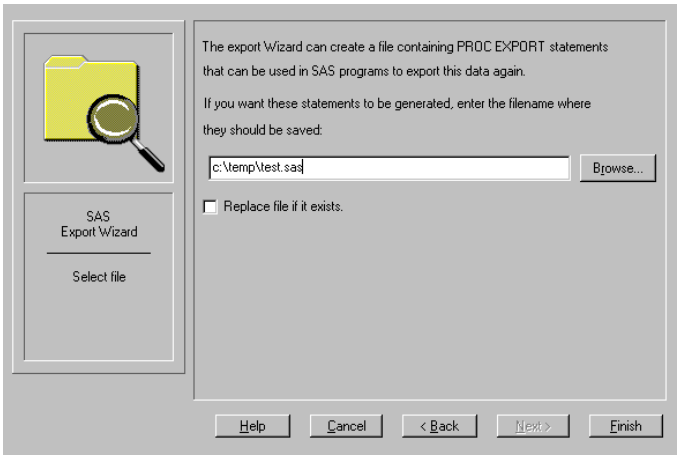
4 Assign the table name (in Excel, sheet name).

Display 4.9 Export Wizard: Name Table



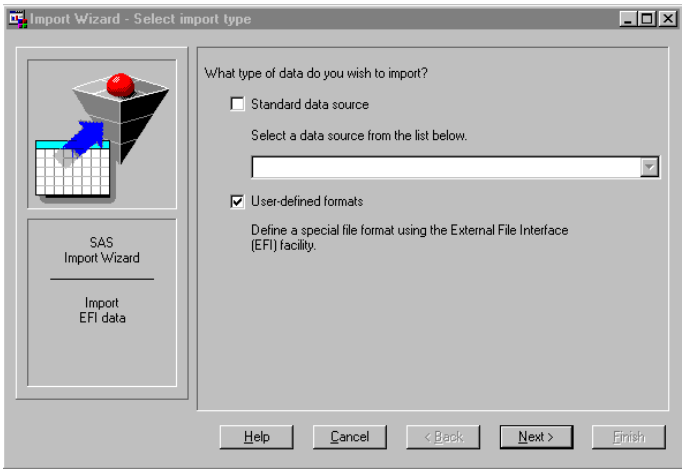
5 Save the generated PROC EXPORT code. (Optional)

Display 4.10 Export Wizard: Save Generated Code



From the primary window of the Import/Export wizard, you can also access the *External File Interface* (EFI). EFI is a point-and-click interface that enables you to read and write data in a file type that is not known to the Import/Export wizard. For example, you could use EFI to transfer data from a SAS data set to a file format that is proprietary for your company. Detailed information about using EFI is available from the Help button. The following display shows you how to access EFI from the Import wizard.

Display 4.11 Accessing the External File Interface



IMPORT and EXPORT Procedures

Like the “Import/Export Wizard” on page 50, the IMPORT and EXPORT procedures transfer data between SAS and external data sources. These external data sources can include tables, PC files, spreadsheets, and delimited external files, which are files containing columns of data values that are separated by a delimiter such as a blank or a comma.

IMPORT Procedure

The syntax for the IMPORT procedure is shown here briefly but is described in detail in the *Base SAS Procedures Guide*. See "Import/Export Overview for PC Files" on page 49 for a list of file formats supported under your operating environment.

PROC IMPORT
DATAFILE=*"filename"* | TABLE=*"tablename"*
OUT=<*libref.*> *SAS-data-set* <*(SAS-data-set-options)*>
<DBMS=*identifier*><REPLACE>;
<*data-source-statements*>;

After you invoke the IMPORT procedure, it reads the input file and writes the data to a SAS data set, where the names of the SAS variables are based on the column names of the input data. PROC IMPORT imports the data by one of the following methods:

- generated DATA step code
- generated SAS/ACCESS code
- translation engines.

You control the results with options and statements that are specific to your input data source. PROC IMPORT produces the specified SAS data set and writes information about the import to the SAS log. In the log, you see the DATA step or the SAS/ACCESS code that is generated by PROC IMPORT. If a translation engine is used, then the code is not submitted.

Example: Importing a Microsoft Access File

This example imports a Microsoft Access table called customers and from it creates a permanent SAS data set named sasuser.cust. The Microsoft Access table has user-level security and, therefore, you need to specify the following statements: PWD=, UID=, and WGDB=.

```
PROC IMPORT DBMS=ACCESS TABLE="customers" OUT=sasuser.cust;
     DATABASE="c:\demo\customers.mdb";
     UID="bob";                              /* Microsoft Access Database User ID */
     PWD="cat";                              /* Microsoft Access Database Password */
     WGDB="c:\winnt\system32\system.mdb";  /* Workgroup Administrator Database */
RUN;
proc print data=sasuser.cust;
run;
```

Note: See the *Base SAS Procedures Guide* for restrictions, interactions, and tips about PROC IMPORT. △

Example: Importing a Table from a Microsoft Excel Workbook File

This example imports a sheet (Invoice) in a Microsoft Excel workbook (sasdemo.xls), and from it creates a permanent SAS data set named work.invoice.

```
PROC IMPORT DBMS=EXCEL OUT= work.invoice
            DATAFILE= "c:\excel\sasdemo.xls" REPLACE ;
     VERSION='2002';   /* Excel File Version                        */
     SHEET="Invoice";     /* Sheet name                                 */
     GETNAMES=YES;        /* Use the first row of data as column names   */
     SCANTEXT=YES;   /* Scan all rows of data for the largest size  */
```

```
     USEDATE=YES;   /* Use DATE format for date/time columns    */
     SCANTIME=YES;   /* Scan and identify time columns        */
     DBSASLABEL=NONE;     /* Leave SAS label names to be nulls        */
     TEXTSIZE=512;   /* largest text size allowed               */
RUN;
```

Note: See the *Base SAS Procedures Guide* for restrictions, interactions, and tips about PROC IMPORT. △

Example: Importing a Locally available JMP File (Running SAS on UNIX)

This example imports a JMP file that is located on a local drive.

```
proc import dbms=jmp out=bicycle
             datafile="/jmp/stored/here/bicycle.jmp";
run;

proc print data=bicycle;
run;
```

Note: See the *Base SAS Procedures Guide* for restrictions, defaults, requirements, and limitations of PROC IMPORT. △

EXPORT Procedure

The syntax for the EXPORT procedure is shown here briefly but is described in detail in the *Base SAS Procedures Guide*. See "Import/Export Overview for PC Files" on page 49 for a list of file formats supported under your operating environment.

PROC EXPORT
DATA=<*libref.*>*SAS-data-set* <*(SAS-data-set-options)*>
OUTFILE=*"filename"* | OUTTABLE=*"tablename"*
<DBMS=*identifier*> <REPLACE>;

The EXPORT procedure reads data from a SAS data set and exports it to an external data source by using one of the following methods:

- generated DATA step code
- generated SAS/ACCESS code
- translation engines.

PROC EXPORT also controls the results with options and statements that are specific to the output data source.

Example: Exporting a Delimited File

The following example exports a SAS data set named myfile.class and creates a delimited external file called Class. Notice that the DELIMITER= statement specifies the ampersand (&) delimiter to separate the column names in the new file. This example is repeated from the *Base SAS Procedures Guide*; see it for the SAS log.

```
proc export data=myfiles.class
     outfile="/myfiles/class"
     dbms=dlm;
     delimiter='&';
run;
```

The following code shows the first five rows of the external file that PROC EXPORT produces:

```
NAMES&SEX&AGE&HEIGHT&WEIGHT
Alice&F&13&56.5&84
Becka&F&13&65.3&98
Gail&F&14&64.3&90
Karen&F&12&56.3&77
Kathy&F&12&59.8&84.5
```

Note: See the *Base SAS Procedures Guide* for restrictions, defaults, requirements, and limitations of PROC EXPORT. △

Example: Exporting a Table to a Microsoft Excel File on a PC Server

This example exports a data set (work.employee) to a Microsoft Excel workbook (sasdemo.xls) on a PC Server (Sales) and from it, creates a new sheet (employee) in the Excel workbook. This example is for SAS UNIX users using Client Server Model.

```
PROC EXPORT DBMS=EXCELCS DATA= work.employee
            OUTFILE= "c:\temp\sasdemo.xls" REPLACE;
   SHEET="Employee";
   VERSION="2002";    /* Excel Version */
   SERVER="sales";    /* Server Name   */
   SERVICE=PCFILE ;          /* Service Name  */
RUN;
```

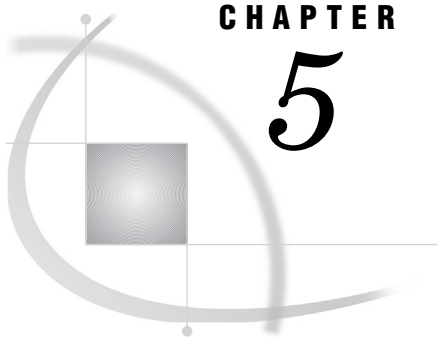
Note: See the *Base SAS Procedures Guide* for restrictions, interactions, and tips about PROC EXPORT. △

Example: Exporting a Locally Available JMP File

This example exports a JMP file that is located on a local drive.

```
PROC EXPORT DBMS=jmp DATA=results OUTFILE= "c:\invoicing\customers.jmp"
            REPLACE;
```

Note: See the *Base SAS Procedures Guide* for restrictions, interactions, and tips about PROC EXPORT. △



CHAPTER 5

The DBF and DIF Procedures

Introduction to the DBF and DIF Procedures

The DBF and DIF procedures give Windows, OS/390, and UNIX users an alternative way of accessing DBF and DIF files. Instead of creating access descriptors and view descriptors, you can convert these PC file types to SAS data sets, or vice versa.

Under UNIX and Windows operating environments, you can use the DBF and DIF procedures to convert a DBF or DIF file to a SAS data set or to convert a SAS data set to a DBF or DIF file. Under OS/390, only the DBF procedure is available.

See "Methods for Accessing PC Files Data" on page 3 for the other methods for accessing data in PC file formats under Windows, UNIX, and OpenVMS operating environments.

The DBF Procedure

Converts a dBASE file to SAS data set or a SAS data set to a dBASE file

Syntax

PROC DBF *options*;

PROC DBF Options

DB2|DB3|DB4|DB5=*fileref* | *filename*
specifies the fileref or filename of a DBF file. The DB*n* option must correspond to the version of dBASE with which the DBF file is compatible. You specify the version with the DB*n* option, where *n* is the version number and can have a value of 2, 3, 4, or 5.

If you specify a fileref, the FILENAME statement that you used to define it must specify the filename plus a .dbf extension (for example, **`filename myref '/my_dir/myfile.dbf'`**).

If you specify a filename instead of a fileref, you can only specify the name itself (omitting the .dbf extension) and the file must be in the current directory. For

example, this PROC DBF statement creates the EMP.DBF file (with the name in uppercase) from the MyLib.Employee data set:

```
proc dbf db5=emp data=mylib.employee;
```

You *cannot* specify **emp.dbf** or a full pathname (**proc dbf db5='/my/unix_directory/emp.dbf'**).

The DB*n*= option is required.

DATA=<*libref.*>*member*
names the input SAS data set. Use this option if you are creating a DBF file from a SAS data set. If you use the DATA= option, do not use the OUT= option. If you omit the DATA= option, SAS software creates an output SAS data set from the DBF file.

OUT=<*libref.*>*member*
names the SAS data set that is created to hold the converted data. Use this option only if you are creating a SAS data set from a DBF file and you did not specify the DATA= option.

If OUT= is omitted, SAS creates a temporary data set in the Work library. (Under UNIX and OS/390, the temporary data set is named Data1 [...Data*n*]; under Windows, it is called _DATA_.) If OUT= is omitted or if you do not specify a two-level name in the OUT= option, the data set remains available during your current SAS session, but it is not permanently saved.

Details

The DBF procedure converts dBASE files to SAS data sets that are compatible with the current release of SAS, or it converts SAS data sets to DBF files.

PROC DBF produces one output file but no printed output. The output file contains the same information as the input file but in a different format.

The DBF procedure works with DBF files created by all the current versions and releases of dBASE (II, III, III PLUS, IV, and 5.0) and with most DBF files that are created by other software products.

Future versions of dBASE files might not be compatible with the current version of the DBF procedure. To use the DBF procedure, you must have a SAS/ACCESS interface to PC files license.

Converting DBF Fields to SAS Variables

Numeric variables are stored in character form by DBF files. These numeric variables become SAS numeric variables when converted from a DBF file to a SAS data set. If a DBF numeric value is missing, the corresponding dBASE numeric field is filled with the character **9**, by default.

Character variables become SAS character variables. Logical fields become SAS character variables with a length of 1. Date fields become SAS date variables. When you are converting a DBF file to a SAS data set, fields whose data is stored in auxiliary DBF files (Memo and General fields) are ignored.

When a dBASE II file is translated into a SAS data set, any colons in dBASE variable names are changed to underscores in SAS variable names. Conversely, when a SAS data set is translated into a dBASE file, any underscores in SAS variable names are changed to colons in dBASE field names.

Converting SAS Variables to DBF Fields

Numeric variables are stored in character form by DBF files. SAS numeric variables become numeric variables with a length of 16 when converting from a SAS data set to a DBF file. A SAS numeric variable with a decimal value must be stored in a decimal

format in order to be converted to a DBF numeric field with a decimal value. In other words, unless you associate the SAS numeric variable with an appropriate format in a SAS FORMAT statement, the corresponding DBF field will not have any value to the right of the decimal point. You can associate a format with the variable in a SAS data set when you create the data set or by using the DATASETS procedure.

If the number of digits — including a possible decimal point — exceeds 16 a warning message is issued and the DBF numeric field is filled with the character **9**. All SAS character variables become DBF fields of the same length. When you are converting data from a SAS data set to a DBF file that is compatible with dBASE III or later, SAS date variables become DBF date fields. When you are converting data from a SAS data set to a dBASE II file, SAS date variables become dBASE II character fields in the form *YYYYMMDD*.

Transferring Other Software Files to DBF Files

You might find it helpful to save another software vendor's file to a DBF file and then convert that file into a SAS data set. UNIX users find this especially helpful. For example, you could save an Excel XLS file to a DBF file (by selecting

File ► Save As

from within an Excel spreadsheet and selecting the Emp.dbf file) and then use PROC DBF to convert that file into a SAS data set. Or you could do the reverse: use PROC DBF to convert a SAS data set into a DBF file and then load that file into an Excel spreadsheet.

Examples for UNIX

Example 1: Converting a dBASE II File to a SAS Data Set In this example, a dBASE II file named Employee.dbf is converted to a SAS data set. Because no FILENAME statement is specified, the last level of the filename is assumed to be .dbf and the file is assumed to be in your current directory and in uppercase.

```
libname save '/my/unx_save_dir';
proc dbf db2=employee out=save.employee;
run;
```

Example 2: Converting a SAS Data Set to a dBASE 5 File In this example, a SAS data set is converted to a dBASE 5 file. A FILENAME statement specifies a fileref that names the dBASE 5 file. You must specify the FILENAME statement before the PROC DBF statement.

```
libname mylib '/my/unix_directory';
filename employee '/sasdemo/employee.dbf';
proc dbf db5=employee data=mylib.employee;
run;
```

In a Windows environment, this example would be:

```
libname mylib 'c:\my\directory';
filename employee 'c:\sasdemo\employee.dbf';
proc dbf db5=employee data=mylib.employee;
run;
```

In an OS/390 environment, this example would be:

```
libname mylib 'sasdemo.employee.data';
filename dbfout 'sasdemo.newemp.dbf' recfm=n;
proc dbf db5=dbfout data=mylib.employee;
run;
```

The DIF Procedure

Converts a DIF file to SAS data set or a SAS data set to a DIF file

Restrictions: The DIF procedure is only available under UNIX and Windows operating environments.

Syntax

PROC DIF *options*;

PROC DIF Options

DIF=*fileref* | *filename*
specifies the fileref or filename of a DIF file.

If you specify a fileref, the FILENAME statement that you used to define it must specify the filename plus a .dif extension (for example, **`filename myref '/my_dir/myfile.dif'`**).

If you specify a filename instead of a fileref, you can only specify the name itself (omitting the .dif extension) and the file must be in the current directory. For example, this PROC DIF statement creates the Emp.dif file from the MyLib.Employee data set:

```
proc dif dif=emp data=mylib.employee;
```

You *cannot* specify **`emp.dif`** or a full pathname (**`proc dif dif='/my/unix_directory/emp.dif'`**).

DATA=<*libref.*>*member*
names the input SAS data set. Use this option if you are creating a DIF file from a SAS data set. If you use this option, do not use the OUT= option. If you omit the DATA= option, SAS creates an output SAS data set from the DIF file.

OUT=<*libref.*>*member*
names the SAS data set to hold the converted data. You use this option only if you omit the DATA= option and you are creating a SAS data set from a DIF file.

If OUT= is omitted, SAS creates a temporary data set in the Work library. (Under UNIX, the temporary data set is named Data1 [...Data*n*]; under Windows, it is called _DATA_. If OUT= is omitted or if you do not specify a two-level name in the OUT= option, the data set remains available during your current SAS session but is not permanently saved.

LABELS
causes PROC DIF to write the names of the SAS variables as the first row of the DIF file and a row of blanks as the second row of the DIF file. The actual data portion of

the DIF file begins in the third row. The LABELS option is allowed only when you are converting a SAS data set to a DIF file.

PREFIX=*name*
specifies a prefix to be used in constructing SAS variable names when you are converting a DIF file to a SAS data set. For example, if PREFIX=VAR, the new variable names are VAR1, VAR2, ... VAR*n*. If you omit the PREFIX= option, PROC DIF assigns the names Col1, Col2, ... Col*n*.

SKIP=*n*
specifies the number of rows, beginning at the top of the DIF file, to be ignored when converting a DIF file to a SAS data set. For example, suppose the first row of your DIF file contains column headings and the second row of your DIF file is a blank row. The actual data in your DIF file begin in row 3. You should specify SKIP=2 so that PROC DIF ignores the nondata portion of your DIF file. Alternatively, you could delete the first two rows of your DIF file before using PROC DIF.

Details

The DIF procedure converts data interchange format (DIF) files to SAS data sets that are compatible with the current release of SAS software, or it converts SAS data sets to DIF files.

PROC DIF produces one output file but no printed output. The output file contains the same information as the input file but in a different format.

Software Arts, Inc. developed the data interchange format to be used as a common language for data. Originally, DIF was made popular by products such as Lotus 1-2-3 and VisiCalc. Although DIF is not as popular today as it once was, it is still supported by many software products.

Note: Any DIF file that you plan to convert to a SAS data set should be in a tabular format. All items in a given column should represent the *same* type of data. If any rows in the DIF file contain inconsistent data — for example, a row of underscores, dashes, or blanks — delete these rows before converting the DIF file to a SAS data set. It is recommended that you make a backup copy of your DIF table before you make these modifications. △

When you are converting data from a DIF file to a SAS data set, each row of the DIF file becomes an observation in the SAS data set. Conversely, when you are converting a SAS data set to a DIF file, each SAS observation becomes a row in the DIF file. To use the DIF procedure, you must have a SAS/ACCESS interface to PC files license.

Converting DIF Variables to SAS Variables

Character variables in a DIF file (sometimes referred to as *string values*) become SAS character variables of length 20. If a DIF character variable's value is longer than 20 characters, it is truncated to a length of 20 in the SAS output data set. The quotation marks that normally enclose character variable values in a DIF file are removed when the value is converted to a SAS character value.

Numeric variables, which can be represented in either integer or scientific notation in a DIF file, become SAS numeric variables when a DIF file is converted to a SAS data set.

Transferring SAS Data Sets to and from Other Software Products Using DIF

DIF files are not generally used as the native file format for a software product's data storage. Therefore, transferring data between SAS and another software product is a two-step process when using DIF files.

To send SAS data sets to another software product using DIF files, you must first run PROC DIF to convert your SAS data set to a DIF file. Use whatever facility is provided by the target software product to read the DIF file. For example, you use the Lotus 1-2-3 Translate Utility to translate a DIF file to a 1-2-3 worksheet file. (This facility might be provided by an import tool or from an Open window in that software product.) After the application reads the DIF file data, the data can be manipulated and saved in the application's native format.

To transfer data in the opposite direction — from a software product to a SAS data set — the process is reversed. First, export the data to a DIF file and then run PROC DIF to read the DIF file into a SAS data set.

Missing Values

The developers of the data interchange format (DIF) files suggest that you treat all numeric values that have a value indicator other than V as missing values. PROC DIF follows this convention. When a DIF file is converted to a SAS data set, any numeric value with a value indicator other than V becomes a SAS missing value.

When a SAS data set that has missing values for some numeric variables is converted to a DIF file, the following assignments are made in the DIF file for the variables with missing values:

- the type indicator field value is set to **0**
- the number field value contains a string of 16 blanks
- the value indicator is set to **NA**.

Examples

Example 1: Converting a DIF File to a SAS Data Set In this example, a DIF file named Employee.dif is converted to a SAS data set. Because no FILENAME statement is specified, the last level of the filename is assumed to be .dif, and the file is assumed to be in your current directory and in uppercase.

```
libname save '/my/my_unx_dir';
proc dif dif=employee out=save.employee;
run;
```

Example 2: Converting a SAS Data Set to a DIF File In this example, a SAS data set is converted to a DIF file. A FILENAME statement is used to specify a fileref that names the DIF file. You must specify the FILENAME statement before the PROC DIF statement.

```
filename employee 'c:\sasdemo\employee.dif';
proc dif dif=employee data=save.employee;
run;
```

Or, in a UNIX environment, this example would be:

```
filename employee '/sasdemo/employee.dif';
proc dif dif=employee data=save.employee;
run;
```

See Also

"Programmer's Guide to the DIF," *Software Arts Technical Notes* (SATN-18).

CHAPTER 6

The ACCESS Procedure for PC Files

Overview of the ACCESS Procedure for PC Files

The ACCESS procedure for PC files is only available under Windows operating environments. You can use the ACCESS procedure with DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats. See "Methods for Accessing PC Files Data" on page 3 for alternate methods for accessing data in PC file formats under Windows, UNIX, OS/390, and OpenVMS operating environments.

The ACCESS procedure enables you to create access descriptors, view descriptors, and SAS data files. Descriptor files describe PC files data so that you can directly read, update, or extract the PC files data while working within a SAS program. See "SAS/ ACCESS Descriptors for PC Files" on page 67 for more information.

CAUTION:

Altering a PC file might invalidate defined descriptors. Altering the format of a PC file that has descriptor files defined on it might cause these descriptors to be out-of-date or invalid. For example, if you add a column to a file and an existing access descriptor is defined on that file, the access descriptor and any view descriptors based on it do not show the new column. You must re-create the descriptors to be able to show and select the new column. △

Using ACCESS Procedure Statements

The following table presents a task-oriented overview of the statements you use inside a PROC ACCESS program block to create or modify access and view descriptors. See "ACCESS Procedure Syntax" on page 71 for the complete syntax for this procedure.

Table 6.1 Options and Statements Required for the ACCESS Procedure

Task	Options and Statements You Use
create an access descriptor	**PROC ACCESS** DBMS=DBF|DIF|WK*n*|XLS; **CREATE** *libref.member-name*.ACCESS; *required-database-description-statements*; *optional-editing-statements*; **RUN**;
create an access descriptor and a view descriptor	**PROC ACCESS** DBMS=DBF|DIF|WK*n*| XLS; **CREATE** *libref.member-name*.ACCESS; *required-database-description-statements*; *optional-editing-statements*; **CREATE** *libref.member-name*.VIEW; **SELECT** *column-list*; *optional-editing-statements*; **RUN**;
create a view descriptor from an existing access descriptor	**PROC ACCESS** DBMS=DBF|DIF|WK*n*|XLS ACCDESC=*libref.access-descriptor*; **CREATE** *libref.member-name*.VIEW; **SELECT** *column-list*; *optional-editing-statements*; **RUN**;

As the table indicates, you can create one or more access descriptors and view descriptors in one execution of PROC ACCESS, or you can create the descriptors in separate executions. See "CREATE Statement" on page 74 for additional information about statement order.

SAS/ACCESS Descriptors for PC Files

SAS/ACCESS descriptor files are the tools that the ACCESS procedure uses to establish a connection to a PC file. To create descriptor files, use the ACCESS procedure. There are two kinds of descriptor files: access descriptors and view descriptors. The following sections give a brief overview of these files.

Access Descriptors

An *access descriptor* holds essential information about the structure of the PC file that you want to access. For example, you can access the file's format and name, its database field or column names, and its data types. Access descriptors can also contain the corresponding SAS information such as the SAS variable names and formats. Typically, you have only one access descriptor for each PC file.

An access descriptor only describes a PC file's format and contents to SAS; that is, it is a master description file of the PC file for SAS. You cannot use an access descriptor in a SAS program. Rather, you use an access descriptor to create other SAS files, called view descriptors, that you use in SAS programs.

View Descriptors

A *view descriptor* defines some or all of the data that is described by one access descriptor (and, therefore, one PC file). For example, you might want to use only three of nine possible database columns and only some of the rows in a PC file. The view descriptor enables you to do this by selecting the database fields or columns that you want to use and specifying criteria to retrieve only the rows you want. Typically, you create several view descriptors based on one access descriptor, where each view descriptor selects a different subset of the PC files data.

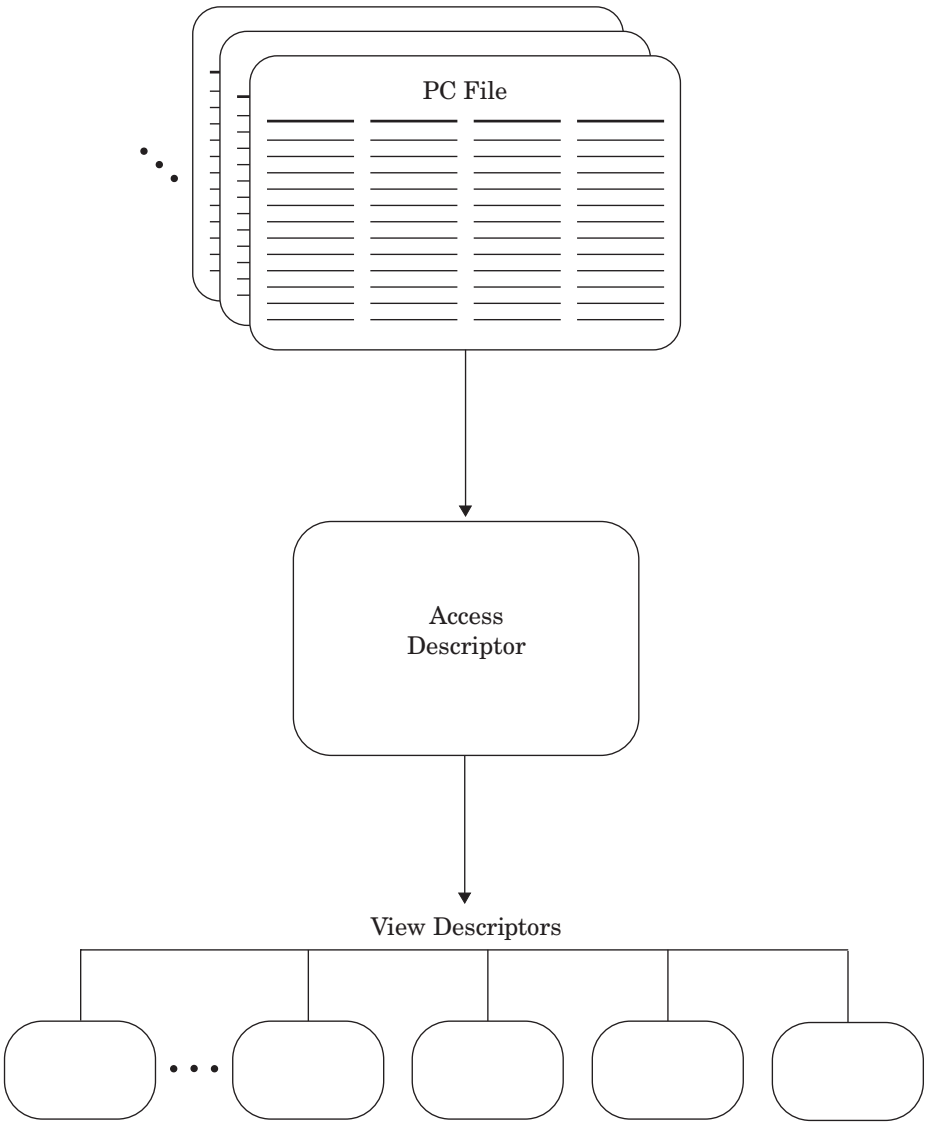
A view descriptor is a SAS data set or, more specifically, a SAS data view. You use a view descriptor in a SAS program much as you would any SAS data set. For example, you can specify a view descriptor in the DATA= statement of a SAS procedure or the SET statement of a DATA step. You can use a view descriptor in a SELECT statement of the SQL procedure to join, for example, the view descriptor's data with SAS data.

You can use a view descriptor to update data directly in some of the PC file formats, such as the DBF file format. For example, you can use a view descriptor to add records or mark records for deletion in a DBF file or to change the values in a DBF file field by using the DBF, FSEDIT, or SQL procedures. You can also modify DBF file data by specifying a view descriptor in the MODIFY or REPLACE statements in a DATA step. See the "Essentials" section in the appropriate chapter for information on whether a PC file format allows updates.

In some cases, you might also want to create a SAS data file from data stored in a PC file. Using a view descriptor to copy PC files data into a SAS data file is called *extracting* the data. You can extract PC files data in a number of ways, for example, by specifying a view descriptor when you are using various methods within the ACCESS procedure. Or you could specify a view descriptor in a DATA step or in a SAS procedure's OUT= option. When you need to use the same PC files data in a number of SAS procedures or DATA steps, extracting the PC files data into a SAS data file might use fewer resources than directly accessing the data repeatedly.

The following figure illustrates the relationships between a PC file, an access descriptor, and one or more view descriptors.

Figure 6.1 Relationships between a PC File, an Access Descriptor, and View Descriptors



SAS Passwords for Descriptors

SAS enables you to control access to SAS data sets and access descriptors by associating one or more SAS passwords with them. You must first create the descriptor files before assigning SAS passwords to them, as described in "Assigning Passwords" on page 69. The following table summarizes the levels of protection that SAS passwords have and their effects on access descriptors and view descriptors.

Table 6.2 Password and Descriptor Interaction

	READ=	WRITE=	ALTER=
access descriptor	no effect on descriptor	no effect on descriptor	protects descriptor from being read or updated
view descriptor	protects PC file data from being read or updated	protects PC file data from being updated	protects descriptor from being read or updated

When you create view descriptors, you can use a SAS data set option after the ACCDESC= option to specify the access descriptor's password (if one exists). In this case, you are *not* assigning a password to the view descriptor that is being created. Rather, using the password grants you permission to use the access descriptor to create the view descriptor. For example:

```
proc access dbms=dbf
     accdesc=adlib.customer(alter=rouge);
  create vlib.customer.view;
  select all;
run;
```

By specifying the ALTER-level password, you can read the AdLib.Customer access descriptor and therefore create the VLib.Customer view descriptor.

For detailed information on the levels of protection and the types of passwords you can use, refer to your Base SAS software documentation.

Assigning Passwords

To assign, change, or delete a SAS password, use the DATASETS procedure's MODIFY statement. Here is the basic syntax for using PROC DATASETS to assign a password to an access descriptor, a view descriptor, or a SAS data file:

PROC DATASETS LIBRARY=*libref* MEMTYPE=*member-type*;

MODIFY *member-name* (*password-level*=*password-modification*);

RUN;

The *password-level* argument can have one or more of the following values: READ=, WRITE=, ALTER=, or PW=. PW= assigns read, write, and alter privileges to a descriptor or data file. The *password-modification* argument enables you to assign a new password or to change or delete an existing password. For example, this PROC DATASETS statement assigns the password MONEY with the ALTER level of protection to the access descriptor AdLib.Salaries

```
proc datasets library=adlib memtype=access;
   modify salaries (alter=money);
run;
```

In this case, you are prompted for the password whenever you try to browse or edit the access descriptor or to create view descriptors that are based on AdLib.Salaries.

In the next example, the PROC DATASETS statement assigns the passwords MYPW and MYDEPT with READ and ALTER levels of protection to the view descriptor VLib.JobC204:

```
proc datasets library=vlib memtype=view;
   modify jobc204 (read=mypw alter=mydept);
run;
```

In this case, you are prompted for the SAS passwords when you try to read the PC file data, or try to browse or edit the view descriptor VLib.JobC204 itself. You need both levels to protect the data and descriptor from being read. However, you could still update the data accessed by VLib.JobC204, for example, by using a PROC SQL UPDATE statement. Assign a WRITE level of protection to prevent data updates.

To delete a password on an access descriptor or any SAS data set, put a slash after the password:

```
proc datasets library=vlib memtype=view;
   modify jobc204 (read=mypw/ alter=mydept/);
run;
```

Performance and Efficient View Descriptors for PC Files

General Guidelines

When you create and use view descriptors, follow these guidelines to minimize the use of SAS resources and to reduce the time it takes to access data:

- Select only the columns your SAS program needs. Selecting unnecessary columns adds extra processing time.
- Where possible, specify selection criteria to subset the number of observations processed by SAS.
- To present PC files data in sorted order, reference a view descriptor in a PROC SQL query. Otherwise, you might need to extract the data to sort it, as described below.

Extracting Data Using a View

In some cases, it might be more efficient to use a view descriptor to extract PC files data and place it in a SAS data file instead of using the view descriptor to read the data directly.

A PC file is read every time a view descriptor is referred to in a SAS program and the program is executed; the program's output reflects the latest updated level of the PC file. If many users are reading the same PC file repeatedly, performance might decrease. If you create several reports during the same SAS session, they might not be based on the same PC files data due to updates by other users. Therefore, in the following circumstances, it is better to extract data:

- Extract PC files data if the file is large and you use the data repeatedly in SAS programs.

  If a view descriptor describes a large PC file and you plan to use the same data in several procedures or DATA steps during the same SAS session, you might improve performance by extracting the data. Placing the data into a SAS data file requires a certain amount of disk space to store the data and I/O to write the data. However, SAS data files are organized to provide optimal performance with PROC and DATA steps. Programs that use SAS data files are often more efficient than programs that read PC files data directly.

- Extract PC files data if you use sorted data several times in a SAS program.

  If you intend to use PC file data in a particular sorted order several times, run the SORT procedure on the view descriptor using the OUT= option to extract the data. The OUT= option is required whenever PROC SORT references a view descriptor. Extracting the data in this way is more efficient than requesting the

same sort repeatedly on the PC files data. Note that using the ORDER BY clause in the SQL procedure does not sort the data in the physical PC file; it only presents the data in a sorted order.

- Extract PC files data for added security.

  If you are the owner of a PC file and do not want anyone else to read the data, you might want to extract the data (or a subset of the data) and not distribute information about either the access descriptor or view descriptor. Or, you might want to assign PC file security features to your PC files to prevent unauthorized reading or writing to them.

  On the SAS side, you might also want to assign SAS passwords to your descriptors for additional security. If a view descriptor has a password assigned to it and you extract the data, the new SAS data file is automatically assigned the same password. If a view descriptor does not have a password, you can assign a password to the extracted SAS data file.

ACCESS Procedure Syntax

The general syntax for the ACCESS procedure is presented here; see the DBF, DIF, WK*n*, MDB, and XLS chapters for file format specific information.

PROC ACCESS *options*;

Create and Update Statements

CREATE *libref.member-name.*ACCESS|VIEW ;

UPDATE *libref.member-name.*ACCESS|VIEW ;

Database Description Statement

PATH= *'path-and-filename<.PC-filename-extension>'* | *<'>filename<'>* | *fileref*;

(See your file format-specific chapter for additional database-description statements.)

Editing Statements

ASSIGN <=>YES|NO|Y|N;

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 <'>>*;

MIXED <=> YES | NO | Y | N;

(The MIXED statement is not available for DIF and DBF files.)

QUIT;

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*;

TYPE *<'>column-identifier-1<'>*<=> C | N
<…column-identifier-n <=> C | N>;

(The TYPE statement is not available for DBF files.)

UNIQUE <=>YES|NO|Y|N ;

RUN;

PROC ACCESS Statement

Requirement: This statement is required.

Valid: for DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 files under Windows operating environments

PROC ACCESS *options*;

Options

The PROC ACCESS statement options that are available with all supported PC file formats are described below. Other options, specific to particular PC file formats, are described in the file format specific chapters.

DBMS=*pc-file-format*
specifies which PC database product or spreadsheet system you want to access from SAS. This is the only required option. The valid types are DBF, DIF, WK1, WK3, WK4, and XLS.

ACCDESC=*libref.access-descriptor* <(READ|WRITE|ALTER=*password*)>
specifies an existing access descriptor.

Use this option when creating or updating a view descriptor based on an access descriptor that was created in a separate PROC ACCESS step.

You name the view descriptor in the CREATE statement. You can also use a SAS data set option on the ACCDESC= option to specify a SAS password for the access descriptor.

The ACCDESC= option has two aliases: AD= and ACCESS=.

VIEWDESC=*libref.view-descriptor*
specifies a view descriptor as input for the OUT= option. (See the description of OUT=.)

OUT= <*libref.*>*member-name*
specifies a SAS data file. When VIEWDESC= and OUT= are used together, you can write data that is accessed from the view descriptor to the SAS data set that is specified in OUT=. For example:

```
proc access viewdesc=vlib.invq4
            out=dlib.invq4;
run;
```

ASSIGN Statement

Indicates whether SAS variable names and formats are automatically generated

Valid: for DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments

Applies to: access descriptor

Interacts with: FORMAT, RENAME, RESET, UNIQUE

Not allowed with: UPDATE

Default: NO

Alias: AN

ASSIGN <=> YES|NO|Y|N;

Details

The ASSIGN statement indicates whether SAS variable names and formats are automatically generated. Where long names must be shortened to the SAS length limit of 8 characters, variable names are automatically generated.

An editing statement such as ASSIGN appears after the CREATE and database-description statements. See "CREATE Statement" on page 74 for more information.

The value NO (or N) enables you to modify SAS variable names and formats when you create an access descriptor and when you create view descriptors that are based on this access descriptor. During an access descriptor's creation, you use the RENAME statement to change SAS variable names, and you use the FORMAT statement to change SAS formats.

Specify a YES (or Y) value for this statement to generate unique SAS variable names from the first 8 characters of the PC file column names, according to the rules listed below. With YES, you can change the SAS variable names only in the access descriptor. The SAS variable names that are saved in an access descriptor are *always* used when view descriptors are created from the access descriptor; you cannot change them in the view descriptors.

Default SAS variable names are generated according to these rules:

- If the column name is longer than 8 characters, SAS uses only the first 8 characters. If truncating results in duplicate names, numbers are appended to the ends of the names to prevent duplicate names. For example, the names clientsname and clientsnumber become the SAS names clientsn and clients0.
- If the column name in the PC file contains blank characters, SAS ignores the blank characters. For example, the column name Paid On becomes the SAS name PaidOn.
- If the column name in the PC file starts with a digit (0 through 9), SAS adds the character Z before it. For example, the column name 1stYear becomes the SAS name Z1stYear.
- If the column name contains characters that are invalid in SAS names (including national characters), SAS replaces the invalid characters with underscores (_). For example, the column name $Paid becomes the SAS variable name _Paid

If you specify YES for this statement, SAS automatically resolves any duplicate variable names. However, if you specify YES, you cannot specify the RENAME, FORMAT, RESET, or UNIQUE statements when you create view descriptors that are based on the access descriptor. When you are updating an access descriptor, you cannot specify the ASSIGN statement.

When the SAS/ACCESS interface encounters the next CREATE statement to create an access descriptor, the ASSIGN statement is reset to the default NO value.

CREATE Statement

Creates a SAS/ACCESS descriptor file

Requirement: This statement is required.

Valid: for DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments

Applies to: access descriptor or view descriptor

CREATE *libref.descriptor-name*.ACCESS|VIEW;

Details

Use CREATE to create an access or view descriptor for a PC file you want to access from SAS. To access a particular PC file of a supported type, you must create first an access descriptor, and then one or more view descriptors based on the access descriptor.

The descriptor name has three parts, separated by periods(.). The *libref* identifies a SAS data library, which is associated with a directory on the local system's disk where the descriptor will be created. The *libref* must already have been created using the LIBNAME statement. The *descriptor-name* is the name of the descriptor to be created. The third part is the descriptor type. Specify ACCESS for an access descriptor or VIEW for a view descriptor.

You can use the CREATE statement as many times as necessary in one procedure execution. That is, you can create multiple access descriptors, as well as one or more view descriptors based on these access descriptors, within the same execution of the ACCESS procedure. Or, you can create access descriptors and view descriptors in separate executions of the procedure.

You can use CREATE and UPDATE in the same PROC ACCESS block with one restriction: a CREATE statement for a view descriptor may not follow an UPDATE statement.

Creating Access Descriptors

When you create an access descriptor, you must place statements or groups of statements in a certain order after the PROC ACCESS statement and its options, as listed here:

1 CREATE must be the first statement after the PROC ACCESS statement with one exception: if the block includes both CREATE and UPDATE statements, either statement may be the first in the block.

2 Next, specify any database-description statement, such as PATH=. This information describes the location and characteristics of the PC file. These statements must be placed before any editing statements. Do not specify these statements when you create view descriptors.

   Information from database-description statements is stored in an access descriptor. Therefore, you do not repeat this information when you create view descriptors.

3 Next, specify any editing statements: ASSIGN, DROP, FORMAT, LIST, RENAME, RESET, and SUBSET. QUIT is also an editing statement, but using it terminates PROC ACCESS without creating your descriptor.

4 Finally, specify the RUN statement. RUN executes the ACCESS procedure.

The order of the statements within the database-description and editing groups sometimes matters; see the individual statement descriptions for more information.

Note: Altering a PC file that has descriptor files defined on it might cause the descriptor files to be out-of-date or invalid. For example, if you re-create a file and add a new column to the file, an existing access descriptor defined on that file does not show that column, but the descriptor can still be valid. However, if you re-create a file and delete an existing column from the file, the descriptor might be invalid. If the deleted column is included in a view descriptor and this view is used in a SAS program, the program fails and an error message is written to the SAS log. △

Creating View Descriptors

You can create view descriptors and access descriptors in the same ACCESS procedure or in separate procedures.

To create a view descriptor and the access descriptor on which it is based within the *same* PROC ACCESS execution, you must place the statements or groups of statements in a particular order after the PROC ACCESS statement and its options, as listed below:

1 First, create the access descriptor as described in “Creating Access Descriptors” on page 74, except omit the RUN statement.

2 Next, specify the CREATE statement for the view descriptor. The CREATE statement must follow the PROC ACCESS statements that you used to create the access descriptor.

3 Next, specify any editing statements: SELECT, SUBSET, and UNIQUE are valid only when creating view descriptors. FORMAT, LIST, RENAME, and RESET are valid for both view and access descriptors. FORMAT, RENAME, and UNIQUE can be specified only when ASSIGN=NO is specified in the access descriptor referenced by this view descriptor. QUIT is also an editing statement but using it terminates PROC ACCESS without creating your descriptor.

   The order of the statements within this group usually does not matter; see the individual statement descriptions for any restrictions.

4 Finally, specify the RUN statement. RUN executes PROC ACCESS.

To create a view descriptor based on an access descriptor that was created in a *separate* PROC ACCESS step, you specify the access descriptor’s name in the ACCDESC= option in the new PROC ACCESS statement. You must specify the CREATE statement before any of the editing statements for the view descriptor.

If you create only one descriptor in a PROC step, the CREATE statement and its accompanying statements are checked for errors when you submit PROC ACCESS for processing. If you create multiple descriptors in the same PROC step, each CREATE statement (and its accompanying statements) is checked for errors as it is processed.

If no errors are found when the RUN statement is processed, all descriptors are saved. If errors are found, error messages are written to the SAS log, and processing is terminated. After you correct the errors, resubmit your statements.

Examples

The following example creates the access descriptor AdLib.Product for the worksheet file named **c:\sasdemo\specprod.wk4**:

```
libname adlib 'c:\sasdata';

proc access dbms=wk4;
   create adlib.product.access;
   path='c:\sasdemo\specprod.wk4';
   getnames=yes;
   assign=yes;
   rename productid prodid
          fibername fiber;
   format productid  4.
          weight     e16.9
          fibersize  e20.13
          width      e16.9;
run;
```

The following example creates an access descriptor named AdLib.Employ for the Excel worksheet named **c:\dubois\employ.xls**. It also creates a view descriptor named VLib.Emp1204 for this same file:

```
libname adlib 'c:\sasdata';
libname vlib 'c:\sasviews';

proc access dbms=xls;
   /* create access descriptor  */
   create adlib.employ.access;
   path='c:\dubois\employ.xls';
   getnames=yes;
   assign=no;
   list all;

   create vlib.emp1204.view;
   /* create view descriptor  */
   select empid lastname hiredate salary
          dept gender birthdate;
   format empid 6.
          salary dollar12.2
          jobcode 5.
          hiredate datetime7.
          birthdate datetime7.;
   subset where jobcode=1204;
run;
```

The following example creates a view descriptor VLib.BDays from the AdLib.Employ access descriptor, which was created in the previous PROC ACCESS step. Note that FORMAT could be used because the access descriptor was created with ASSIGN=NO.

```
libname adlib 'c:\sasdata';
libname vlib 'c:\sasviews';
```

```
proc access accdesc=adlib.employ;
   create vlib.bdays.view;
   select empid lastname birthdate;
   format empid 6.
          birthdate datetime7.;
run;
```

DROP Statement

Drops a column from a descriptor

Valid: for DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments

Applies to: access descriptor, view descriptor

Interacts with: RESET, SELECT, UPDATE

DROP <'>*column-identifier-1*<'>
<…<'>*column-identifier-n*<'>>;

Details

The DROP statement drops the specified column from an access descriptor. The column cannot be selected for a view descriptor that is based on the access descriptor. However, the specified column in the PC file remains unaffected by this statement.

Note: The DROP statement can only be specified when creating or updating an access descriptor, or when you are updating a view descriptor. DROP is not allowed when you are creating a view descriptor. When you use the UPDATE statement, you can specify DROP to remove a column from the view descriptor. However, the specified column in the PC file remains unaffected by the DROP statement. △

An editing statement, such as DROP, must follow the CREATE and database-description statements when you create an access descriptor. See "CREATE Statement" on page 74 for more information about the order of statements.

The *column-identifier* argument can be either the column name or the positional equivalent from the LIST statement, which is the number that represents the column's place in the access descriptor or view descriptor. For example, to drop the third and fifth columns, submit the following statement:

```
drop 3 5;
```

If the column name contains lowercase characters, special characters, or national characters, enclose the name in quotation marks. You can drop as many columns as you want in one DROP statement.

To display a column that was previously dropped, specify that column name in the RESET statement. However, doing so also resets all the column's attributes (such as SAS variable name, format, and so on) to their default values.

FORMAT Statement

Changes a SAS format for a PC file column

Valid: for DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments

Applies to: access descriptor or view descriptor

Interacts with: ASSIGN, DROP, RESET

FORMAT|FMT <'>*column-identifier-1*<'><=>*SAS-format-name-1*
<…<'>*column-identifier-n*<'><=>*SAS-format-name-n*>;

Details

The FORMAT statement changes a SAS variable format from its default format; the default SAS variable format is based on the data type and format of the PC file column. (See your PC file's chapter for information about the default data types and formats that SAS assigns to PC files data.)

An editing statement, such as FORMAT, must follow the CREATE statement and the database-description statements when you create a descriptor. See "CREATE Statement" on page 74 for more information about the order of statements.

The *column-identifier* argument can be either the column name or the positional equivalent from the LIST statement, which is the number that represents the column's place in the access descriptor. For example, to associate the DATE9. format with the BIRTHDATE column and with the second column in the access descriptor, submit the following statement:

```
format 2=date9. birthdate=date9.;
```

The column identifier is specified on the left and the SAS format is specified on the right of the expression. The equal sign (=) is optional. If the column name contains lowercase characters, special characters, or national characters, enclose the name in quotation marks. You can enter formats for as many columns as you want in one FORMAT statement.

You can use the FORMAT statement with a view descriptor only if the ASSIGN statement that was used when creating the access descriptor was specified with the **NO** value.

Note: When you use the FORMAT statement with access descriptors, the FORMAT statement also reselects columns that were previously dropped with the DROP statement. △

LIST Statement

Lists columns in the descriptor and gives information about them

Valid: for DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments

Applies to: access descriptor or view descriptor

Default: ALL

LIST <ALL|VIEW|<'>*column-identifier*<'>>;

Details

The LIST statement lists columns in the descriptor along with information about the columns. You can use the LIST statement when creating an access descriptor or a view descriptor. The LIST information is written to your SAS log.

If you use an editing statement, such as LIST, it must follow the CREATE statement and the database-description statements when you create a descriptor. You can specify LIST as many times as you want while creating a descriptor; specify LIST last in your PROC ACCESS code to see the entire descriptor. Or, if you are creating multiple descriptors, specify LIST before the next CREATE statement in order to list all the information about the descriptor you are creating.

The LIST statement can take one or more of the following arguments:

ALL
: lists all the columns in the PC file, the positional equivalents, the SAS variable names, and the SAS variable formats that are available for the access descriptor. When you are creating an access descriptor, ***NON-DISPLAY*** appears next to the column description for any column that has been dropped. When you are creating a view descriptor, ***SELECTED*** appears next to the column description for columns that you have selected for the view.

VIEW
: lists all the columns that are selected for the view descriptor, along with their positional equivalents, their SAS names and formats, and any subsetting clauses. Any columns that were dropped in the access descriptor are not displayed. The VIEW argument is valid only for a view descriptor.

column-identifier
: lists the specified column name, its positional equivalent, its SAS variable name and format, and whether the column has been selected. If the column name contains lowercase characters, special characters, or national characters, enclose the name in quotation marks.

  The *column-identifier* argument can be either the column name or the positional equivalent, which is the number that represents the column's place in the descriptor. For example, to list information about the fifth column in the descriptor, submit the following statement:

```
list 5;
```

You can use one or more of these previously described options in a LIST statement, in any order.

MIXED Statement

Determines whether to convert numeric data values in a column to their character representation when the corresponding SAS variable is expecting a character value

Valid: for WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments

MIXED <=> YES | NO | Y | N;

Details

You use the MIXED statement with WK*n* and XLS files if you have both numeric and character data in a column. Specifying YES allows both numeric and character data to be displayed as SAS character data. NO, the default, treats any data in a column that does not match the specified type as missing values.

You can change the default value to YES by setting the SS_MIXED environment variable. See "Setting Environment Variables for WK*n* Files" on page 182 for more information about setting and changing environment variables.

The MIXED statement is an editing statement and must follow the CREATE statement and any database descriptions when you create an access descriptor.

PATH= Statement

Specifies the path and filename of the file to be accessed

Requirement: This statement is required.

Valid: for DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, Excel 95 file formats under Windows operating environments

Applies to: access descriptor

PATH= *'path-and-filename<.PC-file-extension>'* |*<'>filename<'>* | *fileref* ;

Details

The PATH= statement indicates the path and name of the file you want to access. The length of the filename and its other conventions can vary with the operating system. See the host documentation for your operating environment for more information.

For compatibility, place the PATH= statement immediately after the CREATE statement and before any other database-description statements when creating access descriptors. See "CREATE Statement" on page 74 for more information.

You can specify the PATH= statement with one of the following arguments:

'*path-and-filename<.PC-file-extension>*'

specifies the fully qualified path and filename. You must enclose the entire path and filename in quotation marks, including the appropriate PC file extension, such as .dbf, .dif, .wk1, .wk3, wk4, .mdb, or .xls. If you omit the file extension, SAS/ACCESS software supplies it for you.

<'>*filename*<'>
specifies the name of a file. The file must be located in your current (default) directory. If no extension is specified, the SAS/ACCESS interface supplies it for you. If the filename includes characters that are invalid in SAS names, such as the dollar sign ($) or if the filename begins with a number, you must enclose the entire filename in quotation marks.

fileref
specifies a fileref that references the path and name of the file. (Assigning filerefs with the FILENAME statement is described in *Step-by-Step Programming with Base SAS Software*.)

QUIT Statement

Terminates the procedure

Valid: for DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments

Applies to: access descriptor or view descriptor

Alias: EXIT

QUIT;

Details

The QUIT statement terminates the ACCESS procedure without any further descriptor creation.

RENAME Statement

Modifies the SAS variable name

Valid: for DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments

Applies to: access descriptor or view descriptor

Interacts with: ASSIGN, RESET

RENAME <'>*column-identifier-1*<'><=>*SAS-variable-name-1*
<…<'>*column-identifier-n*<'><=>*SAS-variable-name-n*>;

Details

The RENAME statement enters or modifies the SAS variable name that is associated with a column in a PC file. Use the RENAME statement when creating an access descriptor or a view descriptor.

An editing statement, such as RENAME, must follow the CREATE statement and the database-description statements when you create a descriptor. See "CREATE Statement" on page 74 for more information about the order of statements.

Two factors affect the use of the RENAME statement: whether you specify the ASSIGN statement when you are creating an access descriptor, and the kind of descriptor you are creating.

- If you omit the ASSIGN statement or specify it with a NO value, the renamed SAS variable names that you specify in the access descriptor are retained throughout an ACCESS procedure execution. For example, if you rename the Customer column to CustNum when you create an access descriptor, that column continues to be named CustNum when you select it in a view descriptor unless a RESET statement or another RENAME statement is specified.

  When creating a view descriptor that is based on this access descriptor, you can specify the RESET statement or another RENAME statement to rename the variable again, but the new name applies only in that view. When you create other view descriptors, the SAS variable names are derived from the access descriptor variable names.
- If you specify the YES value in the ASSIGN statement, you can use the RENAME statement to change SAS variable names only while creating an access descriptor. As described earlier in the ASSIGN statement, SAS variable names and formats that are saved in an access descriptor are always used when creating view descriptors that are based on it.

The *column-identifier* argument can be either the PC file column name or the positional equivalent from the LIST statement, which is the number that represents the column's place in the descriptor. For example, to rename the SAS variable names that are associated with the seventh column and the nine-character FIRSTNAME column in a descriptor, submit the following statement:

```
rename 7 birthdy  'firstname'=fname;
```

The column name (or positional equivalent) is specified on the left side of the expression, with the SAS variable name on the right side. The equal sign (=) is optional. If the column name contains lowercase characters, special characters, or national characters, enclose the name in quotation marks. You can rename as many columns as you want in one RENAME statement.

When you are creating a view descriptor, the RENAME statement automatically selects the renamed column for the view. That is, if you rename the SAS variable associated with a column, you do not have to issue a SELECT statement for that column.

When you are creating an access descriptor, the RENAME statement also reselects columns that were previously dropped with the DROP statement.

RESET Statement

Resets PC file columns to their default settings

Valid: for DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments

Applies to: access descriptor or view descriptor

Interacts with: ASSIGN, DROP, FORMAT, RENAME, SELECT

Not allowed with: UPDATE

RESET ALL|<'>*column-identifier-1*<'><…<'>*column-identifier-n*<'>>;

Details

The RESET statement resets either the attributes of all the columns or the attributes of the specified columns to their default values. The RESET statement can be used when you create an access descriptor or a view descriptor, but it is not allowed when you are updating a descriptor. RESET has different effects on access and view descriptors, as described below.

If you use an editing statement, such as RESET, it must follow the CREATE statement and the database-description statements when you create a descriptor. See "CREATE Statement" on page 74 for more information about the order of statements.

The RESET statement can take one or more of the following arguments:

ALL
for access descriptors, resets all the PC file columns that have been defined to their default names and format settings and reselects any dropped columns.

For view descriptors, ALL resets all the columns that have been selected so that no columns are selected for the view; you can then use the SELECT statement to select new columns. See "SELECT Statement" on page 84 for more information about that statement.

column-identifier
can be either the PC file column name or the positional equivalent from the LIST statement, which is the number that represents the column's place in the access descriptor. For example, to reset the SAS variable name and format associated with the third column, submit the following statement:

```
reset 3;
```

If the column name contains lowercase characters, special characters, or national characters, enclose the name in quotation marks. You can reset as many columns as you want in one RESET statement, or use the ALL option to reset all the columns.

When creating an access descriptor, the *column-identifier* is reset to its default name and format settings. When creating a view descriptor, the specified column is no longer selected for the view.

Access Descriptors

When you create an access descriptor, the default setting for a SAS variable name is a blank. However, if you have previously entered or modified any of the SAS variable

names, the RESET statement resets the modified names to the default names that are generated by the ACCESS procedure. How the default SAS variable names are set depends on whether you included the ASSIGN statement. If you omitted ASSIGN or set it to NO, the default names are blank. If you set ASSIGN=YES, the default names are the first eight characters of each PC file column name.

The current SAS variable format is also reset to the default SAS format, which was determined from the column's data type. Any columns that were previously dropped, but that are specified in the RESET statement, become available; they can be selected in view descriptors that are based on this access descriptor.

View Descriptors

When you create a view descriptor, the RESET statement clears any columns that were included in the SELECT statement (that is, it de-selects the columns).

When creating the view descriptor, if you reset a SAS variable and then select it again within the same procedure execution, the SAS variable names and formats are reset to their default values, which are generated from the column names and data types. This applies only if you have omitted the ASSIGN statement or set the value to NO when you created the access descriptor on which the view descriptor is based. If you specified ASSIGN=YES when you created the access descriptor, the RESET statement has no effect on the view descriptor.

SELECT Statement

Selects PC file columns for the view descriptor

Requirement: This statement is required.

Valid: for DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments

Applies to: view descriptor

Interacts with: RESET

Not allowed with: UPDATE

SELECT ALL|<'>*column-identifier-1*<'><…<'>*column-identifier-n*<'>>;

Details

The SELECT statement specifies which PC file columns in the access descriptor to include in the view descriptor. This is a required statement and is used only when you create view descriptors. You cannot use the SELECT statement when you are updating a view descriptor.

If you use an editing statement, such as SELECT, it must follow the CREATE statement when you create a view descriptor. See "CREATE Statement" on page 74 for more information on the order of statements.

The SELECT statement can take one or more of the following arguments:

ALL
includes in the view descriptor all the columns that were defined in the access descriptor and that were not dropped.

column-identifier
can be either the column name or the positional equivalent from the LIST statement, which is the number that represents the column's place in the access descriptor on which the view is based. For example, to select the first three columns, submit the following statement:

```
select 1 2 3;
```

If the column name contains lowercase characters, special characters, or national characters, enclose the name in quotation marks. You can select as many columns as you want in one SELECT statement.

SELECT statements are cumulative within the same view creation. That is, if you submit the following two SELECT statements, columns 1, 5, and 6 are selected, not just columns 5 and 6:

```
select 1;
select 5 6;
```

To clear all your current selections when creating a view descriptor, use the RESET ALL statement; you can then use another SELECT statement to select new columns.

SUBSET Statement

Adds or modifies selection criteria for a view descriptor

Valid: for DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments

Applies to: view descriptor

SUBSET *selection-criteria*;

Details

You use the SUBSET statement to specify selection criteria when you create a view descriptor. This statement is optional; if you omit it, the view retrieves all the data (that is, all the rows) in the PC file.

An editing statement, such as SUBSET, must follow the CREATE statement when you create a view descriptor. See "CREATE Statement" on page 74 for more information about the order of statements.

TYPE Statement

Changes the expected data types of SAS variables

Valid: for DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments

TYPE <'>*column-identifier-1*<'><=> C | N<…*column-identifier-n* <=> C | N>;

Details

SAS data sets have two data types: character (C) and numeric (N). Spreadsheet files have the same two data types: character (for labels and formula strings) and numeric (for numbers and formulas). Changing the default data type of a SAS variable in a descriptor file also changes its associated default format in the loaded file.

If you omit the TYPE statement, the database field types are generated from the PC files data types. You can change as many database field types as you want in one TYPE statement.

This statement is not available for use with DBF files.

UNIQUE Statement

Generates SAS variable names based on PC file column names

Valid: for DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments

Applies to: view descriptor

Interacts with: ASSIGN

Not allowed with: UPDATE

Alias: UN

UNIQUE <=> YES|NO|Y|N ;

Details

The UNIQUE statement specifies whether the SAS/ACCESS interface generates unique SAS variable names for PC file columns for which SAS variable names have not been entered. You cannot use the UNIQUE statement when you are updating a view descriptor.

An editing statement, such as UNIQUE, must follow the CREATE statement when you create a view descriptor. See "CREATE Statement" on page 74 for more information about the order of statements. The UNIQUE statement is affected by whether you specified the ASSIGN statement when you created the access descriptor on which this view is based, as follows:

- If you specified the ASSIGN=YES statement, you cannot specify UNIQUE when creating a view descriptor. YES causes SAS to generate unique names, so UNIQUE is not necessary.
- If you omitted the ASSIGN statement or specified ASSIGN=NO, you must resolve any duplicate SAS variable names in the view descriptor. You can use UNIQUE to generate unique names automatically, or you can use the RENAME statement to resolve duplicate names yourself. See "RENAME Statement" on page 81 for information about that statement.

If duplicate SAS variable names exist in the access descriptor on which you are creating a view descriptor, you can specify UNIQUE to resolve the duplication. When you specify UNIQUE=YES, the SAS/ACCESS interface appends numbers to any duplicate SAS variable names, thus making each variable name unique. (See "CREATE Statement" on page 74 for an explanation of how to create descriptors.)

If you specify UNIQUE=NO, the SAS/ACCESS interface continues to allow duplicate SAS variable names to exist. You must resolve these duplicate names before saving (and thereby creating) the view descriptor.

Note: It is recommended that you use the UNIQUE statement. If you omit it and SAS encounters duplicate SAS variable names in a view descriptor, your job fails.

The equal (=) sign is optional in the UNIQUE statement. △

UPDATE Statement

Updates a SAS/ACCESS descriptor file

Valid: for DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments

Applies to: access descriptor or view descriptor

Not allowed with: ASSIGN, RESET, SELECT, UNIQUE

UPDATE *libref.descriptor-name*.ACCESS | VIEW ;

Details

Use the UPDATE statement to perform a quick, simple update of a descriptor. For example, if the PC database file for an existing access descriptor is relocated, you can use UPDATE with the PATH option to specify the new location.

Descriptors modified by UPDATE are not checked for errors. Where validation is crucial, use CREATE to overwrite a descriptor rather than UPDATE.

The descriptor is a name in three parts separated by periods (.):

libref
identifies the library container, which is a location either on the local system's disk or that the local system can directly access. The *libref* must have been previously created by a LIBNAME statement.

descriptor-name
is the descriptor you are updating. It must already exist in *libref*. (See "CREATE Statement" on page 74.)

ACCESS
indicates that you are updating an access descriptor while VIEW indicates you are updating a view descriptor.

Multiple UPDATE statements may appear in one ACCESS procedure block. If you use UPDATE to change an access descriptor, one or more UPDATE statements might be required for views that depend on the modified access descriptor.

You can use UPDATE and CREATE in the same PROC ACCESS block.

Updating Access Descriptors

The order of statements in an UPDATE block is as follows:

1. UPDATE must be the first statement after the PROC ACCESS statement with one exception: if the block includes both UPDATE and CREATE statements, either statement may be the first in the block.
2. Data source description statements are next. All are allowed.
3. Editing statements are next. These editing statements are not allowed: ASSIGN, LIST, RESET, SELECT, VIEW.

Since the UPDATE block does not validate the updated descriptor, the order of description and editing statements does not matter.

Updating View Descriptors

1. UPDATE must be the first statement after the PROC ACCESS statement with one exception: if the block includes both UPDATE and CREATE statements, either statement may be the first in the block.
2. Data source description statements are next. All are allowed.
3. Editing statements are next. These editing statements are not allowed: ASSIGN, DROP, RESET, SELECT, and UNIQUE.

Examples

The following example upates an existing access descriptor named AdLib.Product:

```
libname adlib 'c:\sasdata';

proc access dbms=wk4;
   update adlib.product.access;
   path=c:\lotus\specprod.wk4;
   rename productid prodid
          fibername fiber;
   format productid  4.
          weight     e16.9
          fibersize  e20.13
          width      e16.9;
run;
```

The following example updates the access descriptor Employ, located in **c:\sasdata**, for the spreadsheet named **c:\excel\employ.xls** and updates the view descriptor for Employ named Emp1204, located in **c:\sasviews**:

```
libname adlib 'c:\sasdata';
libname vlib 'c:\sasviews';

proc access dbms=xls;
   update adlib.employ.access;
   path='c:\excel\employ.xls';
   list all;

   update vlib.emp1204.view;
   format empid 6.
          salary dollar12.2
          jobcode 5.
```

```
          hiredate datetime9.
          birthdate datetime9.;
   subset where jobcode=1204;
run;
```

The following example updates a second view descriptor that is based on Employ named BDays. It is also located in **c:\sasviews**. When you update a view, it is not necessary to specify the access descriptor (using ACCDESC=) in the PROC ACCESS statement. Note that FORMAT can be used because the access descriptor Employ was created with ASSIGN=NO.

```
libname vlib 'c:\sasviews';

proc access dbms=xls;
   update vlib.bdays.view;
   format empid 6.
          birthdate datetime7.;
run;
```

CHAPTER 7

The DBLOAD Procedure for PC Files

Overview of the DBLOAD Procedure for PC Files

The DBLOAD procedure for PC files is only available under Windows operating environments. You can use the DBLOAD procedure with DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats. See "Methods for Accessing PC Files Data" on page 3 for alternate methods for accessing data in PC file formats under Windows, UNIX, OS/390, and OpenVMS operating environments.

This section provides general reference information for the DBLOAD procedure. It presents the PROC DBLOAD options and statements that are common to all formats. File format specific information for the SAS/ACCESS interface to your PC file is included in separate sections.

Refer to *SAS Language Reference: Dictionary* and to the SAS documentation for your operating environment for more information about SAS data sets and SAS data libraries and their naming conventions or for help with the terminology used in this procedure description.

The DBLOAD procedure loads data to and creates PC files. This data can be from any of the following: a SAS data file, a PROC SQL view, a DATA step view, or a view descriptor from any SAS/ACCESS interface product. The DBLOAD procedure associates each SAS variable with a PC file column and assigns a default name and data type to each column. You can use the default information or change it as necessary. When you are finished customizing the columns, the procedure creates the PC file and loads it with the input data.

DBLOAD Procedure Naming Conventions

When you use the DBLOAD procedure to load a SAS data set into a PC file, the SAS variable names cannot exceed 8 characters. This restriction is applied in order to have compatibility with Version 6 naming conventions.

DBLOAD Procedure Syntax

When you use the DBLOAD procedure, you use different statements depending on your task and your PC file. Not all statements are available with all PC file formats, and additional statements might be used with your PC file. The general syntax for this procedure is presented here; see the format-specific sections for file format specific information.

PROC DBLOAD <DBMS=*pc-file*>
<DATA=<*libref.*>*SAS-data-set*>;

Database-Description Statement
PATH='*path-and-filename*<*.PC-file-extension*>'|
<'>*filename*<'>|*fileref* ;

Editing Statements
ACCDESC=<*libref.*>*access-descriptor*;
DELETE *variable-identifier-1*<*…variable-identifier-n*;>
ERRLIMIT=*error-limit*;
LABEL;
LIMIT=*load-limit*;
LIST <ALL|COLUMNS|FIELDS|*variable-identifier*>;
QUIT;
RENAME *variable-identifier-1*=<'>*column-name-1*<'>
<*…variable-identifier-n*<'>*column-name-n*<'>>;
RESET ALL|*variable-identifier-1* <*…variable-identifier-n* >;
WHERE *SAS-where-expression*;

Creating and Loading Statement
LOAD;

RUN;

PROC DBLOAD Statement

Requirement: This statement is required.

Valid: for DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments

PROC DBLOAD <DBMS=*pc-file*>
<DATA=<*libref.*>*SAS-data-set*>;

Arguments

DBMS=*pc-file*
Specifies which PC file format that you want to access. Specify DBMS=DBF for DBF files, DBMS=DIF for DIF files, DBMS=WK1 | WK3 | WK4 for WK*n* files, DBMS=MDB for MDB files, or DBMS=XLS for XLS files. The DBMS= option is required.

DATA=<*libref.*>*SAS-data-set*
specifies the input data set. The input data can be retrieved from a SAS data file, a PROC SQL view, a DATA step view, or a SAS/ACCESS view descriptor. If the data set is permanent, you must use its two-level name, *libref.SAS-data-set*. If you omit the DATA= option, the default is the last SAS data set that was created.

ACCDESC= Statement

Creates an access descriptor based on the new file

Valid: for DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments

Aliases: ACCESS= and AD=

ACCDESC=<*libref.*>*access-descriptor*;

Details

The ACCDESC= statement creates an access descriptor based on the PC file that you are creating and loading. After the new PC file is created and loaded, the access descriptor is automatically created. You must specify an access descriptor that does not already exist.

An editing statement, such as ACCDESC=, must be specified after the database-description statements when you create and load a file. See "LOAD Statement" on page 96 for more information.

DELETE Statement

Prevents variables from being loaded into the new PC file

Valid: for DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments

Interacts with: RENAME, RESET

DELETE *variable-identifier-1* <*…variable-identifier-n*>;

Details

The DELETE statement drops the specified SAS variables from the PC file being created. The *variable-identifier* argument can be either the SAS variable name or the positional equivalent from the LIST statement. The positional equivalent is the number that represents the variable's place in the data set. For example, if you want to drop the third variable, submit the following statement:

```
delete 3;
```

You can drop as many variables as you want in one DELETE statement. If you drop more than one variable, separate the identifiers with spaces, not commas.

Even if you drop a variable from the list of variables, the positional equivalents of the variables do not change. For example, if you drop the second variable, the third variable is still referenced by the number 3, not 2.

An editing statement, such as DELETE, must be specified after the database-description statements when you create and load a file. See "LOAD Statement" on page 96 for more information.

ERRLIMIT= Statement

Stops loading data after a specified number of errors

Valid: for DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments

Default: 100

ERRLIMIT=*error-limit*;

Details

The ERRLIMIT= statement stops loading observations after the specified number of errors has occurred while inserting rows into the file.

The *error-limit* argument must be a nonnegative integer. Specify ERRLIMIT=0 to allow an unlimited number of errors to occur.

An editing statement, such as ERRLIMIT=, must be specified after the database-description statements when you create and load a file. See "LOAD Statement" on page 96 for more information.

LABEL Statement

Causes column names to default to SAS labels

Valid: for DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments

Interacts with: RENAME, RESET

LABEL;

Details

The LABEL statement causes the column names to default to the SAS variable labels when the new table is created. If a SAS variable has no label, the variable name is used. If the label is too long to be a valid column name, the label is truncated.

For the LABEL statement to take effect, the RESET statement must be used *after* the LABEL statement.

An editing statement, such as LABEL, must be specified after the database-description statements when you create and load PC files. See "LOAD Statement" on page 96 for more information.

LIMIT= Statement

Limits the number of observations loaded

Valid: for DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments

Default: 5000

LIMIT=*load-limit*;

Details

The LIMIT= statement places a limit on the number of observations that can be loaded into a new file. The maximum number for the limit statement varies with each PC file. The *load-limit* argument must be a nonnegative integer. To load all the observations from your input data set, specify LIMIT=0.

If you omit the LIMIT= statement, a maximum of 5,000 observations are inserted.

LIST Statement

Lists information about the variables to be loaded

Valid: for DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments

Default: ALL

LIST <ALL | COLUMNS | FIELDS | *variable-identifier*>;

Details

The LIST statement lists information about all or some of the SAS variables to be loaded into the new file. By default, the list is sent to the SAS log.

The LIST statement can take one or more of the following arguments:

ALL
lists information about all the variables in the input SAS data set, whether or not those variables are selected for the load.

COLUMNS
lists information about only the input SAS variables that are selected for the load. This argument does not apply to DBF files.

FIELDS
lists information about only the input SAS variables that are selected for the load.

variable-identifier
lists information about only the specified variable. The *variable-identifier* argument can be either the SAS variable name or the positional equivalent. The positional equivalent is the number that represents the variable's position in the data set. For example, if you want to list information for the column associated with the third SAS variable, submit the following statement:

```
list 3;
```

You can specify LIST as many times as you want while creating a file; specify LIST before the LOAD statement to see the entire table. LIST must be specified after the database-description statements. See "LOAD Statement" on page 96 for more information.

LOAD Statement

Creates and loads the new PC file

Requirement: This statement is required.

Valid: for DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments

LOAD;

Details

The LOAD statement causes the interface view engine to create a file and to transfer data to it from the input SAS data set after the DBLOAD procedure is submitted for processing. This statement is required to create and load a new file.

When you create and load a file, you must place statements or groups of statements in a certain order after the PROC DBLOAD statement and its options, as follows:

1. Database-description statements: PATH= and your PC file specific statements.
2. Editing statements: ACCDESC=, DELETE, ERRLIMIT, LABEL, LIMIT=, LIST, RENAME, RESET, and WHERE. The order within this group usually does not matter. See the individual statements for more information. QUIT is also an editing statement but using it immediately terminates PROC DBLOAD.
3. Creating and loading statement: LOAD must appear last before RUN in order to create and load the new table.
4. RUN statement: this statement is used to process the DBLOAD procedure.

PATH= Statement

Indicates the name and path of the PC file to be created and loaded

Requirement: This statement is required.

Valid: for DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments

PATH='*path-and-filename <.PC-file-extension>*'|
<'>*filename*<'>|*fileref* ;

Details

The PATH= statement indicates the path and name of the PC file you want to create and load. The length of the filename can vary with the operating environment. See the SAS documentation for your operating environment for any restrictions.

The PATH= statement can take one of the following arguments:

'*path-and-filename<.PC-file-extension>*'
specifies the fully qualified path and filename. You must enclose the entire path and filename in quotation marks, including the appropriate PC file extension, such as .dbf, .dif, .wk*n*, .mdb, or .xls. If you omit the file extension, SAS/ACCESS supplies it for you.

<'>*filename* <'>
specifies the name of a file. The file must be located in your current (default) directory. If no extension is specified, the SAS/ACCESS interface supplies it for you. If the filename includes characters that are invalid in SAS names, such as

the dollar sign ($) or if the filename begins with a number, you must enclose the entire filename in quotation marks.

fileref
specifies a fileref that references the path and name of the file. (Assigning filerefs with the FILENAME statement is described in *Step-by-Step Programming with Base SAS Software*.)

A file with the same name must not already exist. If one does exist, it is not overwritten. An error message is written to the SAS log, and the PC file that is specified in this statement is not loaded.

QUIT Statement

Exits the DBLOAD procedure

Valid: for DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments

Alias: EXIT

QUIT;

Details

The QUIT statement exits the procedure without further processing.

RENAME Statement

Renames PC file columns

Valid: for DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments

Interacts with: DELETE, LABEL, RESET

Alias: COLUMN

RENAME *variable-identifier-1=<'>column-name-1<'>*
<…variable-identifier-n =<'>column-name-n<'>>;

Details

The RENAME statement changes the names of the PC file's columns that are associated with the listed SAS variables. If you omit the RENAME statement, all the

column names default to the corresponding SAS variable names (unless the LABEL statement is specified).

The *variable-identifier* argument can be either the SAS variable name or the positional equivalent from the LIST statement. The positional equivalent is the number that represents the variable's place in the data set. For example, if you want to rename the column associated with the third SAS variable, submit the following statement:

```
rename 3='employname';
```

The *column-name* argument must be a valid PC file column name. If the column name includes lowercase characters, special characters, or national characters, you must enclose the column name in quotation marks.

The RENAME statement enables you to include variables that you have previously deleted. For example, suppose you submit the following statements:

```
delete 3;
rename 3='empname';
```

The DELETE statement first drops the third variable. Then the RENAME statement includes the third variable and assigns the name EMPNAME and the default column type to it.

You can rename as many variables as you want in one RENAME statement. The RENAME statement overrides the LABEL statement for columns that are renamed.

An editing statement, such as RENAME, must be specified after the database-description statements when you create and load a PC file. See "LOAD Statement" on page 96 for more information.

RESET Statement

Resets column names and data types to their default values

Valid: for DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments

Interacts with: DELETE, LABEL, RENAME

RESET ALL|*variable-identifier-1*<*…variable-identifier-n*>;

Details

The RESET statement resets the columns that are associated with the listed SAS variables to the default column name, column data type, and ability to accept null values. If you specify ALL, all columns are reset to their default values, and any deleted columns are restored with their default values.

The *variable-identifier* argument can be either the SAS variable name or the positional equivalent from the LIST statement. The positional equivalent is the number that represents the variable's place in the data set. For example, if you want to reset the column associated with the third SAS variable, submit the following statement:

```
reset 3;
```

You can reset as many columns as you want in one RESET statement.

You must use the RESET statement after the LABEL statement for the LABEL statement to take effect.

An editing statement, such as RESET, must be specified after the database-description statements when you create and load a PC file. See "LOAD Statement" on page 96 for more information.

WHERE Statement

Loads a subset of data into the new PC file

Valid: for DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments

WHERE *SAS-where-expression*;

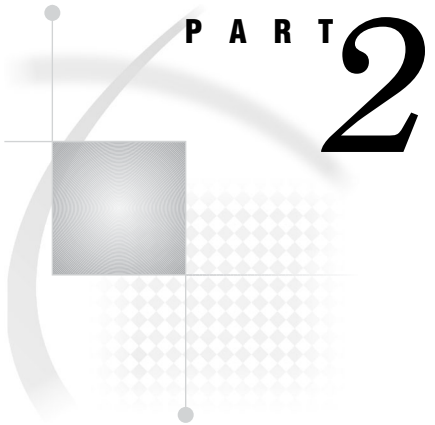
Details

The WHERE statement loads a subset of observations into the new PC file. The *SAS-where-expression* must be a valid SAS WHERE statement that uses SAS variable names (not column names) as defined in the input data set. The following example loads only the observations in which the SAS variable Country has the value **Brazil**.

```
where country='Brazil';
```

For more information about the syntax of the SAS WHERE statement, see *SAS Language Reference: Dictionary*.

An editing statement, such as WHERE, must be specified after the database-description statements when you create and load a PC File. See "LOAD Statement" on page 96 for more information.



PART 2

Accessing PC Files on UNIX

CHAPTER

8

Overview of the SAS/ACCESS Interface to PC Files on UNIX

Introduction to the SAS/ACCESS Interface to PC Files on UNIX

The SAS/ACCESS interface to PC files on UNIX has been significantly enhanced in SAS 9.1.

It is now possible to access a wider range of PC file formats from UNIX, provided that these files reside on the PC. This is made possible by the PC files server, which resides on a network PC. You can use the LIBNAME statement, the IMPORT and EXPORT procedures, and the Pass-Through Facility to access these new PC file formats.

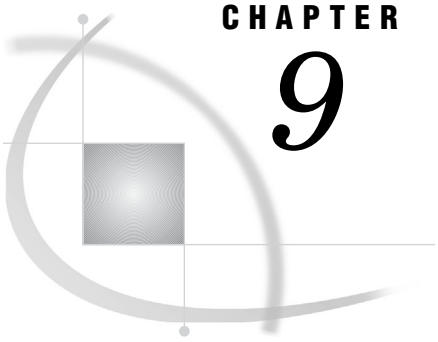
When the PC files reside on UNIX, however, SAS/ACCESS to PC files instead behaves as it did in SAS Version 8, working with a limited range of PC file formats via the Import/Export wizard and procedures. In SAS 9.1, locally-residing JMP files can also be accessed in this way.

The following table summarizes the capabilities of the SAS/ACCESS interface to PC files on UNIX for SAS 9.1:

Table 8.1 Capabilities of SAS/ACCESS Interface to PC Files on UNIX

Method	Location of Data	Supported PC File Formats
LIBNAME statement	UNIX	None
	PC	Microsoft Access (version 97, 2000, and 2002)
		Microsoft Excel (version 5, 95, 97, 2000, and 2002)
		ODBC data source
Import/Export wizard	UNIX	dBASE DBF (III, III PLUS, IV, and 5.0)
		Delimited (tab, space, comma, and other delimitor)
		JMP
	PC	None

Method	Location of Data	Supported PC File Formats
IMPORT and EXPORT procedures	UNIX	dBASE DBF (III, III PLUS, IV, and 5.0)
		Delimited (tab, space, comma, and other delimitors)
		JMP
	PC	Microsoft Access (version 97, 2000, and 2002)
		Microsoft Excel (version 5, 95, 97, 2000, and 2002)
		JMP
Pass-Through Facility	UNIX	None
	PC	Microsoft Access (version 97, 2000, and 2002)
		Microsoft Excel (version 5, 95, 97, 2000, and 2002)
		ODBC data source
DBF procedure	UNIX	dBASE DBF (III, III PLUS, IV, and 5.0)
	PC	None
DIF procedure	UNIX	DIF files
	PC	None



CHAPTER 9

The PC Files Server

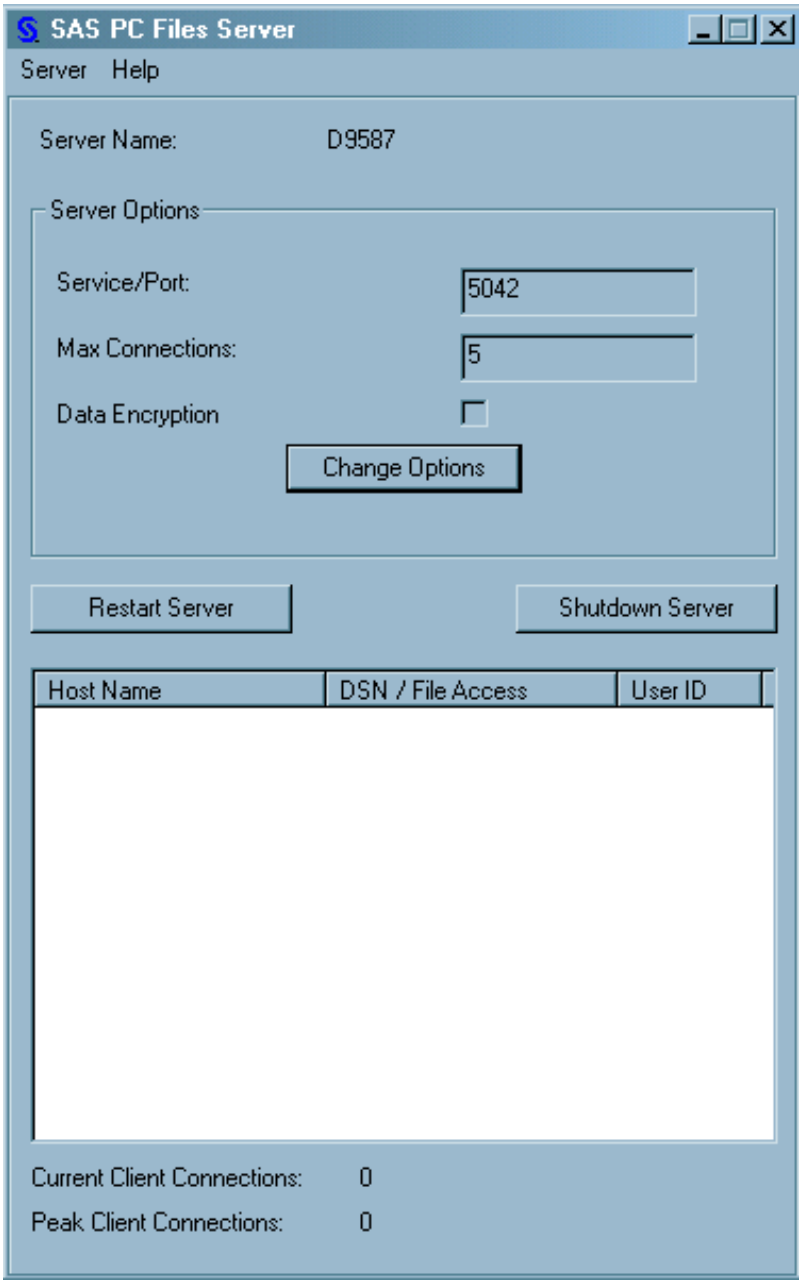
Overview of the PC Files Server

The PC files server, when combined with an installation of SAS/ACCESS for PC Files on UNIX, enables you to access PC data from UNIX. The server is a multi-threaded server with user controls that are defined in the following sections.

Starting the PC Files Server

After you install the server components, consisting of the server, pcfserver.exe, and several utility files, you are ready to start the server.

Open the **pcfserver.exe** file to open the SAS PC Files Server window. The SAS PC Files Server window opens:



The top of the SAS PC Files Server window shows the server name, which identifies the name of the server that started the PC files server application. It is display only and cannot be modified.

The **Server Options** section contains connection information. You can change this information as needed. See “Configuring the PC Files Server” on page 107.

The bottom of the window displays the active connections to the server. It contains information about who is connected and what they are looking for via the **Host Name**, **User ID**, and **DSN/File Access** fields respectively. If a single user opens multiple connections then the most current **DSN/File Access** information is shown.

Configuring the PC Files Server

Setting the Service Name or Port Number

The **`Service/Port`** field in the SAS PC Files Server window specifies the port number or service name that the server will use to check for UNIX connection requests from SAS/ACCESS. You should configure the service or port as a unique component on your internal network, especially if you are going to run multiple PC files servers on multiple PCs. The service or port that you specify is saved in the Windows registry. It is used with subsequent invocations of the server.

Setting Maximum Connections

The **`Max Connections`** field in the SAS PC Files Server window specifies the number of connections that will be supported by the server. The default is 10. Configure the number of connections based on the load you expect on the PC from your UNIX users.

When calculating the potential connections, estimate one conncetion for every concurrent LIBNAME statement and two for every concurrent Import/Export execution. For example, if you have ten UNIX users connecting to the PC server concurrently, using only PROC IMPORT/EXPORT, then the **`Max Connections`** field should be set to 20 (2 * number of users). If the users are utilizing a mix of LIBNAME and procedure statements, **`Max Connections`** should still be set to 20, due to the reusability of client connections from an individual user (that is, multiple LIBNAME statements can be executed on one client connection).

The number of connections that you set is saved in the Windows registry. The value is used with subsequent invocations of the server.

Setting Data Encryption

To enable data encryption between the SAS/ACCESS to PC Files on UNIX client and the SAS PC files server, select the **`Data Encryption`** box on the SAS PC Files Server window. When this option is selected, plain text is not transmitted across the network.

Note: When this item is checked you might see a decrease in performance. △

Constraints

There are several administrative constraints to be aware of:

- Only one SAS PC files server can be active on a given PC.
- Service names and port numbers need to be unique on each server.
- If you make any changes to the parameters, you must restart the server in order for them to take effect. During the restart process, any users currently connected to the server will be disconnected, which can result in loss of data.
- If you stop or restart the server, all users sessions will be closed, which can result in a loss of data.

Shared Information

After you have configured the PC server, you need to share the configuration information with the UNIX users who will get data from the PC. They need to be supplied with the following information:

- server name
- service name or port number
- path to any files or ODBC data sources to which they can have access.

CHAPTER
10

The LIBNAME Statement for PC Files on UNIX

Overview of the LIBNAME Statement for PC Files on UNIX

For PC files, the SAS/ACCESS LIBNAME statement extends the SAS global LIBNAME statement to support assigning a libref to Microsoft Excel, Microsoft Access, and ODBC data sources. This enables you to reference spreadsheets, databases, and ODBC sources directly in a DATA step or SAS procedure, and to read from and write to a Microsoft Access, Microsoft Excel, or ODBC table as though it were a SAS data set. This section specifies the syntax for this statement and provides examples.

Sorting PC Files Data

Because librefs that refer to PC data refer to database and workbook objects such as tables, they are stored in a format that differs from that of normal SAS data sets. This is important to remember when you access and work with PC files data.

For example, you can sort the observations in a normal SAS data set and store the output to another data set. However, in a Microsoft Access database, sorting data has no effect on how it is stored. Because your data might not be sorted in the external file, you must sort the data at the time of query.

Furthermore, when you sort PC files data, the results might vary depending on whether the external spreadsheet or database places data with NULL values (which are translated in SAS to missing values) at the beginning or the end of the result set.

Using SAS Functions with PC Files Data

When you use librefs that refer to PC files data with SAS functions, some functions might return a value different from what is returned when you use the functions with normal SAS data sets. For example, the PATHNAME function normally returns the pathname for the assigned libref. However, when the libref refers to PC files data, the function might return a Microsoft Excel filename assigned for the libref.

Usage of some functions might also vary. For example, the LIBNAME function can accept an optional *SAS-data-library* argument. When you use the LIBNAME function to assign or deassign a libref that refers to PC files data, however, you omit this

argument. For full details about how to use SAS functions, see the *SAS Language Reference: Dictionary*.

LIBNAME Statement Syntax for PC Files on UNIX

Associates a SAS libref with a workbook, database, or ODBC data source

Valid in: anywhere

Syntax

❶ **LIBNAME** *libref* **pcfiles**
<*connection-options*>
<*libname-options*>;

❷ **LIBNAME** *libref* CLEAR | _ALL_ CLEAR;

❸ **LIBNAME** *libref* LIST | _ALL_ LIST;

Arguments

libref
is any SAS name that serves as an alias to associate SAS with a spreadsheet, data source, or database. Like the global SAS LIBNAME statement, the SAS/ACCESS LIBNAME statement creates shortcuts or nicknames for data storage locations. While a SAS libref is an alias for a virtual or physical directory, a SAS/ACCESS libref is an alias for the spreadsheet, data source, or database where your data is stored.

pcfiles
is the SAS/ACCESS engine name for the interface to PC files on UNIX.

CLEAR
deassigns one or more currently assigned librefs.

Specify *libref* to deassign a single libref. Specify _ALL_ to deassign all currently assigned librefs.

ALL
specifies that the CLEAR or LIST argument applies to all currently assigned librefs.

LIST
writes the attributes of one or more SAS/ACCESS libraries or SAS data libraries to the SAS log.

Specify *libref* to list the attributes of a single SAS/ACCESS library or SAS data library. Specify _ALL_ to list the attributes of all libraries that have librefs in your current session.

connection-options
provide connection information to SAS/ACCESS to connect to your PC files. If the connection options contain characters that are not allowed in SAS names, enclose the values of the arguments in quotation marks. In some instances, if you specify the appropriate system options or environment variables for your data source, you can omit the connection options.

See "Connection Options" on page 111 for detailed information about connection options.

libname-options
define how SAS interacts with your data source, and provide enhanced control of the way that SAS processes data source objects. For example, some LIBNAME options can improve performance. For many tasks, you do not need to specify any of these advanced options.

See "LIBNAME Options for PC Files on Windows" on page 11 for detailed information about LIBNAME options.

Connection Options

SAS/ACCESS provides many ways to connect to your PC files.
The following options are used when you are connecting to the PC files server.

DSN="*data-source-name*"
specifies the ODBC data source name that will be used to access the PC data via an ODBC driver on the PC.

Note: This ODBC data source must be defined on the PC where the PC files server is currently running. △

CONNECT_STRING= "*connection-string*"
specifies connection options for your data source or database. Separate multiple options with a semicolon. This is an advanced connection method that should only be used when you know the exact syntax of all the connection options that the ODBC driver requires for a successful connection.

PASSWORD="*user-password*"
specifies a password for the user account, if required by the data source. Passwords are case-sensitive.
Aliases: PWD, PW, PASS, PASSWORD

PATH="*pathname*"
specifies the location of the data file. This is the full path and filename for your Microsoft Access database file or Microsoft Excel workbook file. Always enter file extension .mdb for Microsoft Access and .xls for Excel.

PORT="*port-number*"
specifies the port or service name that the SAS PC files server is listening on on the PC. This port or service name is displayed on the PC files server control panel screen when it is started on the PC. This is a required field when connecting to the PC files server for data.
Aliases: SERVICE, SERVICE_NAME
Default 8621

SERVER="*pc-server-hostname*"
specifies the computer name of the PC on which you started the PC files server. This name is required by UNIX users to connect to this server machine and is reflected on the server control panel. This is a required field when you are connecting to the PC files server for data.
This hostname can be specified as a simple computer name (wxp320), a fully qualified network name (**`wxp320.domain.com`**), or an IP address.

USER="*user-ID*"
specifies a user account name, if one is required to connect to the data source. For Microsoft Access, if you have user-level security set in your .mdb file, you need to use this option and the PASSWORD= option to be able to access your file.

Alias: UID

The following options are used only for Microsoft Access.

DBPASSWORD="*database-file-password*"
enables you to access your file if you have database-level security set in your .mdb file. A database password is case-sensitive and can be defined instead of user-level security.
Aliases: DBPWD, DBPW, PASS, PASSWORD

DBSYSFILE="*workgroup-information-file*"
contains information about the users in a workgroup based on information that you define for you Microsoft Access database. Any user and group accounts or passwords you create are saved in the workgroup information file.
Alias: SYSTEMDB

The following option is used only for Microsoft Excel.

VERSION=2002 | 2000 | 97
sets the version of Microsoft Excel. The default value is 97.
Alias: VER

Note: You do not need to specify this option if you do not know the version of your Microsoft Excel file. However, if you want to create a new Microsoft Excel file, you can use this option to specify the version you want to create. △

2002	sets the version of Microsoft Excel to 2002.
2000	sets the version of Microsoft Excel to 2000.
97	sets the version of Microsoft Excel to 97.

The following option is used for Microsoft Excel or Microsoft Access.

TYPE="*file-type*"
specifies the file type of the file in the PATH= statement. Valid values of TYPE= are EXCEL or ACCESS. Use TYPE= if the file identified in the PATH= statement does not have the .xls or .mdb extension at the end of the name.

The following example assigns the libref db for an Excel file:

```
libname db pcfiles server=D2323 port=8621 path='c:\demo.xls';
```

Details

❶ Using Data from a PC File

SAS/ACCESS for PC Files on UNIX enables you to directly access PC data from UNIX. You can read from and write to a variety of PC file data residing on a PC, including Microsoft Excel, Microsoft Access, and any other ODBC data source.

The engine utilizes ODBC to support assigning a libref to Microsoft Excel and Microsoft Access files residing on a PC from UNIX. This enables you to reference spreadsheets, databases, and other ODBC data sources directly in a DATA step or SAS procedure, and to read from and write to a Microsoft Access or Excel object as though it were a SAS data set.

❷ Disassociating a Libref from a SAS Data Library

To disassociate or clear a libref, use a LIBNAME statement, specifying the libref (for example, mypclib) and the CLEAR option as follows:

```
libname mypclib CLEAR;
```

You can clear a single specified libref or all current librefs.

SAS/ACCESS disconnects from the data source and closes any free threads or resources that are associated with that libref's connection.

❸ Writing SAS Data Library Attributes to the SAS Log Use a LIBNAME statement to write the attributes of one or more SAS/ACCESS libraries or SAS data libraries to the SAS log. Specify *libref* to list the attributes of a single SAS/ACCESS library or SAS data library, as follows:

```
libname mypclib LIST;
```

Specify _ALL_ to list the attributes of all libraries that have librefs in your current session, as follows:

```
libname _ALL_ LIST;
```

Examples

Assigning a Libref to a Microsoft Access Database The following statement creates the libref mymdb:

```
libname mymdb pcfiles server=D2323 port=8621 path="c:\demo.mdb";
```

The demo.mdb database contains a number of objects, including several tables, such as Staff. After you assign the libref, you can reference the Microsoft Access table like a SAS data set and use it as a data source in any DATA step or SAS procedure. In the following PROC SQL statement, mymdb.staff is the two-level SAS name for the Staff table in the Microsoft Access database Demo.

```
proc sql;
  select idnum, lname
    from mymdb.staff
    where state='NY'
    order by lname;
quit;
```

You can use the Microsoft Access data to create a SAS data set:

```
data newds;
   set mymdb.staff(keep=idnum lname fname);
run;
```

You can also use the libref and data set with any other SAS procedure. This statement prints the information in the staff table:

```
proc print data=mymdb.staff;
run;
```

This statement lists the database objects in the mymdb library:

```
proc datasets library=mymdb;
quit;
```

Assigning a Libref to a Microsoft Excel Workbook The following statement creates a libref, myxls, for an Excel workbook:

```
libname myxls pcfiles server=D2323 port=8621 path="c:\demo.xls";
```

The demo.xls workbook contains a number of sheets, such as sheet1. After you assign the libref, you can reference the Microsoft Excel spreadsheet like a SAS data set and use it as a data source in any DATA step or SAS procedure. In the following example a SAS data set is created from an Excel sheet:

```
data a;
   set myxls.'sheet1$'n;
run;
```

Note: When using a LIBNAME statement with Excel, you must refer to Excel sheets as n-literals because of the "$" character. If you are referencing a named range in an Excel spreadsheet, it is not necessary to refer to it as a n-literal.

The following example illustrates how to reference a named range called pageone in an Excel workbook:

```
libname myxls pcfiles server=d2323 port=8621 path="c:\demo.xls";

data a;
   set myxls.pageone;
run;
```

You can also create an Excel file and use a SAS data set to populate a sheet in that file. A named range is also created for that sheet.

```
libname myxls pcfiles server=D2323 port=8621 path="c:\demo.xls";

data myxls.sheet1;
   set sashelp.air;
run;
```

△

You can also use the libref with any other SAS procedure. For example, you can use PROC PRINT on a sheet in an Excel file:

```
libname myxls pcfiles server=d2323 port=8621 path="c:\test.xls";
proc print data=myxls.'sheet1$'n;
run;
```

Assigning a Libref to a Microsoft SQL Server Database The following statement creates a libref, mysqlsrv, to a Microsoft SQL Server database via ODBC, using the server on the PC:

```
libname mysqlsrv pcfiles server=D2323 port=8621 dsn=MQIS user=scott
                         pwd=tiger schema=dbo;
```

Using the mysqlsrv libref, a SAS data set called sqltest is created from the crime table in the Microsoft SQL Server database:

```
data work.sqltest;
   set mysqlsrv.crime;
run;
```

or

```
proc sql;
   create table work.sqltest as select * from mysqlsrv.crime;
quit;
```

Using the mysqlsrv libref, a SQL Server table is created called newtable from the SAS data set sqltest:

```
data mysqlsrv.newtable;
   set sqltest;
run;
```

Assigning a Libref to an Oracle Database The following statement creates a libref, ora, to an Oracle database table via ODBC, using the PC files server on the PC:

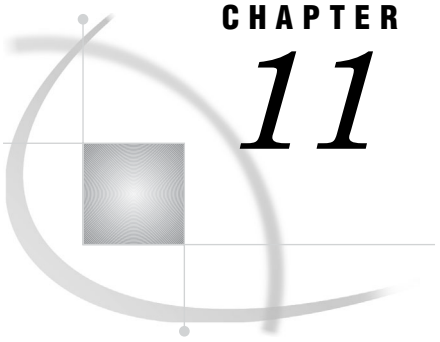
```
libname ora pcfiles server=D2323 port=8621 dsn=ORA9MS
                    user=scott pwd=tiger preserve_tab_names=yes;
```

Using the ora libref, an Oracle table, oratab, is created from a SAS data set sashelp.class:

```
data ora.oratab;
   set sashelp.class;
run;
```

Using the ora libref, a SAS data set, sastab, is created from the Oracle table emp:

```
data sastab;
   set ora.emp;
run;
```



CHAPTER

11

The Import/Export Wizard and Procedures on UNIX

Import/Export Overview for PC Files on UNIX

The Import/Export wizard, PROC IMPORT, and PROC EXPORT enable the transfer of data between SAS and different PC file formats. The Import/Export wizard is a point-and-click interface, while the IMPORT and EXPORT procedures are code-based. At present, the IMPORT and EXPORT procedures handle a broader range of files than the Import/Export wizard, which will only work with certain PC file formats that reside on UNIX.

The IMPORT and EXPORT procedures work within the same limited range of file formats if they reside locally on UNIX, but will also work with a wider range of PC file formats which reside on the PC. For comprehensive documentation about these features, see *Base SAS Procedures Guide*.

The following table summarizes import and export capabilities:

Table 11.1 Import/Export Capabilities on UNIX

Method	Location of Data	Supported PC File Formats
Import/Export wizard	UNIX	dbase DBF (Versions III, III PLUS, IV, and 5.0)
		delimited (tab, space, comma, and other delimitors)
		JMP
	PC	None
IMPORT/EXPORT procedures	UNIX	dBASE DBF (III, III PLUS, IV, and 5.0)
		delimited (tab, space, and comma)
		JMP
	PC	Microsoft Acess Client (versions 97, 2000, and 2002)
		Microsoft Excel (versions 5, 95, 97, 2000, and 2002)
		JMP

Note: The Import/Export wizard and IMPORT and EXPORT procedures are a part of Base SAS software. If you do not have a license for SAS/ACCESS to PC files, however, you can only access delimited files with these features. △

Import/Export Wizard on UNIX

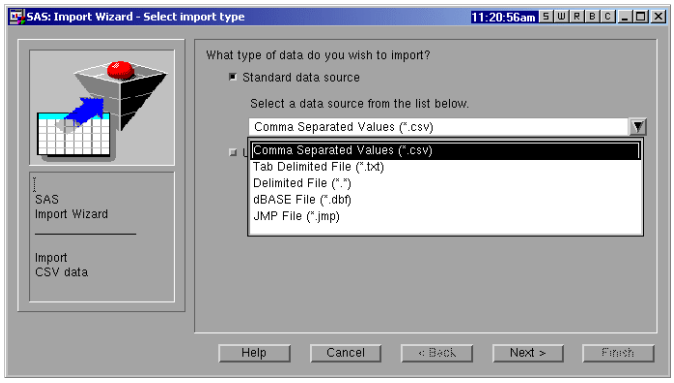
The Import/Export wizard is a point-and-click interface that guides you through the importing and exporting of certain PC file formats that reside on UNIX. If you are working in a UNIX environment, it will NOT work with PC files that reside on the PC.

To invoke the Import/Export wizard, from the SAS windowing environment, select **`File`** and then either **`Import Data`** or **`Export Data`**. Detailed information about using the wizard is available from the Help button.

The Import wizard enables you to read data from an external data source on UNIX and write it to a SAS data set. External data sources can include DBF files, JMP files, or delimited files, which are files containing columns of data values that are separated by a delimiter such as a blank or a comma. Complete the following steps to use the Import wizard on UNIX.

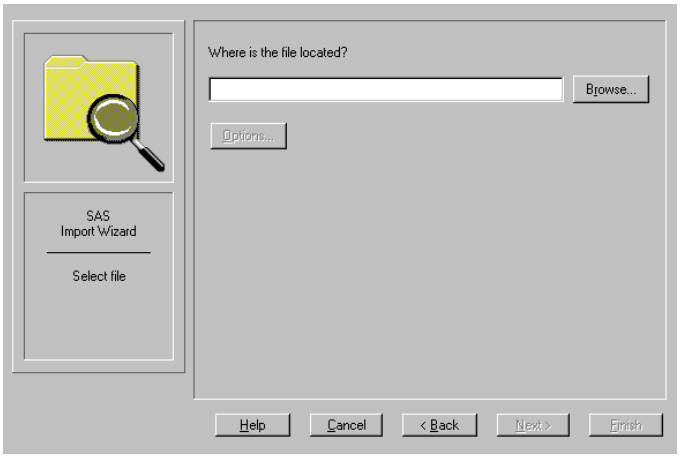
1 Select the type of files you are importing.

Display 11.1 Import Wizard: Select Import Type



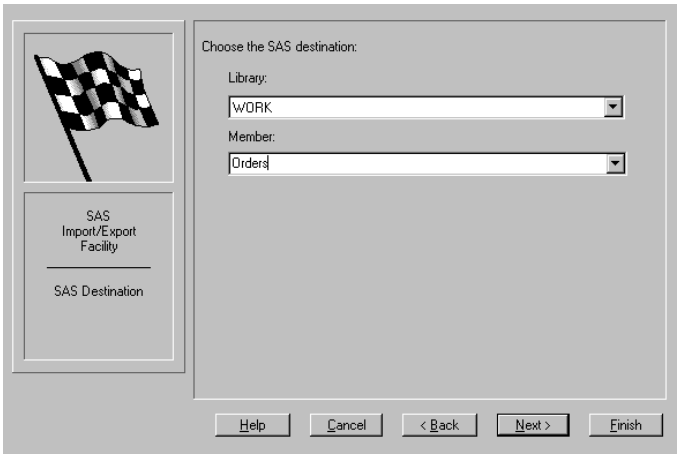
2 Locate the input file.

Display 11.2 Import Wizard: Select File



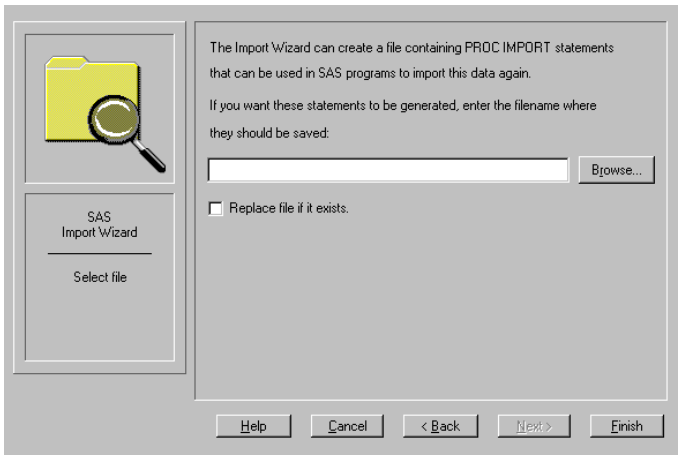
3 Select a location in which to store the imported file.

Display 11.3 Import Wizard: Imported File Location



4 Save the generated PROC IMPORT code. (Optional)

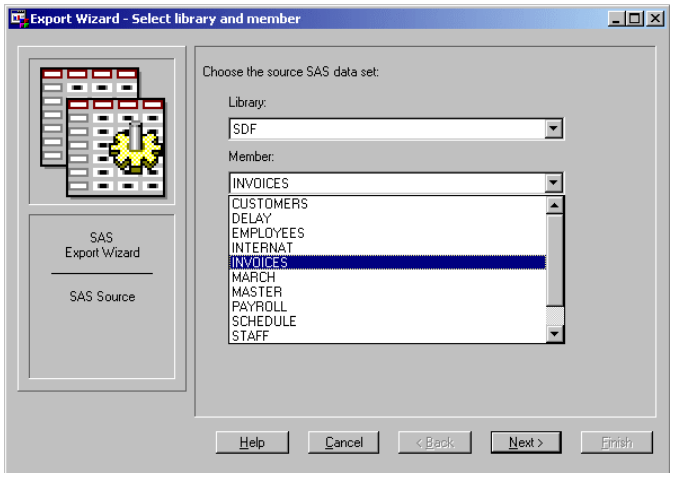
Display 11.4 Import Wizard: Save Generated Code



The Export wizard reads data from a SAS data set and writes it to an external file source. Complete the following steps to use the Export wizard on UNIX.

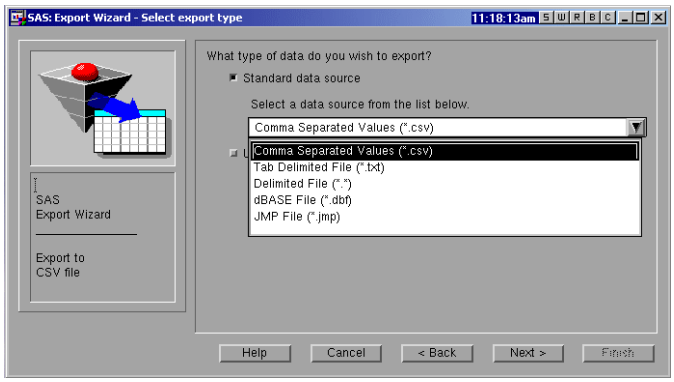
1 Select the SAS data set from which you want to export data.

Display 11.5 Export Wizard: Select Library and Member



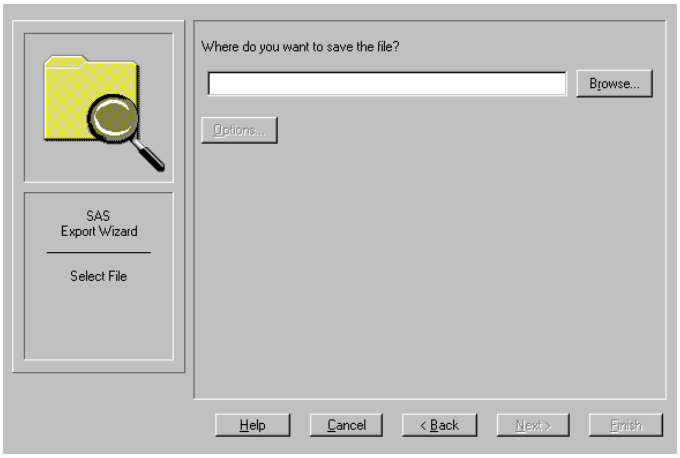
2 Select the type of data source to which you want to export files.

Display 11.6 Export Wizard: Select Export Type



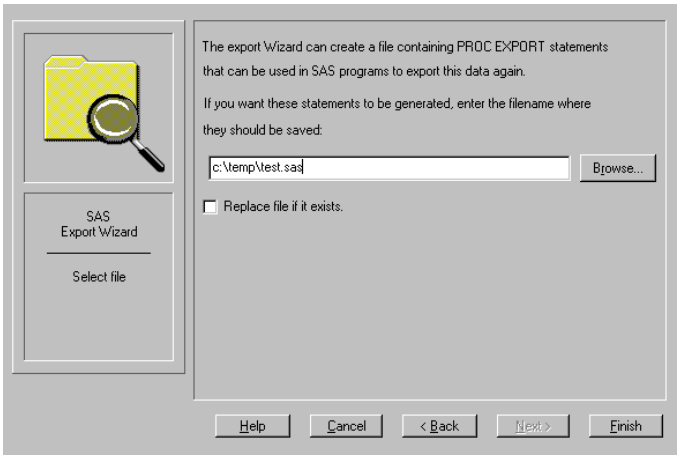
3 Assign a location to save the exported file.

Display 11.7 Export Wizard: Save Location



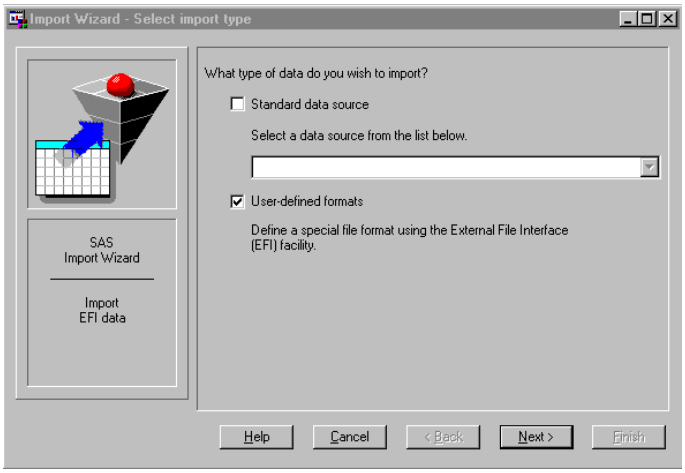
4 Save the generated PROC EXPORT code. (Optional)

Display 11.8 Export Wizard: Save Generated Code



From the primary window of the Import/Export wizard, you can also access the External File Interface (EFI). EFI is a point-and-click interface that enables you to read and write data in a file type that is not known to the Import/Export wizard. For example, you could use EFI to transfer data from a SAS data set to a file format that is proprietary for your company. Detailed information about using EFI is available from the Help button. To access the EFI, select the **User-defined formats** box on the primary Import/Export wizard window:

Display 11.9 Accessing the External File Interface



The IMPORT and EXPORT Procedures on UNIX

Like the Import/Export wizard, the IMPORT and EXPORT procedures enable you to transfer data between SAS and certain PC file formats that reside on UNIX. They also enable you to access other file formats that reside on the PC. (See Table 11.1 on page 118.)

IMPORT Procedure

Overview of the IMPORT Procedure

The syntax for the IMPORT procedure is shown here briefly but is described in detail in the *Base SAS Procedures Guide*. See Chapter 1, "Overview of the SAS/ACCESS Interface to PC Files," on page 3 for a list of file formats supported under your operating environment.

PROC IMPORT
DATAFILE=*"filename"* | TABLE=*"tablename"*
OUT=<*libref.*> *SAS-data-set* <*(SAS-data-set-options)*>
<DBMS=*identifier*> <REPLACE>;
<*data-source-statements*;>

Note: *identifier* should equal ACCESSCS for Microsoft Access, EXCELCS for Microsoft Excel, PCFS for JMP, DBF for DBF files, CSV for comma-delimited files, and DLM for other delimited files (in conjunction with the DELIMITER= option). △

After you invoke the IMPORT procedure, it reads the input file and writes the data to a SAS data set, where the names of the SAS variables are based on the column names of the input data. PROC IMPORT imports the data by one of the following methods:

- generated DATA step code
- generated SAS/ACCESS code
- translation engines.

You control the results with options and statements that are specific to your input data source. PROC IMPORT produces the specified SAS data set and writes information about the import to the SAS log. In the log, you see the DATA step or the SAS/ACCESS code that is generated by PROC IMPORT. If a translation engine is used, then the code is not submitted.

Example: Importing a Microsoft Access File Via the PC Files Server (File Resides on the PC)

This example imports a Microsoft Access table (Customers) and from it creates a permanent SAS data set (sasuser.cus). The Microsoft Access table has user-level security and, therefore, you need to specify the following statements: PWD=, UID=, and WGDB=.

```
proc import dbms=accesscs table="customers" out=sasuser.cust;
     database="c:\demo\customers.mdb";
     server=d2323;                    /* name of pc files server(required) */
     port=8621;                       /* Port number listening on the PC server */
     uid="bob";                            /* Microsoft Access database user ID */
     pwd="cat";                            /* Microsoft Access database password */
     wgdb="c:\winnt\system32\system.mdb";  /* Workgroup administrator database */
run;

proc print data=sasuser.cust;
run;
```

Note: See the *Base SAS Procedures Guide* for restrictions, interactions, and tips about PROC IMPORT. △

Example: Importing Microsoft Excel Workbook Files Via the PC Files Server (File Resides on the PC)

This example imports a worksheet (Invoice) in a Microsoft Excel workbook (sasdemo.xls) on the PC server (sales), and from it, creates a permanent SAS data set named work.invoice.

```
proc import dbms=excelcs out=work.invoice
            datafile="c:\excel\sasdemo.xls"
            replace;
     server="sales";   /* Name of PC files server */
     port=8621;    /* Port number listening on the PC server */
     version='2002';   /* Excel file version */
     sheet="Invoice";     /* Sheet name */
     scantext=yes;   /* Scan all rows data for the largest size */
     usedate=yes;   /* Use DATE format for date/time columns */
     scantime=yes;   /* Scan and identify time columns */
     dbsaslabel=none;     /* Leave SAS label names to be nulls */
     textsize=512;   /* Largest text size allowed */
run;
```

Note: See the *Base SAS Procedures Guide* for restrictions, interactions, and tips about PROC IMPORT. △

Example: Importing a JMP File Via the PC Files Server (File Resides on the PC)

This example imports a JMP file (test.jmp) and from it creates a temporary SAS data set (work.invoice).

```
proc import dbms=pcfs out=work.invoice datafile="c:\jmp\test.jmp";
     server=d2323;   /* Name of PC files server */
     port=8621;    /* Port number listening on the PC server */
run;
```

Note: See the *Base SAS Procedures Guide* for restrictions, interactions, and tips about PROC IMPORT. △

Example: Importing a DBASE File (File Resides on UNIX)

This example imports a DBASE file (test.dbf) and from it creates a temporary SAS data set (work.invoice).

```
proc import dbms=dbf out=work.invoice datafile="/tmp/invoice.dbf" replace;
run;
```

Note: See the *Base SAS Procedures Guide* for restrictions, interactions, and tips about PROC IMPORT. △

Example: Importing a Delimited File (File Resides on UNIX)

This example imports a comma-delimited file (test.csv) and from it creates a temporary SAS data set (work.invoice).

Note: It is not necessary to have a license for SAS/ACCESS to PC files to read in a CSV file in this manner. △

```
proc import dbms=csv out=work.invoice datafile="/tmp/test.csv";
run;
```

Note: See the *Base SAS Procedures Guide* for restrictions, interactions, and tips about PROC IMPORT. △

Example: Importing a JMP File (File Resides on UNIX)

This example imports a JMP file (invoice.jmp) and from it creates a temporary SAS data set (work.invoice)

```
proc import dbms=jmp out=work.invoice datafile="/tmp/invoice.jmp";
run;
```

Note: See the *Base SAS Procedures Guide* for restrictions, interactions, and tips about PROC IMPORT. △

EXPORT Procedure

Overview of the EXPORT Procedures

The syntax for the EXPORT procedure is shown here briefly but is described in detail in the *Base SAS Procedures Guide*. See "Methods for Accessing PC Files Data" on page 3 for a list of file formats supported under your operating environment.

PROC EXPORT
DATA=<*libref.*>*SAS-data-set* <*(SAS-data-set-options)*>
OUTFILE=*"filename"* | OUTTABLE=*"tablename"*
<<DBMS=*identifier*> <REPLACE>>;

Note: *identifier* should equal ACCESSCS for Microsoft Access, EXCELCS for Microsoft Excel, and PCFS for JMP. △

The EXPORT procedure reads data from a SAS data set and exports it to an external data source by using one of the following methods:

- generated DATA step code
- generated SAS/ACCESS code
- translation engines.

PROC EXPORT also controls the results with options and statements that are specific to the output data source.

Example: Exporting to a Microsoft Access File Via the PC Files Server (New File Will Reside on the PC)

This example uses a SAS data set (work.employee) to create a Microsoft Access table (Worktable). The new file will reside on the PC.

```
proc export dbms=accesscs data=work.employee outtable="emptable";
   database="c:\demo\customers.mdb";
   server=d2323;     /* Server name   */
   port=8621;         /* Port number  */
run;
```

Note: See the *Base SAS Procedures Guide* for restrictions, interactions, and tips about PROC EXPORT. △

Example: Exporting to a Microsoft Excel File Via the PC Files Server (New File Will Reside on the PC)

This example uses a SAS data set (work.employee) to create a worksheet (Employee) in a new Microsoft Excel workbook (newfile.xls). The new file will reside on the PC.

```
proc export dbms=excelcs data=work.employee outfile="c:\temp\newfile.xls" replace;
   sheet=employee;
   version="2002";     /* Excel version  */
   server=d2323;       /* Server name    */
   port=8621;         /* Port number    */
RUN;
```

Note: See the *Base SAS Procedures Guide* for restrictions, interactions, and tips about PROC EXPORT. △

Example: Exporting to a JMP File Via the PC Files Server (New File Will Reside on the PC)

This example uses a SAS data set (work.employee) to create a JMP file (employee.jmp) that will reside on the PC.

```
proc export dbms=pcfs data=work.employee outfile="c:\temp\employee.jmp";
   server=d2323;      /* Server name  */
   port=8621;        /* Port number  */
run;
```

Note: See the *Base SAS Procedures Guide* for restrictions, interactions, and tips about PROC EXPORT. △

Example: Exporting to a DBF File Via the PC Files Server (New File Will Reside on UNIX)

This example uses a SAS data set (work.employee) to create a DBF file (employee.dbf) that will reside on UNIX.

```
proc export dbms=dbf data=work.exployee outfile="/tmp/employee.dbf";
run;
```

Note: See the *Base SAS Procedures Guide* for restrictions, interactions, and tips about PROC EXPORT. △

Example: Exporting to a Comma-Delimited File (New File Will Reside on UNIX)

This example uses a SAS data set (work.employee) to create a flat file (employee.txt) that will reside on UNIX

Note: It is not necessary to have a license for SAS/ACCESS to PC files in order to create a delimited flat file in this manner. △

```
proc export data=work.employee outfile="/tmp/employee.txt" dbms=dbf csv;
run;
```

Note: See the *Base SAS Procedures Guide* for restrictions, interactions, and tips about PROC EXPORT. △

Example: Exporting to a Delimited File (New File Will Reside on UNIX)

The following example exports a SAS data set (myfile.class) and creates a delimited external file (Class). Notice that the DELIMITER= statement specifies the ampersand (&) delimiter to separate the column names in the new file. This example is repeated from the *Base SAS Procedures Guide*; see it for the SAS log.

```
proc export data=myfiles.class outfile="/myfiles/class" dbms=dlm;
     delimiter='&';
run;
```

The following code shows the first five rows of the external file that PROC EXPORT produces:

```
NAMES&SEX&AGE&HEIGHT&WEIGHT
Alice&F&13&56.5&84
```

```
Becka&F&13&65.3&98
Gail&F&14&64.3&90
Karen&F&12&56.3&77
Kathy&F&12&59.8&84.5
```

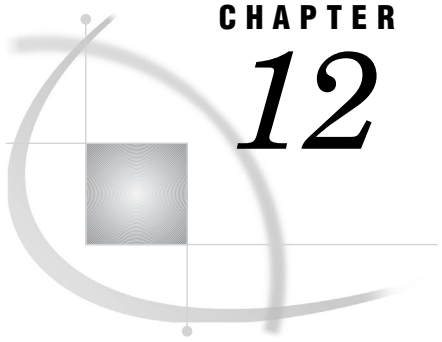
Note: See the *Base SAS Procedures Guide* for restrictions, defaults, requirements, and limitations of PROC EXPORT. △

Example: Exporting to a JMP File (File Resides on UNIX)

The following example exports a SAS data set (work.invoice) and creates a JMP file (invoice.jmp). This new file will reside on UNIX.

```
proc export dbms=jmp data=work.invoice outfile="/tmp/invoice.jmp";
run;
```

Note: See the *Base SAS Procedures Guide* for restrictions, defaults, requirements, and limitations of PROC EXPORT. △



CHAPTER

12

The Pass-Through Facility for PC Files on UNIX

Overview of the Pass-Through Facility for PC Files on UNIX

The SQL procedure implements the Structured Query Language (SQL) for SAS. See the SQL procedure topic in *Base SAS Procedures Guide* for information about PROC SQL. You can send data source specific SQL statements directly to a data source using an extension to the SQL procedure called the Pass-Through Facility.

This facility uses SAS/ACCESS to connect to a data source and to send statements directly to the data source for execution. This facility is a complement to the SAS/ACCESS LIBNAME statement. It enables you to use the SQL syntax of your data source, which can include any non-ANSI standard SQL that is supported by your data source.

The Pass-Through Facility enables you to do the following:

- establish and terminate connections with a data source using the facility's CONNECT and DISCONNECT statements
- send dynamic, non-query, data source specific SQL statements to a data source using the facility's EXECUTE statement
- retrieve data directly from a data source using the facility's CONNECTION TO component in the FROM clause of a PROC SQL SELECT statement.

You can use Pass-Through Facility statements in a PROC SQL query or you can store them in a PROC SQL view. When you create a PROC SQL view, any arguments that you specify in the CONNECT statement are stored with the view. Therefore, when the view is used in a SAS program, SAS can establish the appropriate connection to the data source.

Syntax for the Pass-Through Facility for PC Files

This section presents the syntax for the Pass-Through Facility statements and the CONNECTION TO component, which can be used in conjunction with the PROC SQL SELECT statement to query data from a data source.

PROC SQL <*options-list*>;

CONNECT TO *data-source-name* <AS *alias*> <(<*connect-statement-arguments*> <*database-connection-arguments*>)>;

DISCONNECT FROM *data-source-name* | *alias*;

EXECUTE (*data-source-specific-SQL-statement*) BY *data-source-name* | *alias*;

SELECT *column-list* FROM **CONNECTION TO** *data-source-name* | *alias* *(data-source-query)*

Return Codes

As you use the PROC SQL statements that are available in the Pass-Through Facility, any error conditions are written to the SAS log. The Pass-Through Facility generates return codes and messages that are available to you through the following two SAS macro variables:

SQLXRC
contains the data source return code that identifies the data source error.

SQLXMSG
contains descriptive information about the data source error that is generated by the data source.

The contents of the SQLXRC and SQLXMSG macro variables are printed in the SAS log using the %PUT macro. They are reset after each Pass-Through Facility statement has been executed.

CONNECT Statement

Establishes a connection with the data source

Valid in: PROC SQL steps

Syntax

CONNECT TO *data-source-name* <AS *alias*> <(<*connect-statement-arguments*> <*database-connection-arguments*>)>;

Arguments

data-source-name
identifies the data source to which you want to connect. Since this method requires connecting through the PC files server, you must use PCFILES as your data source. You can also specify an optional alias in the CONNECT statement.

alias
specifies an optional alias for the connection that has 1 to 32 characters. If you specify an alias, the keyword AS must appear before the alias. If an alias is not specified, the data source name is used as the name of the Pass-Through connection.

connect-statement-arguments
specifies arguments that indicate whether you can make multiple connections, shared or unique connections, and so on to the database. Some of these arguments are optional.

database-connection-arguments
specifies the data source specific arguments that are needed by PROC SQL to connect to the data source. These arguments are not required and the default behavior opens a dialog box that prompts you for information they provide.

Database Connection Arguments

The arguments that are listed below are available with the Pass-Through Facility for PC files. These arguments extend some of the LIBNAME statement connection management features to the Pass-Through Facility.

The following options are used when connecting to the PC files server.

PATH="*path-for-file*"
specifies the data source file location for the Microsoft Access database file or Microsoft Excel workbook file.

PASSWORD="*user-password*"
specifies a password for the user account, if required by the data source. Passwords are case-sensitive.

USER="*user-ID*"
specifies a default user account name. The default value is Admin. User names can be 1 to 20 characters long and can include alphabetic characters, accented characters, numbers, and spaces. If you have user-level security set in your MDB file, you need to use this option and the PASSWORD= option to be able to access your file.

The following options are used only for Microsoft Access.

DBPASSWORD="*database-file-password*"
enables you to access your file if you have database-level security set in your MDB file. A database password is case-sensitive and can be defined instead of user-level security

DBSYSFILE="*workgroup-information-file*"
contains information about the users in a workgroup based on information that you define for you Microsoft Access database. Any user and group accounts or passwords you create are saved in the new workgroup information file.

The following option is used only for Microsoft Excel.

VERSION=2002 | 2000 | 97 | 95 | 5
sets the version of Microsoft Excel. The default value is 97.

2002	sets the version of Microsoft Excel to 2002.
2000	sets the version of Microsoft Excel to 2000.
97	sets the version of Microsoft Excel to 97.
95	sets the version of Microsoft Excel to 95.
5	sets the version of Microsoft Excel to 5.

CONNECT Statement Arguments

The arguments that are listed below are available with the Pass-Through Facility CONNECT statement for PC files. These arguments extend some of the LIBNAME statement connection management features to the Pass-Through Facility.

AUTOCOMMIT=YES | NO
determines whether the ACCESS engine commits (saves) updates as soon as the user submits them.

YES
specifies that updates are committed (that is, saved) to the table as soon as they are submitted, and no rollback is possible.

NO
specifies that the SAS/ACCESS engine automatically performs the commit when it reaches the end of the file.

Default: YES

Note: The default value for this option is different from the LIBNAME option. △

COMMAND_TIMEOUT=*number-of-seconds*
specifies the number of seconds that pass before a data source command times out.

Default: 0 (no timeout)

Alias: TIMEOUT=

CONNECTION= SHARE |GLOBAL
specifies whether multiple CONNECT statements for a data source can use the same connection. The CONNECTION= option enables you to control the number of connections, and therefore transactions, that your SAS/ACCESS engine executes and supports for each CONNECT statement.

SHARED
specifies that the CONNECT statement makes one connection to the DBMS. Only Pass-Through statements that use this alias share the connection.

GLOBAL
specifies that multiple CONNECT statements using identical values for CONNECTION=, CONNECTION_GROUP=, and any database connection arguments can share the same connection to the DBMS.
GLOBAL is the default value for CONNECTION= when you specify CONNECTION_GROUP=.

Default: SHARED

CONNECTION_GROUP= *connection-group*
causes operations against multiple Pass-Through Facility CONNECT statements to share a connection to the data source.

CURSOR_TYPE= DYNAMIC | FORWARD_ONLY | KEYSET_DRIVEN | STATIC |
specifies the cursor type for read-only and updatable cursors.

DYNAMIC
specifies that the cursor reflects all of the changes that are made to the rows in a result set as you move the cursor. The data values and the membership of rows in the cursor can change dynamically on each fetch. This is the default for the DB2 UNIX, PC files, and Microsoft SQL Server interfaces.

FORWARD_ONLY
specifies that the cursor behaves like a DYNAMIC cursor, except that it only supports fetching the rows sequentially.

KEYSET_DRIVEN
specifies that the cursor determines which rows belong to the result set when the cursor is opened. However, changes that are made to these rows are reflected as you scroll around the cursor.

STATIC
specifies that the complete result set is built when the cursor is opened. No changes that are made to the rows in the result set after the cursor is opened are reflected in the cursor. Static cursors are read-only.

Default: none

Alias: CURSOR=

DBGEN_NAME=DBMS | SAS
specifies that the data source columns are renamed, and specifies the format that the new names will follow.

DBMS
specifies that the data source columns are renamed to valid SAS variable names. Disallowed characters are converted to underscores. If a column is converted to a name that already exists, then a sequence number is appended to the end of the new name.

SAS
specifies that data source columns are renamed to the format _COL*n*, where *n* is the column number (starting with zero).

Default: DBMS

DBMAX_TEXT=*n*
specifies an integer between 1 and 32,767 that indicates the maximum length for a character string. Longer character strings are truncated. This option only applies when you are reading, appending, and updating Microsoft Access or Excel character data from SAS.

Note: Although you can specify a value less than 256, it is not recommended. △

Default: 1,024

DEFER=NO | YES
enables you to specify when the CONNECT statement occurs.

NO
specifies that the connection to the data source occurs when the libref is assigned by a LIBNAME statement.

YES
specifies that the connection to the data source occurs when a table in the data source is opened.

Default: NO

PORT="*port-number*"
The port or service name on the PC that the SAS PC files server is listening on. This port or service name is displayed on the SAS PC Files Server window when it is started on the PC. This is a required field when connecting to the PC files server for data.

Aliases: SERVICE=, SERVICE_NAME=

READBUFF=*number-of-rows*
specifies the number of rows to use when you are reading data from a data source. Setting a higher value for this option reduces I/O and increases performance, but

also increases memory usage. Additionally, if too many rows are read at once, values returned to SAS might be out of date.

Default: 1

Alias: ROWSET=
ROWSET_SIZE=

SERVER="*pc-server-hostname* "
specifies the computer name of the PC on which you started the PC files server. This name is required by UNIX users to connect to this server machine and is reflected on the server control panel. This is a required field when connecting to the PC files server for data.

This hostname can be specified as a simple computer name (for example, wxp320), a fully qualified network name (for example, **`wxp320.domain.com`**), or an IP address.

STRINGDATES=YES | NO
specifies whether datetime values are read from the data source as character strings or as numeric date values. STRINGDATES= is not available as a data set option.

YES
specifies that SAS/ACCESS reads datetime values as character strings.

NO
specifies that SAS/ACCESS reads datetimes values as numeric date values.

Default: NO

Alias: STRDATES

Details

The CONNECT statement establishes a connection with the data source. You establish a connection to send data source specific SQL statements to the data source or to retrieve data source data. The connection remains in effect until you issue a DISCONNECT statement or terminate the SQL procedure.

To connect to a data source using the Pass-Through Facility, complete the following steps:

1. Initiate a PROC SQL step.
2. Use the Pass-Through Facility's CONNECT statement with the PC files engine name, and (optionally) assign an alias.
3. Specify any arguments needed to connect to the database.
4. Specify any attributes for the connection.

The CONNECT statement is optional for some data sources. However, if it is not specified, the default values for all of the database connection arguments are used.

Any return code or message that is generated by the data source is available in the macro variables SQLXRC and SQLXMSG after the statement executes. See "Return Codes" on page 36 for more information about these macro variables.

Example

The following example uses the CONNECT statement with the PATH= option to connect to the Microsoft Access database file, **c:\demo.mdb:**

```
proc sql;
connect to pcfiles as db (server=d2323 path="c:\demo.mdb");
```

DISCONNECT Statement

Terminates the connection to the data source

Valid in: PROC SQL steps (when accessing PC files data using SAS/ACCESS software)

Syntax

DISCONNECT FROM PCFILES | *alias*

Arguments

alias
specifies an alias for the connection that was defined in the CONNECT statement.

Details

The DISCONNECT statement ends the connection with the data source. If the DISCONNECT statement is omitted, an implicit DISCONNECT is performed when PROC SQL terminates. The SQL procedure continues to execute until you submit a QUIT statement, another SAS procedure, or a DATA step.

Any return code or message that is generated by the data source is available in the macro variables SQLXRC and SQLXMSG after the statement executes. See "Return Codes" on page 36 for more information about these macro variables.

Example

The following example, after the connection and SQL processing, uses the DISCONNECT statement to disconnect the connection from the database, and uses the QUIT statement to quit the SQL procedure:

```
disconnect from pcfiles;
quit;
```

EXECUTE Statement

Sends data source specific, non-query SQL statements to the data source

Valid in: PROC SQL steps

Syntax

EXECUTE (*data-source-specific-SQL-statement*) BY PCFILES | *alias*;

Arguments

(*data-source-specific-SQL-statement*)
a dynamic nonquery, data source specific SQL statement. This argument is required and must be enclosed in parentheses. However, the SQL statement cannot contain a semicolon because a semicolon represents the end of a statement in SAS. The SQL statement can be case-sensitive, depending on your data source, and it is passed to the data source exactly as you type it.

Any return code or message that is generated by the data source is available in the macro variables SQLXRC and SQLXMSG after the statement executes. See "Return Codes" on page 36 for more information about these macro variables.

alias
specifies an alias for the connection that was defined in the CONNECT statement. (You cannot use an alias if the CONNECT statement was omitted.)

Details

The EXECUTE statement sends dynamic nonquery, data source specific SQL statements to the data source and processes those statements.

The EXECUTE statement cannot be stored as part of a Pass-Through Facility query in a PROC SQL view.

Useful Statements to Include in EXECUTE Statements

You can pass the following statements to the data source by using the Pass-Through Facility's EXECUTE statement.

CREATE
creates a data source table, view, index, or other data source object, depending on how the statement is specified.

DELETE
deletes rows from a data source table.

DROP
deletes a data source table, view, or other data source object, depending on how the statement is specified.

GRANT
gives users the authority to access or modify objects such as tables or views.

INSERT
adds rows to a data source table.

REVOKE
revokes the access or modification privileges that were given to users by the GRANT statement.

UPDATE
modifies the data in the specified columns of a row in a data source table.

For more information about these and other SQL statements, see the SQL documentation for your data source.

Example

The following example, after the connection, uses the EXECUTE statement to drop a table, create a table, and insert a row of data:

```
execute(drop table 'My Invoice') by pcfiles;
execute(create table 'My Invoice'(
'Invoice Number' LONG not null,
'Billed To' VARCHAR(20),
'Amount' CURRENCY,
'BILLED ON' DATETIME)) by pcfiles;
execute(insert into 'My Invoice'
values( 12345, 'John Doe', 123.45, #11/22/2003#)) by pcfiles;
```

CONNECTION TO Component

Retrieves and uses data source data in a PROC SQL query or view

Valid in: PROC SQL step SELECT statements

Syntax

CONNECTION TO PCILES <*AS alias*> <*(database-connection-options)*>

Arguments

alias
specifies an alias, if one was defined in the CONNECT statement.

Details

The CONNECTION TO component specifies the data source connection that you want to use or that you want to create (if you have omitted the CONNECT statement). CONNECTION TO then enables you to retrieve data source data directly through a PROC SQL query.

You use the CONNECTION TO component in the FROM clause of a PROC SQL SELECT statement:

SELECT *column-list*

FROM CONNECTION TO *data source-name (data source-query);*

CONNECTION TO can be used in any FROM clause, including those in nested queries (that is, in subqueries).

You can store a Pass-Through Facility query in a PROC SQL view and then use that view in SAS programs. When you create a PROC SQL view, any options that you specify in the corresponding CONNECT statement are stored too. Thus, when the

PROC SQL view is used in a SAS program, SAS can establish the appropriate connection to the data source.

Because external data sources and SAS have different naming conventions, some data source column names might be changed when you retrieve data source data through the CONNECTION TO component.

Example

The following example, after the connection, uses the CONNECTION TO component to query a table or a subtable:

```
select * from connection to pcfiles(select * from 'my invoice');
select * from connection to pcfiles
(select 'Invoice Number', Amount from 'my invoice');
```

The following code creates a SAS data set (Newtable) from a Microsoft Access table:

```
proc sql;
connect to pcfiles(server=d2323 port=8621
                   path="c:\temp\household.inventory.mdb");
create table newtable as select * from;
connect to pcfiles(select * from rooms);
disconnect from pcfiles;
quit;
```

The following code connects to an Excel file and query the INVOICE table (range) within the Excel workbook:

```
proc sql dquote=ansi;
connect to pcfiles (path="c:\sasdemo\sasdemo.xls" server=d2323 port=8621);
select * from connection to pcfiles
   (select * from invoice);
disconnect from pcfiles;
quit;
```

The following code, to create a SAS data set (Work) from a list of tables in a Microsoft SQL Server database, using a PC files server via ODBC:

```
proc sql dquote=ansi;
connect to pcfiles (server=d2323 port=8621 dsn=mqis user=scott pwd=tiger);
create table work as select * from connection to pcfiles (PCFILES::SQLTABLES);
disconnect from pcfiles;
quit;
```

Special PC Files Queries

The following special queries are supported by the SAS/ACCESS interface to PC files on UNIX. Many databases provide or use system tables that allow queries to return the list of available tables, columns, procedures, and other useful information. In PC files, much of this functionality is provided through special APIs (application programming interfaces) in order to accommodate databases that do not follow the SQL table structure. You can use these special queries on non-SQL and SQL databases. The general format of the special queries is as follows:

PCFILES::SQLAPI "*parameter 1*","*parameter n*"

where

PCFILES::
is required to distinguish special queries from regular queries.

SQLAPI
is the specific API that is being called. Both PCFILES:: and SQLAPI are case sensitive.

"*parameter n*"
is a quoted string that is delimited by commas.

Within the quoted string, two characters are universally recognized: the percent sign (%) and the underscore (_). The percent sign matches any sequence of zero or more characters; the underscore represents any single character. Each driver also has an escape character that can be used to place characters within the string. Consult the driver's documentation to determine the valid escape character.

The values for the special query arguments are DBMS specific. For example, you supply the fully qualified table name for a "*Catalog*" argument. In dBase, the value of "*Catalog*" might be **`c:\dbase\tst.dbf`** and in SQL Server, the value might be **`test.customer`**. In addition, depending on the DBMS that you are using, valid values for "*Schema*" argument might be a user ID, a database name, or a library. All arguments are optional. If you specify some but not all the arguments within a parameter, use a comma to indicate the omitted parameters. If you do not specify any parameters, commas are not necessary.

Note: These special queries might not be available for all PCFILES drivers. △

The following special queries are supported:

PCFILES::SQLTables <"*Catalog*", "*Schema*", "*Table-name*", "*Type*">
returns a list of all the tables that match the specified arguments. If no arguments are specified, all accessible table names and information are returned.

PCFILES::SQLColumns <"*Catalog*", "*Schema*", "*Table-name*", "*Column-name*">
returns a list of all the columns that match the specified arguments. If no arguments are specified, all accessible column names and information are returned.

PCFILES::SQLColumnPrivileges <"*Catalog*", "*Schema*", "*Table-name*", "*Column-name*">
returns a list of all the column privileges that match the specified arguments. If no arguments are specified, all accessible column names and privilege information are returned.

PCFILES::SQLForeignKeys <"*PK-catalog*", "*PK-schema*", "*PK-table-name*", "*FK-catalog*", "*FKschema*", "*FKtable-name*">
returns a list of all the columns that comprise foreign keys that match the specified arguments. If no arguments are specified, all accessible foreign key columns and information are returned.

PCFILES::SQLPrimaryKeys <"*Catalog*", "*Schema*", "*Table-name*">
returns a list of all the columns that compose the primary key that matches the specified table. A primary key can be composed of one or more columns. If no table name is specified, this special query fails.

PCFILES::SQLProcedureColumns <"*Catalog*", "*Schema*", "*Procedure-name*", "*Column-name*">
returns a list of all the procedure columns that match the specified arguments. If no arguments are specified, all accessible procedure columns are returned.

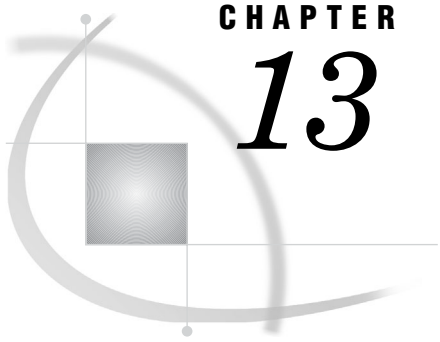
PCFILES::SQLProcedures <"*Catalog*", "*Schema*", "*Procedure-name*">
returns a list of all the procedures that match the specified arguments. If no arguments are specified, all accessible procedures are returned.

PCFILES::SQLSpecialColumns <"*Identifier-type*", "*Catalog-name*", "*Schema-name*", "*Table-name*", "*Scope*", "*Nullable*">
returns a list of the optimal set of columns that uniquely identify a row in the specified table.

PCFILES::SQLStatistics <"*Catalog*", "*Schema*", "*Table-name*">
returns a list of the statistics for the specified table name, with options of SQL_INDEX_ALL and SQL_ENSURE set in the SQLStatistics API call. If the table name argument is not specified, this special query fails.

PCFILES::SQLTablePrivileges <"*Catalog*", "*Schema*", "*Table-name*">
returns a list of all the tables and associated privileges that match the specified arguments. If no arguments are specified, all accessible table names and associated privileges are returned.

PCFILES::SQLGetTypeInfo
returns information about the data types that are supported in the data source.



CHAPTER

13

The DBF and DIF Procedures on UNIX

Introduction to the DBF and DIF Procedures

The DBF and DIF procedures give UNIX users an alternative way of accessing DBF and DIF files. Instead of creating access descriptors and view descriptors, you can convert these PC file types to SAS data sets, or vice versa.

Note: The DBF and DIF files must reside locally on the UNIX machine. △

You can use the DBF and DIF procedures to convert a DBF or DIF file to a SAS data set or to convert a SAS data set to a DBF or DIF file.

The DBF Procedure

Converts a dBASE file to SAS data set or a SAS data set to a dBASE file

Restrictions: none

Syntax

PROC DBF *options*;

PROC DBF Options

DB2|DB3|DB4|DB5=*fileref* | *filename*
specifies the fileref or filename of a DBF file.

The DB*n* option must correspond to the version of dBASE with which the DBF file is compatible. You specify the version with the DB*n* option, where *n* is the version number and can have a value of 2, 3, 4, or 5. You can specify only one of these values.

If you specify a fileref, the FILENAME statement that you used to define it must specify the filename plus a .dbf extension (for example, **`filename myref '/my_dir/myfile.dbf'`**).

If you specify a filename instead of a fileref, you can only specify the name itself (omitting the .dbf extension) and the file must be in the current directory. For

example, this PROC DBF statement creates the EMP.DBF file (with the name in uppercase) from the MyLib.Employee data set:

```
proc dbf db5=emp data=mylib.employee;
```

You *cannot* specify **emp.dbf** or a full pathname (**proc dbf db5='/my/unix_directory/emp.dbf'**) in the DB*n*= option.

The DB*n*= option is required.

DATA=<*libref.*>*member*
names the input SAS data set. Use this option if you are creating a DBF file from a SAS data set. If you use the DATA= option, do not use the OUT= option. If you omit the DATA= option, SAS software creates an output SAS data set from the DBF file.

OUT=<*libref.*>*member*
names the SAS data set that is created to hold the converted data. Use this option only if you are creating a SAS data set from a DBF file and you did not specify the DATA= option.

If OUT= is omitted, SAS creates a temporary data set in the Work library. (Under UNIX and OS/390, the temporary data set is named Data1 [...Data*n*]; under windows, it is called _DATA_.) If OUT= is omitted or if you do not specify a two-level name in the OUT= option, the data set remains available during your current SAS session, but it is not permanently saved.

Details

The DBF procedure converts dBASE files to SAS data sets that are compatible with the current release of SAS, or it converts SAS data sets to DBF files.

PROC DBF produces one output file but no printed output. The output file contains the same information as the input file but in a different format.

The DBF procedure works with DBF files created by all the current versions and releases of dBASE (II, III, III PLUS, IV, and 5.0) and with most DBF files that are created by other software products.

Future versions of dBASE files might not be compatible with the current version of the DBF procedure. To use the DBF procedure, you must have a SAS/ACCESS interface to PC files license.

Converting DBF Fields to SAS Variables

Numeric variables are stored in character form by DBF files. These numeric variables become SAS numeric variables when converted from a DBF file to a SAS data set. If a DBF numeric value is missing, the corresponding dBASE numeric field is filled with the character **9**, by default.

Character variables become SAS character variables. Logical fields become SAS character variables with a length of 1. Date fields become SAS date variables. When you are converting a DBF file to a SAS data set, fields whose data is stored in auxiliary DBF files (Memo and General fields) are ignored.

When a dBASE II file is translated into a SAS data set, any colons in dBASE variable names are changed to underscores in SAS variable names. Conversely, when a SAS data set is translated into a dBASE file, any underscores in SAS variable names are changed to colons in dBASE field names.

Converting SAS Variables to DBF Fields

Numeric variables are stored in character form by DBF files. SAS numeric variables become numeric variables with a length of 16 when converting from a SAS data set to a DBF file. A SAS numeric variable with a decimal value must be stored in a decimal

format in order to be converted to a DBF numeric field with a decimal value. In other words, unless you associate the SAS numeric variable with an appropriate format in a SAS FORMAT statement, the corresponding DBF field will not have any value to the right of the decimal point. You can associate a format with the variable in a SAS data set when you create the data set or by using the DATASETS procedure.

If the number of digits — including a possible decimal point — exceeds 16 a warning message is issued and the DBF numeric field is filled with the character **9**. All SAS character variables become DBF fields of the same length. When you are converting data from a SAS data set to a DBF file that is compatible with dBASE III or later, SAS date variables become DBF date fields. When you are converting data from a SAS data set to a dBASE II file, SAS date variables become dBASE II character fields in the form *YYYYMMDD*.

Transferring Other Software Files to DBF Files

You might find it helpful to save another software vendor's file to a DBF file and then convert that file into a SAS data set. UNIX users find this especially helpful. For example, you could save an Excel XLS file to a DBF file (by selecting

File ▶ Save As ▶ EMP.DBF

from within an Excel spreadsheet and selecting the Emp.dbf file) and then use PROC DBF to convert that file into a SAS data set. Or you could do the reverse: use PROC DBF to convert a SAS data set into a DBF file and then load that file into an Excel spreadsheet.

Examples

Example 1: Converting a dBASE II File to a SAS Data Set In this example, a dBASE II file named Employee.dbf is converted to a SAS data set. Because no FILENAME statement is specified, the last level of the filename is assumed to be .dbf and the file is assumed to be in your current directory and in uppercase.

```
libname save '/my/unx_save_dir';
proc dbf db2=employee out=save.employee;
run;
```

Example 2: Converting a SAS Data Set to a dBASE 5 File In this example, a SAS data set is converted to a dBASE 5 file. A FILENAME statement specifies a fileref that names the dBASE 5 file. You must specify the FILENAME statement before the PROC DBF statement.

```
libname mylib '/my/unix_directory';
filename employee '/sasdemo/employee.dbf';
proc dbf db5=employee data=mylib.employee;
run;
```

The DIF Procedure

Converts a DIF file to SAS data set or a SAS data set to a DIF file

Restrictions: The DIF procedure is only available under UNIX and Windows operating environments.

Syntax

PROC DIF *options*;

PROC DIF Options

DIF=*fileref* | *filename*
specifies the fileref or filename of a DIF file.

If you specify a fileref, the FILENAME statement that you used to define in must specify the filename plus a .dif extension (for example, **`filename myref '/my_dir/myfile.dif'`**).

If you specify a filename instead of a fileref, you can only specify the name itself (omitting the .dif extension) and the file must be in the current directory. For example, this PROC DIF statement creates the Emp.dif file from the MyLib.Employee data set:

```
proc dif dif=emp data=mylib.employee;
```

You *cannot* specify **`emp.dif`** or a full pathname (**`proc dif dif='/my/unix_directory/emp.dif'`**).

DATA=<*libref.*>*member*
names the input SAS data set. Use this option if you are creating a DIF file from a SAS data set. If you use this option, do not use the OUT= option. If you omit the DATA= option, SAS creates an output SAS data set from the DIF file.

OUT=<*libref.*>*member*
names the SAS data set to hold the converted data. You use this option only if you omit the DATA= option and you are creating a SAS data set from a DIF file.

If OUT= is omitted, SAS creates a temporary data set in the Work library. (Under UNIX, the temporary data set is named Data1 [...Data*n*]; under windows, it is called _DATA_. If OUT= is omitted or if you do not specify a two-level name in the OUT= option, the data set remains available during your current SAS session but is not permanently saved.

LABELS
causes PROC DIF to write the names of the SAS variables as the first row of the DIF file and a row of blanks as the second row of the DIF file. The actual data portion of the DIF file begins in the third row. The LABELS option is allowed only when you are converting a SAS data set to a DIF file.

PREFIX=*name*
specifies a prefix to be used in constructing SAS variable names when you are converting a DIF file to a SAS data set. For example, if PREFIX=VAR, the new variable names are VAR1, VAR2, ... VAR*n*. If you omit the PREFIX= option, PROC DIF assigns the names Col1, Col2, ... Col*n*.

SKIP=*n*
specifies the number of rows, beginning at the top of the DIF file, to be ignored when converting a DIF file to a SAS data set. For example, suppose the first row of your DIF file contains column headings and the second row of your DIF file is a blank row. The actual data in your DIF file begin in row 3. You should specify SKIP=2 so that PROC DIF ignores the nondata portion of your DIF file. Alternatively, you could delete the first two rows of your DIF file before using PROC DIF.

Details

The DIF procedure converts data interchange format (DIF) files to SAS data sets that are compatible with the current release of SAS software, or it converts SAS data sets to DIF files.

PROC DIF produces one output file but no printed output. The output file contains the same information as the input file but in a different format.

Software Arts, Inc. developed the data interchange format to be used as a common language for data. Originally, DIF was made popular by products such as Lotus 1-2-3 and VisiCalc. Although DIF is not as popular today as it once was, it is still supported by many software products.

Note: Any DIF file that you plan to convert to a SAS data set should be in a tabular format. All items in a given column should represent the *same* type of data. If any rows in the DIF file contain inconsistent data — for example, a row of underscores, dashes, or blanks — delete these rows before converting the DIF file to a SAS data set. It is recommended that you make a backup copy of your DIF table before you make these modifications. △

When you are converting data from a DIF file to a SAS data set, each row of the DIF file becomes an observation in the SAS data set. Conversely, when you are converting a SAS data set to a DIF file, each SAS observation becomes a row in the DIF file. To use the DIF procedure, you must have a SAS/ACCESS interface to PC files license.

Converting DIF Variables to SAS Variables

Character variables in a DIF file (sometimes referred to as *string values*) become SAS character variables of length 20. If a DIF character variable's value is longer than 20 characters, it is truncated to a length of 20 in the SAS output data set. The quotation marks that normally enclose character variable values in a DIF file are removed when the value is converted to a SAS character value.

Numeric variables, which can be represented in either integer or scientific notation in a DIF file, become SAS numeric variables when a DIF file is converted to a SAS data set.

Transferring SAS Data Sets to and from Other Software Products Using DIF

DIF files are not generally used as the native file format for a software product's data storage. Therefore, transferring data between SAS and another software product is a two-step process when using DIF files.

To send SAS data sets to another software product using DIF files, you must first run PROC DIF to convert your SAS data set to a DIF file. Use whatever facility is provided by the target software product to read the DIF file. For example, you use the Lotus 1-2-3 Translate Utility to translate a DIF file to a 1-2-3 worksheet file. (This facility might be provided by an import tool or from an Open window in that software product.) After the application reads the DIF file data, the data can be manipulated and saved in the application's native format.

To transfer data in the opposite direction — from a software product to a SAS data set — the process is reversed. First, export the data to a DIF file and then run PROC DIF to read the DIF file into a SAS data set.

Missing Values

The developers of the data interchange format (DIF) files suggest that you treat all numeric values that have a value indicator other than V as missing values. PROC DIF follows this convention. When a DIF file is converted to a SAS data set, any numeric value with a value indicator other than V becomes a SAS missing value.

When a SAS data set that has missing values for some numeric variables is converted to a DIF file, the following assignments are made in the DIF file for the variables with missing values:

- the type indicator field value is set to **0**
- the number field value contains a string of 16 blanks
- the value indicator is set to **NA**.

Examples

Example 1: Converting a DIF File to a SAS Data Set In this example, a DIF file named Employee.dif is converted to a SAS data set. Because no FILENAME statement is specified, the last level of the filename is assumed to be .dif, and the file is assumed to be in your current directory and in uppercase.

```
libname save '/my/my_unx_dir';
proc dif dif=employee out=save.employee;
run;
```

Example 2: Converting a SAS Data Set to a DIF File In this example, a SAS data set is converted to a DIF file. A FILENAME statement is used to specify a fileref that names the DIF file. You must specify the FILENAME statement before the PROC DIF statement.

```
filename employee '/sasdemo/employee.dif';
proc dif dif=employee data=save.employee;
run;
```

See Also

"Programmer's Guide to the DIF," *Software Arts Technical Notes* (SATN-18).

CHAPTER
14

JMP Essentials for PC Files

Overview of JMP Essentials

SAS/ACCESS software for PC files works with JMP files that are created by JMP Versions 1 to 5 including both Windows and Macintosh based versions. This section describes only how JMP is processed within SAS/ACCESS. For more information about a JMP concept or term, see the JMP documentation packaged with your system.

JMP Files

A JMP file is a file format created by the JMP software program, which is an interactive statistics package. JMP is available for both Windows and Macintosh.

A JMP file contains data that is organized in a tabular format of fields and records. Each field can contain one type of data, and each record can hold one data value for each field.

JMP File Naming Conventions

Filenames must follow operating system specific conventions. Refer to the documentation that comes with your JMP product.

JMP Variable Naming Conventions

Variable names can be up to 31 characters in length. When you are reading a JMP file, any embedded blank or special character in a variable name is replaced with an underscore (_). This is noted in the log.

JMP Data Types

Every field in a JMP file has a name and a data type. The data type indicates how much physical storage to set aside for the field and the format in which the data is stored. The following list describes each data type.

Character
: specifies a field for character string data. The maximum length is 255 characters. Characters can be letters, digits, spaces, or special characters.

Numeric
: specifies an 8 byte floating point number. This is also called a double precision number.

  When you are reading data, this maps directly to the SAS double precision number. When you are writing data, all SAS numeric variables (regardless of length) become JMP numeric variables.

Rowstate
: specifies an integer variable that takes on the value of 1 or missing. When you are reading data, this maps to a SAS double precision number.

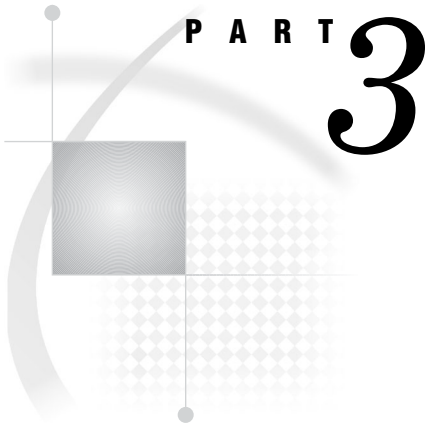
Date
: specifies the date format. When you are reading data, the date values are mapped to a SAS number and scaled to the base date. The JMP date display format maps to the appropriate SAS date display format. When you are writing data, the SAS numeric variable's output format is checked to determine if it is a date format. If so, the SAS numeric value is scaled to a JMP date value with the appropriate date display format.

DateTime
: specifies the datetime format. When you are reading data, the datetime values are mapped to a SAS number and scaled to the base datetime. The JMP datetime display format maps to the appropriate SAS datetime display format. When you are writing data, the SAS numeric variable's output format is checked to determine if it is a datetime format. If so, the SAS numeric value is scaled to a JMP datetime value with the appropriate datetime display format.

Time
: specifies the time format. When you are reading data, the time values are mapped to a SAS number and scaled to the base time. The JMP time display format maps to the appropriate SAS time display format. When you are writing data, the SAS numeric variable's output format is checked to determine if it is a time format. If so, the SAS numeric value is scaled to a JMP time value with the appropriate time display format.

JMP Missing Values

JMP supports a single missing value in all variable types other than character. When you are reading a JMP file, missing values map to the (.) missing value. When you are writing a JMP file, all SAS missing values will map to the single JMP missing value.



PART 3

File Format Specific Reference

CHAPTER

15

Microsoft Excel XLS Files

How to Access XLS Files from SAS

You can interact with Excel files from SAS by using the following features:

LIBNAME statement (UNIX and Windows operating environments)
provides direct, transparent access to data in PC file formats. Available for Excel 5, 95, 97, 2000, or 2002 formats. For details, refer toChapter 2, “The LIBNAME Statement for PC Files on Windows,” on page 5.

Pass-Through Facility (UNIX and Windows operating environments)
enables you to interact with Microsoft Excel (5, 95, 97, 2000, or 2002) data using the data source’s SQL syntax without leaving your SAS session. The SQL statements are passed directly to the data source for processing. For details, refer to Chapter 3, “The Pass-Through Facility for PC Files on Windows,” on page 35.

Import/Export wizard or procedures (UNIX and Windows operating environments)
enable you to transfer data between SAS and several PC file formats. Available for Excel 4, 5, 95, 97, 2000, or 2002 formats. For details, refer to “Import/Export Wizard” on page 50.

ACCESS procedure (Windows operating environments)
creates descriptor files that describe data in a PC file to SAS, enabling you to directly read, update, or extract PC file data into a SAS data file. Available for Excel 4, 5, or 95 formats.

DBLOAD procedure (Windows operating environments)
creates PC files and loads them with data from a SAS data set. Available for Excel 4, 5, or 95 formats.

LIBNAME Statement Data Conversions for XLS Files

The following table shows the default SAS variable formats that SAS/ACCESS assigns to XLS data types when you read or import XLS data with the LIBNAME statement.

Table 15.1 Default SAS Variable and Type Formats for Excel Formats

Excel Column Format	SAS Variable Format	SAS Variable Type
Text	$w.	character
General		
Number		
Scientific	See Note 3	numeric
Percentage		
Fraction		
Currency	DOLLAR21.2	numeric
Accounting		
Date		
Datetime	DATE9. See Notes 1 and 2	numeric
Time		

1 The default format is DATE9. However, you can use the SASDATEFMT option to change the format to other date or datetime formats. The LIBNAME engine automatically converts the internal date value for you.
2 If you have a time only field in your Microsoft Excel range, you can use SASDATEFMT to assign it with the SAS TIME. format. Note that the SAS date/time value uses 01Jan1960 as a cutoff line while the Jet provider date/time value uses 30Dec1899 as a cutoff line.
3 To access Fraction or Percent format data in your Excel file, you can use the FORMAT statement to assign the FRACT. or PERCENT. format in your data step code.

Note: Microsoft Excel limits for 97, 2000, and 2002 are: columns — 256, rows — 65,536. △

The following table shows the default XLS data types that SAS/ACCESS assigns to SAS variable formats when you write SAS data to an XLS file with the LIBNAME statement.

Table 15.2 Default Excel Formats for SAS Variable Formats

SAS Variable Format	XLS Column Data Type
$BINARY*w*.	Text
$CHAR*w*.	
$HEX*w*.	
$*w*.	
w.*d*	Number
BEST*w*.	
BINARY*w*.	
COMMA*w*.*d*	
COMMAX*w*.*d*	
E*w*.	
FRACT*w*.	
HEX*w*.	
NEGPAREN*w*.*d*	
PERCENT*w*.*d*	
DOLLAR*w*.*d*	Currency
DOLLARX*w*.*d*	
DATE*w*.	Date/Time
DATETIME*w*.*d*	
DDMMYY*w*.	
HHMM*w*.*d*	
JULDAY*w*.	
JULIAN*w*.	
MMDDYY*w*.	
MMYY*w*.*d*	
MONTH*w*.	
MOYY*w*.	
WEEKDATE*w*.	
WEEKDATX*w*.	
WEEKDAY*w*.	
WORDDATE*w*.	
WORDDATX*w*.	

You can override these default conversions by using the LIBNAME option DBTYPE= during output processing.

ACCESS Procedure: XLS Specifics

Chapter 6, "The ACCESS Procedure for PC Files," on page 65 contains general information about this feature. This section provides XLS-specific syntax for the ACCESS procedure and describes ACCESS procedure data conversions.

ACCESS Procedure Syntax for XLS Files

To create an access descriptor, you use the DBMS=XLS option and six database-description statements: PATH=, GETNAMES, RANGE, SCANTYPE, SKIPROWS, and WORKSHEET. These database-description statements supply XLS-specific information to SAS, and must immediately follow the CREATE statement. In addition to the database-description statements, you can use 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 XLS 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 XLS uses the following procedure statements:

PROC ACCESS DBMS=XLS | EXCEL;
  CREATE *libref.member-name.*ACCESS | VIEW;
  UPDATE *libref.member-name.*ACCESS | VIEW;

  GETNAMES <=> YES | NO | Y | N;
  PATH= *'path-and-filename'*<.XLS> '| <'>*filename*<'> | *fileref*;
  RANGE <=> <'>*range-name*<'> | *'range-address'*;
  SCANTYPE <=> YES | NO | Y | N | <*number-of-rows*>;
  SKIPROWS <=> *number-of-rows-to-skip*;
  WORKSHEET <=> *worksheet-name*;
  ASSIGN <=> YES | NO | Y | N ;
  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*<'>> ;
  MIXED <=> YES | NO | Y | N;
  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* ;
  TYPE *column-identifier-1* <=> C | N <… *column-identifier-n* <=> C | N>;
  UNIQUE <=> YES | NO | Y | N ;
RUN;

Note: By default, PROC ACCESS uses Excel 5 files, which have an identical format to Excel 95 files. △

Note: Microsoft Excel 4, 5, and 7 limits are: columns — 256, rows — 16,384. △

The QUIT statement is also available in PROC ACCESS. However, it 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 list provides detailed information about the XLS-specific statements:

GETNAMES <=> YES | NO | Y | N;
determines whether SAS variable names are generated from column names in the first row of the range when an access descriptor is created. When you update a descriptor, you are not allowed to specify the GETNAMES statement.

The GETNAMES statement is optional. If you omit it, the default value GETNAMES=NO is used, and the XLS interface generates the SAS variable names VAR0, VAR1, VAR2, and so on. If you specify GETNAMES=YES, the SAS variable names are generated from the column names in the first row of the range. GETNAMES=YES also sets the SKIPROWS value to 1.

You can change the default value from NO to YES by setting the SS_NAMES environment variable. See "Setting Environment Variables for XLS Files" on page 164 for more information about setting and changing environment variables.

The GETNAMES statement is a database-description statement. It must follow the CREATE statement and precede any editing statements when you create a descriptor.

RANGE <=> <'>*range-name*<'> | '*range-address*';
subsets a specified section of an XLS file worksheet. The *range-name* is the name that is assigned to a range address within the worksheet. Range names can be up to 15 characters long and are not case-sensitive.

The *range-address* is identified by the top left cell that begins the range and the bottom right cell that ends the range within the XLS worksheet file. The beginning and ending cells are separated by two periods; for example, the range address C9..F12 indicates a cell range that begins at cell C9, ends at cell F12 and includes all cells in between.

The RANGE statement is optional. If you omit RANGE, the entire worksheet is accessed as the default range.

The RANGE is a database-description statement. It must follow the CREATE statement and precede any editing statements when you create a descriptor.

SCANTYPE <=> YES | NO | Y | N | <*number-of-rows*>;
finds the most common Excel data type and format for each column in a specified number of rows in an XLS worksheet in order to generate the default SAS format. By default, SAS variable formats are generated from the Excel formats found in the first row of the entire worksheet, or in the first row of a range (if specified) in the worksheet.

The SCANTYPE statement is optional, and its default value is NO. If you specify YES, the ACCESS procedure scans the data types and formats of all the rows in each column of the worksheet or range and uses the most common one to generate the default SAS format for each column. If you specify a number of rows, PROC ACCESS scans the specified number of rows only and returns the most common format.

If you specify the SKIPROWS statement, the ACCESS procedure skips the specified rows and starts scanning from the next row. For example, if you specify SKIPROWS=3, PROC ACCESS skips the first three rows and begins scanning the data type and format on the fourth row.

You can change the default value to YES by setting the SS_SCAN environment variable. See "Setting Environment Variables for XLS Files" on page 164 for more information about setting and changing environment variables.

Specifying SCANTYPE=0 is equivalent to specifying SCANTYPE=NO.

The SCANTYPE statement is a database-description statement. It must follow the CREATE statement and precede any editing statements when you create a descriptor.

SKIPROWS <=> *number-of-rows-to-skip*;
specifies the number rows, beginning at the top of the range in the XLS file, to ignore when you are reading data from the XLS file. The default value for SKIPROWS is 0. The skipped (or ignored) rows often contain information such as column labels or names, or underscores rather than input data.

If GETNAMES=YES, the default value of SKIPROWS automatically changes to 1. The first row of data and formats after SKIPROWS in a range is used to generate the SAS variable types and formats. However, you can use the SCANTYPE statement to scan the formats of a specified number of rows and to use the most common data type and format to generate the default SAS variable types and formats. See "Setting Environment Variables for XLS Files" on page 164 for more information on setting and changing environment variables.

The SKIPROWS statement is a database-description statement. It must follow the CREATE statement and precede any editing statements when you create a descriptor.

WORKSHEET <=> <'>*worksheet-name*<'>;
identifies one worksheet from a group of worksheets while you are reading from an XLS file. The *worksheet-name* is a 31-character name and is not case-sensitive. For example, specifying WORKSHEET=SHEET2 identifies worksheet 2 from a group of worksheets

The WORKSHEET statement is optional. For Excel 4 files, there is only one worksheet identifier, WORKSHEET1. Therefore, the WORKSHEET statement is ignored. Under Excel 5, the default value is SHEET1. If you change the default worksheet from within Excel, you can either supply the new worksheet name or supply the worksheet's value (such as **`Sheet5`**).

The WORKSHEET statement is a database-description statement. It must follow the CREATE statement and precede any editing statements when you create an access descriptor.

ACCESS Procedure Data Conversions for XLS Files

You use PROC ACCESS to define descriptors that identify spreadsheet data and the conversions necessary to use that data in SAS programs. The Microsoft Excel label data type is formatted as a SAS character type, and the Microsoft Excel number data type is formatted as a SAS numeric type.

Fonts, attributes, and colors in the XLS files are not read into the SAS data sets. However, the ACCESS procedure supports most of the XLS number formats and automatically converts them to the corresponding SAS formats. Any XLS data strings longer than 200 characters are truncated while being converted into SAS data sets, and any SAS data file created from XLS files can contain up to 256 variables and 16,384 observations.

The following table shows the default SAS variable formats that the ACCESS procedure assigns to each type of standard XLS file data. Table 15.4 on page 159 provides SAS variable formats for customized XLS format strings. XLS file numeric data includes date and time values. See "Datetime Conversions in the ACCESS Procedure" on page 160 for more information.

Table 15.3 Default SAS Variable Formats for XLS File Data

XLS File Data		SAS Variable Format	
Data Type	XLS Format String	Type	Format
Char[1]	@[2]	Char	$*w.*
Numeric[3]	General	Num	BEST
Numeric	0	Num	*w.d*
Numeric	0.00	Num	*w.d*
Numeric	#,##0	Num	COMMA*w.d*
Numeric	#,##0.00	Num	COMMA*w.d*
Numeric	#,##0_);(#,##0)	Num	NEGPAREN*w.d*
Numeric	#,##0_);[Red](#,##0)	Num	NEGPAREN*w.d*
Numeric	#,##0.00_);(#,##0.00)	Num	NEGPAREN*w.d*
Numeric	#,##0.00_);[Red](#,##0.00)	Num	NEGPAREN*w.d*
Numeric	$#,##0_);($#,##0)	Num	DOLLAR*w.d*
Numeric	$#,##0_);[Red]($#,##0)	Num	DOLLAR*w.d*
Numeric	($#,##0.00_);($#,##0.00)	Num	DOLLAR*w.d*
Numeric	($#,##0.00_);[Red]($#,##0.00)	Num	DOLLAR*w.d*
Numeric	_($*#,##0_);_($*(#,##0);_($*"-"_);_(@_)	Num	DOLLAR*w.d*
Numeric	_(*#,##0_);_(*(#,##0);_(*"-"_);_(@_)	Num	NEGPAREN*w.d*
Numeric	_($*#,##0.00_);_($*(#,##0.00);_($*"-"??_);_(@_)	Num	DOLLAR*w.d*
Numeric	_(*#,##0.00_);_(*(#,##0.00);_(*"-"??_);_(@_)	Num	NEGPAREN*w.d*
Numeric	0%	Num	PERCENT*w.d*
Numeric	0.00%	Num	PERCENT*w.d*
Numeric	0.00E+00	Num	E*w.d*
Numeric	##0.0E+0	Num	E*w.d*
Numeric	*m/d/yy*	Num	MMDDYY*w.*
Numeric	*d-mmm-yy*	Num	MMDDYY*w.*
Numeric	*d-mmm*	Num	DATE*w.*
Numeric	*mmm-yy*	Num	MONYY*w.*
Numeric	*h:mm* AM/PM	Num	TIME*w.*
Numeric	*h:mm:ss* AM/PM	Num	TIME*w.*
Numeric	*h:mm*	Num	TIME*w.*
Numeric	*hh:mm*	Num	TIME*w.*
Numeric	*h:mm:ss*	Num	TIME*w.*
Numeric	*hh:mm:ss*	Num	TIME*w.*

XLS File Data		SAS Variable Format	
Data Type	XLS Format String	Type	Format
Numeric	*m/d/yy h:mm*	Num	DATETIME*w.*
Numeric	*ddmmmyy*	Num	DATE*w.*
Numeric	*ddmmmyyyy:hh:mm:ss*	Num	DATETIME*w.*
Numeric	*dd*	Num	DATE*w.*
Numeric	*dd/mm/yy*	Num	DDMMYY*w.*
Numeric	*dddd*	Num	DATE*w.*
Numeric	*mm/dd/yy*	Num	MMDDYY*w.*
Numeric	*mm:ss*	Num	MMSS*w.*
Numeric	*mm yy*	Num	MONYY*w.*
Numeric	*mm yyyy*	Num	MONYY*w.*
Numeric	*mm:yy*	Num	MONYY*w.*
Numeric	*mm:yyyy*	Num	MONYY*w.*
Numeric	*mm-yy*	Num	MONYY*w.*
Numeric	*mm-yyyy*	Num	MONYY*w.*
Numeric	*mmyy*	Num	MONYY*w.*
Numeric	*mmyyyy*	Num	MONYY*w.*
Numeric	*mm.yy*	Num	MONYY*w.*
Numeric	*mm.yyyy*	Num	MONYY*w.*
Numeric	*mm/yy*	Num	MONYY*w.*
Numeric	*mm/yyyy*	Num	MONYY*w.*
Numeric	*mmmm*	Num	MONYY*w.*
Numeric	*m*	Num	MONYY*w.*
Numeric	*mmmyy*	Num	MONYY*w.*
Numeric	*mmmyyyy*	Num	MONYY*w.*
Numeric	*dddd, mmmm dd, yyyy*	Num	MONYY*w.*
Numeric	*dddd, dd mmmm yyyy*	Num	MONYY*w.*
Numeric	*mmmm dd, yyyy*	Num	MONYY*w.*
Numeric	*dd mmmm yyyy*	Num	MONYY*w.*
Numeric	*yy*	Num	YYMMDD*w.*
Numeric	*yyyy*	Num	YYMMDD*w.*
Numeric	*yy mm*	Num	YYMMDD*w.*
Numeric	*yyyy mm*	Num	YYMMDD*w.*
Numeric	*yy:mm*	Num	YYMMDD*w.*
Numeric	*yyyy:mm*	Num	YYMMDD*w.*
Numeric	*yy-mm*	Num	YYMMDD*w.*
Numeric	*yyyy-mm*	Num	YYMMDD*w.*

XLS File Data		SAS Variable Format	
Data Type	XLS Format String	Type	Format
Numeric	*yymm*	Num	YYMMDD*w.*
Numeric	*yyyymm*	Num	YYMMDD*w.*
Numeric	*yy.mm*	Num	YYMMDD*w.*
Numeric	*yyyy.mm*	Num	YYMMDD*w.*
Numeric	*yy/mm*	Num	YYMMDD*w.*
Numeric	*yyyy/mm*	Num	YYMMDD*w.*
Numeric	*yy-mm-dd*	Num	YYMMDD*w.*
Numeric	*yymmm*	Num	YYMMDD*w.*
Numeric	*yyyymmm*	Num	YYMMDD*w.*

1 Label data.
2 The XLS character format for Excel Version 5.
3 Number, formula, or missing data.

Table 15.4 Default SAS Variable Formats for Customized XLS Format Strings

XLS File Data		SAS Variable Format	
Data Type	XLS Format String	Type	Format
Numeric	"$"	Num	DOLLAR*w.d*
Numeric	"E"	Num	E*w.d*
Numeric	"*m*, *d* and *y*"	Num	MMDDYY*w.*
Numeric	"*m* and *h*"	Num	TIME*w.d*
Numeric	"*m* and *s*"	Num	TIME*w.d*
Numeric	"*m* and *y*"	Num	MONYY*w.*
Numeric	"*m*"	Num	DATE*w.*
Numeric	"*d*"	Num	DATE*w.*
Numeric	"*y*"	Num	DATE*w.*
Numeric	"0.0"	Num	*w.d*
Numeric	Fraction values (#?/?)	Num	BEST*w.d*
Numeric	Percent values (0.0%)	Num	PERCENT*w.d*
Numeric	All others	Num	BEST*w.d*

Note that *w* is based on Excel column width; *.d* is controlled by the Excel format string.

If XLS files data falls outside of the valid SAS data ranges, you receive an error message in the SAS log when you try to access the data.

The SAS/ACCESS interface does not fully support the Microsoft Excel hidden and text formats. XLS data in hidden format is displayed in SAS data sets. However, you can drop the hidden column when you are creating the access descriptor. If you want to display a formula in text format, add a space to indicate that the formula entry is a label. Otherwise, the results of the formula are displayed.

If you have set the SS_MIXED environment variable to YES, the numerical values in XLS files are converted to character strings in SAS data sets if the corresponding SAS variable type is specified as character.

Datetime Conversions in the ACCESS Procedure

An XLS date value is the integer portion of a number that represents the number of days between January 1, 1900 and a specified date. An XLS time value is a decimal portion of a number that represents time as a portion of the day. For example, 0.0 is 12:00:00 a.m., and 0.9999884 is 11:59:59 p.m. While a number can have both a date and a time portion, the formats in XLS display a number only as one or the other. For example, for 1:00 p.m., March 12, 1994, the XLS date value is 34405, the time value is 0.5416667, and the datetime value is 34405.5416667.*

SAS handles date and time values differently than XLS. A SAS date value is an integer that represents the number of days between January 1, 1960 and a specified date. A SAS time value is an integer that represents the number of seconds since midnight of the current day. When a date and a time are both present, SAS stores the value as the number of seconds since midnight, January 1, 1960. For example, for 1:00 p.m., March 12, 1994, the SAS date value is 12,489, and the SAS time value is 46,800. Therefore, the SAS datetime value is 1,079,096,400.

When you create an access descriptor, SAS converts an XLS datetime format to its corresponding SAS datetime format if an XLS datetime format is specified for the variable in the XLS file. Note that if the datetime value does not have an XLS format in the XLS file, SAS treats the datetime value like a numeric value.

To convert an XLS datetime format to a SAS datetime format, you need a SAS datetime format in the access descriptor. For example, changing the default SAS numeric format (15.2) to a SAS date format in the descriptor causes the XLS date value (based on January 1, 1900) to be converted to an equivalent SAS date value (based on January 1, 1960). In other words, the XLS numeric value for January 1, 1960 (which is 21,916) is converted to the equivalent SAS representation of January 1, 1960 (which is 0) only if a SAS datetime format is assigned in the descriptor for that column. Otherwise, the XLS value of 21,916 is treated as a SAS numeric value of 21,916.

The following table shows how SAS uses a Microsoft Excel datetime value to convert to a SAS datetime format.

Table 15.5 Value-to-Format Conversions

For a SAS format	SAS uses
date	integer portion of the Microsoft Excel number
time	decimal portion of the Microsoft Excel number
date-and-time	integer and decimal portion of the Microsoft Excel number

DBLOAD Procedure: XLS Specifics

Chapter 7, "The DBLOAD Procedure for PC Files," on page 91 contains general information about this feature. This section provides XLS-specific syntax for the DBLOAD procedure and describes DBLOAD procedure data conversions.

* In this description, datetime (in lowercase) refers to any value or format that represents a date, a time, or both a date and a time.

DBLOAD Procedure Syntax for XLS Files

To create and load an XLS table, the SAS/ACCESS interface to XLS uses the following statements:

PROC DBLOAD DBMS=XLS | EXCEL <DATA=<*libref.*>*SAS-data-set*>;
PATH=*'path-and-filename*<.XLS>' | <'>*filename*<'> | *fileref*;
VERSION <=> *EXCEL-product-number*;
PUTNAMES <=> YES | NO | Y | N;
ACCDESC= <*libref.*>*access-descriptor*;
DELETE *variable-identifier-1* <…*variable-identifier-n*>;
ERRLIMIT= *error-limit*;
FORMAT *SAS-variable-name-1 SAS-format-1* <=>
<…*SAS-variable-name-n SAS-format-n*>;
LABEL;
LIMIT=*load-limit*;
LIST <ALL | COLUMNS | FIELDS | *variable-identifier*>;
RENAME *variable-identifier-1* <=> <'>*column-name-1*<'>
<…*variable-identifier-n* = <'>*column-name-n*<'>>;
RESET ALL | *variable-identifier-1* <…*variable-identifier-n*>;
WHERE *SAS-where-expression* ;
LOAD ;
RUN ;

The QUIT statement is also available in PROC DBLOAD. However, it 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 list provides detailed information about the XLS-specific statements:

FORMAT *SAS-variable-name-1 SAS-format-1* <…*SAS-variable-name-n SAS-format-n*>;
assigns a temporary format to a SAS variable in the input SAS data set. This format temporarily overrides any other format for the variable. The assignment lasts only for the duration of the procedure. Assign formats to as many variables as you want in one FORMAT statement.

Use FORMAT when you want to change the format, column width, or the number of decimal digits for columns being loaded into the PC file. For example, if you change the SAS variable format 12.1 to DOLLAR15.2, the column format of the loaded data changes from a fixed numeric format with a column width of 12 and one decimal digit to a currency format with a column width of 15 and two decimal digits.

PUTNAMES <=> YES|NO|Y|N;
writes column names to the first row of the new XLS file. The column names can be default SAS variables names or, if you specify the LABEL statement, SAS variable labels. You can modify the column names using the RENAME statement.

The PUTNAMES statement is optional. If you omit PUTNAMES, data is read from the data set and written to the XLS file beginning in the first row of the XLS file, and no column names are written to the file.

You can change the default value to YES by setting the SS_NAMES environment variable. See “Setting Environment Variables for XLS Files” on page 164 for more information on setting and changing environment variables.

VERSION <=> *Excel-product-number*;
specifies the version number of the Excel product you are using, such as Excel 5. The *Excel-product-number* argument can be one of the following values: 5, 95, 97, 2000, or 2002.

The DBLOAD procedure chooses the default version of Excel depending on which operating environment you use. If you use Windows, DBLOAD uses Excel. Excel 5 files have the identical format to Excel 95 files.

PROC DBLOAD does not support Excel 97 files. For information about accessing these files, see "Import/Export Overview for PC Files" on page 49.

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

DBLOAD Procedure Data Conversions for XLS Files

This section explains how SAS data is read into Microsoft Excel data when a table is loaded. In this conversion, the SAS character data type is converted into the Microsoft Excel label type and the SAS numeric type is converted into the Microsoft Excel number type.

The SAS/ACCESS interface automatically converts SAS formats to the same or associated Microsoft Excel formats and column widths. However, you can temporarily assign other formats and column widths to SAS variables by using the FORMAT statement so that the loaded XLS file columns have the formats you want. The following table shows SAS variable types and formats and the XLS data types, formats, and column widths to which you can assign them.

Note: The FORMAT statement in PROC DBLOAD only changes the format of SAS variables while you are creating and loading the XLS files. When the procedure is completed, the formats of SAS variables return to their original settings. △

XLS date and time values are numeric data. See "Datetime Conversions in the DBLOAD Procedure" on page 164 for more information.

Table 15.6 Converting SAS Variable Formats to XLS File Data

SAS Variable Format		XLS File Data	
Type	Format	XLS Format String	Data Type
Char	" "	General	LABEL
Char	$CHAR	General	LABEL
Char	$	General	LABEL
Num	BEST*w.d*	General	NUMBER
Num	COMMA*w.d*	#,##0	NUMBER
Num	COMMAX*w.d*	#,##0	NUMBER
Num	DATE*w.*	*ddmmmyy*	NUMBER
Num	DATETIME*w.d*	*ddmmmyyyy:hh:mm:ss*	NUMBER
Num	DAY*w.*	dd	NUMBER
Num	DDMMYY*w.*	*dd/mm/yy*	NUMBER
Num	DOLLAR*w.d*	"$"#,##0_);("$"#,##0)	NUMBER
Num	DOLLARX*w.d*	"$"#,##0_);("$"#,##0)	NUMBER

SAS Variable Format		XLS File Data	
Type	Format	XLS Format String	Data Type
Num	DOWNAME*w.d*	*dddd*	NUMBER
Num	E*w.*	0.00E+00	NUMBER
Num	HHMM*w.d*	*h:mm*	NUMBER
Num	HOUR*w.d*	*h:mm*	NUMBER
Num	JULDAY*w.*	*m/d/yy*	NUMBER
Num	JULIAN*w.*	*m/d/yy*	NUMBER
Num	MMDDYY*w.*	*mm/dd/yy*	NUMBER
Num	MMSS*w.d*	*mm:ss*	NUMBER
Num	MMYY*xw.*	*mm yy*	NUMBER
Num	MMYYC	*mm:yy*	NUMBER
Num	MMYYD	*mm-yy*	NUMBER
Num	MMYYN	*mmyy*	NUMBER
Num	MMYYP	*mm.yy*	NUMBER
Num	MMYYS	*mm/yy*	NUMBER
Num	MONNAME*w.*	*mmmm*	NUMBER
Num	MONTH*w.*	*m*	NUMBER
Num	MONYY*w.*	*mmmyy*	NUMBER
Num	NEGPAREN*w.d*	#,##0_);(#,##0)	NUMBER
Num	NENGO*w.*	*m/d/yy*	NUMBER
Num	PERCENT*w.d*	0%	NUMBER
Num	QTR*w.*	*m/d/yy*	NUMBER
Num	QTRR*w.*	*m/d/yy*	NUMBER
Num	SSN*w.*	000-00-0000	NUMBER
Num	TIME*w.d*	*h:mm:ss*	NUMBER
Num	TOD*w.*	*h:mm:ss*	NUMBER
Num	W	0	NUMBER
Num	WEEKDATE*w.*	*dddd, mmmm dd, yyyy*	NUMBER
Num	WEEKDATX*w.*	*dddd, dd mmmm yyyy*	NUMBER
Num	WEEKDAY*w.*	m/d/yy	NUMBER
Num	WORDDATE*w.*	*mmmmdd, yyyy*	NUMBER
Num	WORDDATX*w.*	*dd mmmm yyyy*	NUMBER
Num	YEAR*w.*	*yy* or *yyyy*	NUMBER
Num	YYMM	*yy mm*	NUMBER
Num	YYMMC	*yy:mm*	NUMBER
Num	YYMMD	*yy-mm*	NUMBER
Num	YYMMN	*yymm*	NUMBER

SAS Variable Format		XLS File Data	
Type	Format	XLS Format String	Data Type
Num	YYMMP	*yy.mm*	NUMBER
Num	YYMMS	*yy/mm*	NUMBER
Num	YYMMDD*w.*	*yy-mm-dd*	NUMBER
Num	YYMON*w.*	*yymmm*	NUMBER
Num	Z*w.d*	0*w.d*	NUMBER
Num	FRACT*w.*	# ?/?	NUMBER

Note that Excel column widths are set to *w* and displayed in the column. If the data is larger than column width, it is displayed as pound signs (###), in which case it can be viewed by adjusting the column width.

Datetime Conversions in the DBLOAD Procedure

If a SAS variable is specified with a date, time, or datetime format in the FORMAT statement, the interface view engine converts that SAS datetime format into the equivalent Microsoft Excel datetime format when the new XLS file is created.

However, if a SAS datetime format is not specified in the input SAS data set, you have to assign a format by using a PROC DBLOAD FORMAT statement. Doing so assigns a Microsoft Excel datetime format to the SAS variable when the variable is loaded into an XLS file. If you do not assign a SAS datetime format, the SAS numeric value for the date is written to the XLS file. Because SAS dates are based on January 1, 1960, and Microsoft Excel dates are based on January 1, 1900, the date value in the XLS file will be inaccurate.

To maintain a SAS variable format in the input data set, yet change it just while the DBLOAD procedure is in progress, use the FORMAT statement in PROC DBLOAD. This statement enables you to assign a temporary format to a SAS variable for the duration of the procedure without affecting the input SAS data set.

For example, if the SAS format for the BirthDat variable in the MyData.SasEmps access descriptor is left at the default 15.2 format, you can specify the FORMAT statement to change the variable's format to DATE7. while you are creating and loading the XLS file. When you load the XLS file, the DATE7. format becomes an equivalent Microsoft column format, DDMMMYY. When the DBLOAD procedure has completed, the SAS format for the BirthDat variable returns to the 15.2 format.

You can specify the FORMAT statement in the PROC DBLOAD statement when you invoke the procedure using any of the methods of processing.

Setting Environment Variables for XLS Files

You can change the default behavior of PROC ACCESS/PROC DBLOAD by setting environment variables in your SAS configuration file. You can set three SAS/ACCESS environment variables: SS_MIXED, SS_NAMES, and SS_SCAN. Setting these variables in your SAS configuration file changes how the interface works by default.

The configuration file omits these three environment variables, which means their default values are NO.

SS_MIXED YES | NO
: YES allows both Microsoft Excel numeric and character data in a column to be displayed as SAS character data. The Microsoft Excel numeric data is converted to

its character representation when its corresponding SAS variable type is defined as character.

NO does not convert Microsoft Excel numeric data in a column into SAS character data. Microsoft Excel numeric data is read in as SAS missing values when its corresponding SAS variable type is defined as character. NO is the default.

Setting the SS_MIXED environment variable changes the default value of the MIXED statement in PROC ACCESS.

SS_NAMES YES | NO
YES in PROC ACCESS generates SAS variable names from column names in the first row of the worksheet or the specified range of the worksheet and reads data from the second row. YES in PROC DBLOAD writes column names using SAS variable names or SAS variable labels to the first row of the new XLS file, reads the data from the data set, and writes it to the XLS file beginning with the second row.

NO in PROC ACCESS generates the SAS variable names VAR0, VAR1, VAR2, and so on, and reads data from the first row of the worksheet or specified range. NO in PROC DBLOAD reads the data from the data set and writes it to the XLS file beginning with the first row. NO is the default.

Setting the SS_NAMES environment variable changes the default value of the GETNAMES statement in PROC ACCESS and the PUTNAMES statement in PROC DBLOAD.

SS_SCAN YES | NO | *number-of-rows*
YES scans the data type and format of rows in a worksheet or specified range after skipping the number of rows specified in the SKIPROWS statement. After scanning the rows, SS_SCAN finds the most common Microsoft Excel data type and format in order to generate the default SAS data type and format. If a number of rows is specified, SAS/ACCESS software scans the data type and format only from these rows.

NO uses the type and format of the first row in a worksheet or specified range, after skipping the number of rows specified in SKIPROWS, to generate the default SAS data type and format. NO is the default.

Number-of-rows scans the type and format of the specified number of rows only. Setting the number of rows is more efficient because data is read only from the specified number of rows rather than from the entire file.

Setting the SS_SCAN environment variable changes the default value of the SCANTYPE statement in PROC ACCESS.

XLS Essentials

SAS/ACCESS software for PC files works with Microsoft Excel 4 and Excel 5 files, which are are referred to collectively throughout this document as XLS files. You can also access Excel 7, 97, 2000, or 2002 data by using the Import/Export wizard and procedures, LIBNAME engine, and PROC SQL under Windows operating environments.

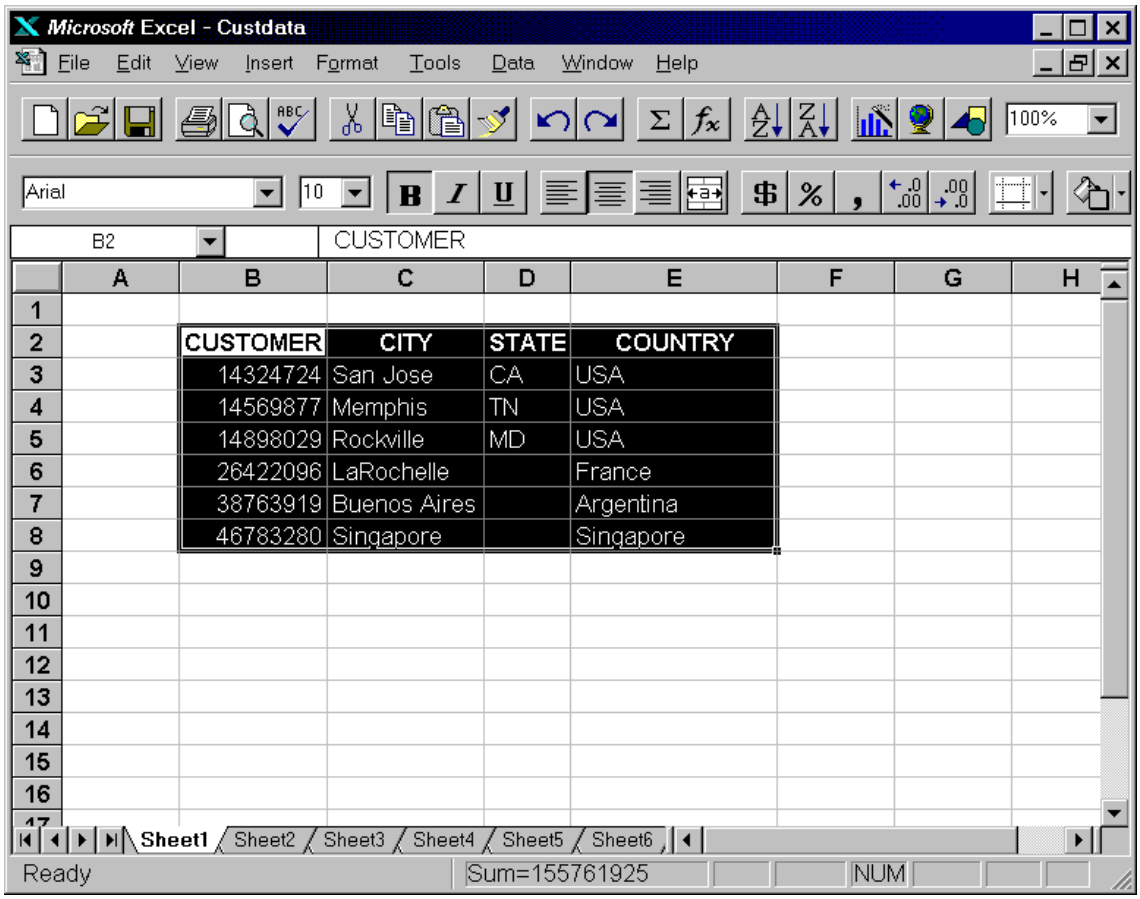
XLS Files

Various software products, such as the Microsoft Excel spreadsheet, enable you to use spreadsheet or database files to enter, organize, and perform calculations on data. Spreadsheets are most often used for general ledgers, income statements, and other types of financial record keeping. Database files also enable you to organize related information, such as the data in an accounts-receivable journal.

In spreadsheets, the data is organized according to certain relationships among data items. These relationships are expressed in a tabular format — in columns and rows. Each *column* represents one category of data, and each *row* can hold one data value for each column.

A Microsoft Excel 5 worksheet, for example, is an electronic spreadsheet consisting of a grid of 256 columns and 16,384 rows. The intersection of a column and a row is called a *cell*. The following display illustrates a portion of a standard Excel worksheet.

Display 15.1 Columns and Rows of Data in an XLS File



Column letters for each column appear above the worksheet. Columns are lettered A through IV (A to Z, AA to AZ, BA to BZ, and so on to IV). Row numbers for each row appear to the left of the worksheet. Rows are numbered 1 to 16,384. For Excel 4 files, only one worksheet (worksheet 1) is allowed per file, but more than one worksheet can be stored in a workbook. You must convert any worksheets you store in a workbook back to worksheets before you can use the data in a SAS program.

A *range* is a subset of cells in a worksheet. A range is identified by its address, which begins with the name of the top left cell and ends with the name of the bottom right cell separated by two periods. For example, the range B2..E8 is the range address for a rectangular block of 12 cells whose top left cell is B2 and whose bottom right cell is E8 (as shaded in the display).

XLS File Naming Conventions

The following conventions apply to XLS filenames. Filenames must also follow operating-system specific conventions, so check the documentation that comes with your Microsoft Excel product or other software products for further information.

- Under Windows 95, 98, NT, 2000, and XP the ACCESS and DBLOAD procedures support long names that are specified in the PATH= statement (such as **path= 'c:\sasdemo\library\new_customer_1999.xls';**). However, XLS files with paths longer than 64 characters might not accepted by some versions of Microsoft Excel.
- Filenames start with a letter, and they can contain any combination of the letters A through Z, the digits 0 through 9, the underscore (_), the hyphen (-), and spaces (blanks) within filenames.
- Filenames can contain spaces. Filenames that contain spaces or lowercase letters are supported by the ACCESS and DBLOAD procedures, but they might not be accepted by some versions of Microsoft Excel.

XLS Data Types

Microsoft Excel software has two data types: character and numeric. Microsoft Excel character data can be entered as labels or formula strings; Microsoft Excel numeric data can be entered as numbers or formulas.

Character data is generally considered text and can include dates and numbers.

Numeric data can include numbers (0 through 9), formulas, and cell entries that begin with one of the following symbols: +, $, @, –, =, or #. When you create and load an Excel file with PROC DBLOAD, the SAS/ACCESS engine supplies **#NA** for a missing, numeric value.

Numeric data also can include date and time values. In Microsoft Excel software, a *date value* is the integer portion of a number that can range from 01 January 1900 to 31 December 2078, that is, 1 to 65,380. A Microsoft Excel software *time value* is the decimal portion of a number that represents time as a proportion of a day. For example, 0.0 is midnight, 0.5 is noon, and 0.999988 is 23:59:59 (on a 24-hour clock). While a number can have both a date and a time portion, the formats in Microsoft Excel display a number only in a date, time, or datetime format. The conversion of date and time values between SAS data sets and Microsoft Excel spreadsheets is transparent to users. However, you are encouraged to understand the differences between them. For information about how the SAS/ACCESS interface handles date and time values and formats, see “Datetime Conversions in the ACCESS Procedure” on page 160 and “Datetime Conversions in the DBLOAD Procedure” on page 164.

When you create an access descriptor, the interface software uses the column types and formats in the XLS file to determine the corresponding SAS variable formats. SAS generates its default formats based on the values that you specify for the SCANTYPE, SKIPROWS, and GETNAMES statements (or in the corresponding fields in the Access Descriptor Identification window). You can change the formats generated by the software interface. For more information, see “How SAS/ACCESS Works with XLS Files” on page 168.

When you create an access descriptor, any data value that does not match the column type (character or numeric) is treated as a missing value. This is the default action. However, you can use the MIXED=YES statement to convert numeric data values in a character column to their character representation.

You can also set the SS_MIXED environment variable to **YES** in your SAS configuration file so that both numeric and character data are displayed as SAS character data. Add this line to your SAS configuration file:

```
-SET SS_MIXED YES
```

See “Setting Environment Variables for XLS Files” on page 164 for more information about environment variables. For more information about changing the column type from the type determined by SAS/ACCESS software when you create an access descriptor, refer to the “DBLOAD Procedure: XLS Specifics” on page 160.

How SAS/ACCESS Works with XLS Files

The SAS/ACCESS interface accesses data in the Microsoft Excel XLS files directly. It enables you to create SAS data sets from XLS files or directly read the XLS file data without creating SAS data sets. The interface does not allow you to update, add, or delete data in XLS files.

Accessing the Data

To access the data, the interface accesses a range in a worksheet as a table. If the range is not specified, the interface accesses the entire worksheet as a table. By default, the interface uses the Microsoft Excel formats of columns in the first row of the range to determine the formats of variables in SAS/ACCESS descriptors.

However, you can manipulate where the interface begins to read data and what format the interface generates by using the SKIPROWS and SCANTYPE statements in the ACCESS procedure. SKIPROWS skips a specified number of rows before reading data. SCANTYPE finds the most common data type and format from among a specified number of rows within an XLS range (after skipping the number of rows specified in SKIPROWS) and uses it to generate the default data type and format for SAS variables.

The ACCESS procedure enables you to create access descriptors and view descriptors for XLS files. You then can use the view descriptors as SAS data sets.

You can retrieve a subset of data using the WHERE statement.

To sort XLS file data, you must first extract the data from an XLS file and place it in a SAS data file, unless you are using the SQL procedure. (The SQL procedure enables you to present output data in a sorted order using the ORDER BY clause of the SELECT statement.) You can extract and sort XLS file data in one step with the OUT= option in the SORT procedure, using a view to the XLS file as input to PROC SORT.

Creating and Loading the Data

When you use PROC DBLOAD to create and load XLS files, the procedure translates the SAS data set into an XLS file. The file is stored in the location specified by the PATH= statement. Only one SAS data set can be loaded into an XLS file at one time. The loaded XLS file can contain only one worksheet. Microsoft Excel then reads data from the loaded XLS file directly.

In the DBLOAD procedure, you can specify the PUTNAMES statement to place the SAS variable names in the first row of the spreadsheet and the first observation in the second row, and so on. If PUTNAMES is not specified, the first observation is placed in the first row, the second observation is placed in the second row, and so on. Columns do not have names. The formats for SAS variables are automatically converted to the closest corresponding Microsoft Excel data types and formats. See the descriptions of individual statements for more information about how the data and columns are read.

CHAPTER
16
Microsoft Access MDB Files

How to Access MDB Files from SAS

You can interact with Microsoft Access MDB files from SAS by using the following features:

LIBNAME statement (UNIX and Windows operating environments)
provides direct, transparent access to data in PC file formats. Available for Microsoft Access 97, 2000, or 2002 formats. For details, refer to Chapter 2, "The LIBNAME Statement for PC Files on Windows," on page 5.

Pass-Through Facility (UNIX and Windows operating environments)
enables you to interact with Microsoft Access (97 or 2000) data using the data source's SQL syntax without leaving your SAS session. The SQL statements are passed directly to the data source for processing. For details, refer to Chapter 3, "The Pass-Through Facility for PC Files on Windows," on page 35.

Import/Export wizard or procedures (UNIX and Windows operating environments)
enable you to transfer data between SAS and several PC file formats. Available for Microsoft Access 97, 2000, or 2002 formats. For details, refer to Chapter 4, "The Import/Export Wizard and Procedures," on page 49.

LIBNAME Statement Data Conversions for MDB Files

The following table shows the default SAS variable formats that SAS/ACCESS assigns to MDB data types when you read or import MDB data with the LIBNAME statement.

Table 16.1 Default SAS Variable Formats for MDB Data

MDB Field Data Type	SAS Variable Format	SAS Variable Type
YES|NO	2.	numeric
Number (FieldSize=Byte)	4.	numeric
Number (FieldSize=Integer)	6.	numeric
Number (FieldSize=Long Integer)	11.	numeric
Number (FieldSize=Single)		numeric
Number (FieldSize=Double)		numeric
AutoNumber (FieldSize=Long Integer)	11.	numeric
AutoNumber (FieldSize=Replication ID)	$38.	character
CURRENCY	DOLLAR21.2	numeric
Date/Time	DATE9. See Notes 1 and 2	numeric
Text	$*w*. See Note 3	character
Memo	$*w*. See Note 4	character
OLE Object	$*w*. See Note 4	character
Hyperlink	$*w*. See Note 4	character

1 The default format is DATE9. However, you can use the SASDATEFMT option to change the format to other date or datetime formats. The engine automatically converts the internal date value for you.

2 If you have a time only field in your Microsoft Access range, you can use SASDATEFMT to assign it with the SAS TIME. format. Note that the SAS date/time value uses 01Jan1960 as the cutoff date, while the Jet provider date/time value uses 30Dec1899 as the cutoff date.

3 The width of $*w*. is equal to the field size of the column defined in your Access table.

4 When the option SCAN_TEXT=YES (which is the default value), the width value of $*w*. is determined by the longest string of data that is scanned in the field or by the value specified in the DBMAX_TEXT option, whichever is less. Otherwise, when the option SCAN_TEXT=NO, the width value of $*w*. is equal to the value specified in DBMAX_TEXT option.

The following table shows the default MDB data types that SAS/ACCESS assigns to SAS variable formats when you write SAS data to an MDB file with the LIBNAME statement.

Table 16.2 Default MDB Data Types for SAS Variable Formats

SAS Variable Format	MDB Data Type
$BINARY*w*.	
$CHAR*w*.	Text (VarChar> or Memo (LongText)
HEX*w*.	See Note 2
$*w*.	

SAS Variable Format	MDB Data Type
w.d	
BEST*w*.	
BINARY*w*.	
COMMA*w.d*	
COMMAX*w.d*	Number.
E*w*.	See Notes 3 and 4.
FRACT*w*.	
HEX*w*.	
NEGPAREN*w.d*	
PERCENT*w.d*	
DOLLAR*w.d*	Currency
DOLLARX*w.d*	
DATE*w*.	
DATETIME*w.d*	
DDMMYY*w*.	
HHMM*w.d*	
JULDAY*w*.	
JULIAN*w*.	
MMDDYY*w*.	
MMYY*w.d*	Date/Time
MONTH*w*.	
MOYY*w*.	
WEEKDATE*w*.	
WEEKDATX*w*.	
WEEKDAY*w*.	
WORDDATE*w*.	
WORDDATX*w*.	

1 You can use the data set option DBTYPE= to override the default data types. For valid data types supported, please refer to the valid data types list.

2 If the character format length is greater than 255 characters, the loaded format is Memo; otherwise, the loaded format is Text.

3 For Access 2000 and 2002, a SAS numeric data type with no format specified is converted to a number data type with a double field size. If the format is specified as *w*. in SAS, the loaded data type in Access is a number data type with an integer field size. If the format is specified as *w.d* in SAS, the loaded data type in Access is a number data type with a decimal field size.

4 For Access 97, if the format is specified as *w*. in SAS, the loaded data type in Access is a number data type with an integer field size. Otherwise, the SAS numeric data type is converted to a number data type with a double field size.

MDB Essentials

This section introduces SAS users to MDB files. It focuses on the terms and concepts that help you use the SAS/ACCESS interface and includes descriptions of MDB files, MDB naming conventions, and MDB data types.

SAS/ACCESS software for PC files works with Microsoft Access MDB 97, 2000, and 2002 files, which are referred to collectively throughout this document as MDB files.

MDB Files

Microsoft Access is a desktop relational database management system (DBMS) that uses the Jet engine to store and retrieve data. All of the objects in a Microsoft Access MDB-type database (including tables, indexes, forms, and reports) are stored in Jet's native MDB file format.

MDB Naming Conventions

The following conventions apply to MDB filenames. Filenames must also follow operating system specific conventions, so check the documentation that comes with your Microsoft Access product or other software products for further information.

- The filename can be up to 255 characters, including spaces.
- Names of Microsoft Access objects can be up to 64 characters long.
- Names of Microsoft Access objects can be composed of any combination of letters, numbers, spaces, and special characters except for the period (.), exclamation point (!), accent grave (‘), square brackets ([ ]), and quotation mark (“).
- Names of Microsoft Access objects cannot start with spaces or control characters (ASCII characters 0 through 31).

MDB Data Types

The following table lists the valid data types supported by Jet provider. You may use these data types when you use the CREATE statement in SQL Pass-Through to create a table in your Microsoft ACCESS database.

Data Type	Column Size	Create Params	Prefix/ Suffix	Comments
BIT	2			
BYTE	3			
SHORT	5			
LONG	10			
SINGLE	7			
DOUBLE	15			
DECIMAL	28	precision, scale		See Note 6.
COUNTER	10			See Notes 2 and 3.
				See Notes 2 and 3.
GUID	16			

Data Type	Column Size	Create Params	Prefix/ Suffix	Comments
CURRENCY	19			
DATETIME	8		#.	See Note 5.
VARCHAR	255	max length		See note 7.
LONGTEXT	536,870,910			See Note 8.
VARBINARY	255	max length		See Note 7.
BIGBINARY	4000			
LONGBINARY	1,073,741,823			See Note 8.

1 Always use the data type listed above when you use data set option DBTYPE= to change the data type for a loaded column. Do not use the synonyms.
2 When using the option DBTYPE=, the data type COUNTER is only valid when you set INSERT_SQL=YES.
3 The data type COUNTER is only supported in the Pass-Through Facility.
4 A column with the BIT data type is not nullable.
5 When using the Pass-Through Facility to set a datetime value, you need to add the prefix and suffix, #. For example, #01/01/2001#.
6 When using the data type DECIMAL, you can specify the precision and scale.
7 When using the data types VARCHAR or VARBINARY, you need to specify the maximum length.
8 When using the data types LONGTEXT or LONGBINARY, you do not need to specify the maximum length.

How SAS/ACCESS Works with MDB Files

The SAS/ACCESS interface accesses data in Microsoft Access MDB files directly. It enables you to create SAS data sets from MDB files or directly read or update the MDB file data without creating SAS data sets.

The SAS/ACCESS LIBNAME engine interacts with MDB files via the Microsoft Jet database engine, which manages data that resides in Microsoft Access MDB-type databases.

To sort MDB file data, you must first extract the data from an MDB file and place it in a SAS data file, unless you are using the SQL procedure. (The SQL procedure enables you to present output data in a sorted order using the ORDER BY clause of the SELECT statement.) You can extract and sort MDB file data in one step with the OUT= option in the SORT procedure, using a view to the MDB file as input to PROC SORT.

CHAPTER
17

Lotus WKn Files

How To Access WK*n* Files from SAS

You can interact with data in the form of Lotus 1-2-3 spreadsheets (WK1, WK3, or WK4 files) from SAS by using the following features:

Import/Export wizard or procedures (Windows operating environments)
enable you to transfer data between SAS and several PC file formats.

ACCESS procedure (Windows operating environments)
creates descriptor files that describe data in a PC file to SAS, enabling you to directly read, update, or extract PC files data into a SAS data file.

DBLOAD procedure (Windows operating environments)
creates PC files and loads them with data from a SAS data set.

This section contains WK*n*-specific information for the ACCESS and DBLOAD procedures. See Chapter 4, “The Import/Export Wizard and Procedures,” on page 49 for information about those features.

ACCESS Procedure: WK*n* Specifics

Chapter 6, "The ACCESS Procedure for PC Files," on page 65 contains general information about this feature. This section provides WK*n*-specific syntax for the ACCESS procedure and describes ACCESS procedure data conversions.

ACCESS Procedure Syntax for WK*n* Files

To create an access descriptor, you use the DBMS=WK*n* option and six database-description statements: PATH=, GETNAMES, RANGE, SCANTYPE, SKIPROWS, and WORKSHEET. These database-description statements supply WK*n*-specific information to SAS and must immediately follow the CREATE or UPDATE statement that specifies the access descriptor to be created or updated. In addition to the database-description statements, you can use 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 WK*n* 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 WK*n* uses the following procedure statements:

PROC ACCESS DBMS=WK1|WK3|WK4;
CREATE *libref.member-name.*ACCESS | VIEW;
UPDATE *libref.member-name.*ACCESS | VIEW;

PATH= *'path-and-filename*<.WK1 | .WK3 | .WK4>'| <'>*filename*<'>| *fileref*;
GETNAMES <=> YES | NO | Y | N;
RANGE <=><'>*range-name*<'>| *'range-address*';
SCANTYPE <=> YES | NO | Y | N | <*number-of-rows*>;
SKIPROWS <=> *number-of-rows-to-skip*;
WORKSHEET <=> *worksheet-letter*| <'>*worksheet-name*<'>;
ASSIGN <=> YES | NO | Y | N;
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*<'>>;
MIXED <=> YES | NO | Y | N;
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*;
TYPE *column-identifier-1*<=> C | N <…*column-identifier-n* <=> C | N>;
UNIQUE <=> YES | NO | Y | N ;

RUN ;

The QUIT statement is also available in PROC ACCESS. However, it 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 list provides detailed information about the WK*n*-specific statements:

GETNAMES <=> YES | NO | Y | N;
determines whether SAS variable names are generated from column names in the first row of the Lotus range when an access descriptor is created. When you update a descriptor, you are not allowed to specify the GETNAMES statement.

The GETNAMES statement is optional. If you omit it, the default value GETNAMES=NO is used, and the SAS/ACCESS interface generates the SAS variable names VAR0, VAR1, VAR2, and so on. If you specify GETNAMES=YES, the SAS variable names are generated from the column names in the first row of the Lotus range. GETNAMES=YES also sets the default value of SKIPROWS to 1.

You can change the default value from NO to YES by setting the SS_NAMES environment variable. See "Setting Environment Variables for WK*n* Files" on page 182 for more information about setting and changing environment variables.

The GETNAMES statement is a database-description statement. It must follow the CREATE statement and precede any editing statements when you create a descriptor.

RANGE <=> <'>*range-name*<'> | '*range-address*';
subsets a specified section of a WK*n* file worksheet. The *range-name* is the name that is assigned to a range address within the worksheet. Range names can be up to 15 characters long and are not case-sensitive. If you specify a range name, the name must have been previously defined in the WK*n* file. The *range-address* is identified by the top left cell that begins the range and the bottom right cell that ends the range within the WK*n* worksheet file. The beginning and ending cells are separated by two periods. For example, the range address C9..F12 indicates a cell range that begins at cell C9, ends at cell F12, and includes all cells in between.

The RANGE statement is optional. If you omit RANGE, the entire worksheet is accessed as the default range.

The RANGE is a database-description statement. It must follow the CREATE statement and precede any editing statements when you create a descriptor.

SCANTYPE <=> YES | NO | Y | N | <*number-of-rows*>;
finds the most common Lotus 1-2-3 format for each column in a specified number of rows in an WK*n* worksheet to generate the SAS format. By default, SAS variable formats are generated from the Lotus 1-2-3 formats found in the first row of the worksheet, or in the range of the worksheet if you specified a range.

The SCANTYPE statement is optional, and its default value is NO. If you specify YES, the ACCESS procedure scans the Lotus 1-2-3 formats of all the rows in each column of the range and uses the most common format to generate the default SAS format for each column. If you specify a number of rows, PROC ACCESS scans the specified number of rows only and returns the most common format.

If you specify the SKIPROWS statement, the ACCESS procedure skips the specified rows and starts scanning the Lotus 1-2-3 format from the next row. For example, if you specify SKIPROWS=3, PROC ACCESS skips the first three rows and begins scanning the formats on the fourth row.

You can change the default value to YES by setting the SS_SCAN environment variable. See "Setting Environment Variables for WK*n* Files" on page 182 for more information about setting and changing environment variables.

Specifying SCANTYPE=0 is equivalent to specifying SCANTYPE=NO.

The SCANTYPE statement is a database-description statement. It must follow the CREATE statement and precede any editing statements when you create a descriptor.

SKIPROWS <=> *number-of-rows-to-skip*;
specifies the number of rows, beginning at the top of the range in the WK*n* file, to ignore when you are reading data from the WK*n* file. The default value for

SKIPROWS is 0. The skipped (or ignored) rows often contain information such as column labels or names, or underscores rather than input data.

If GETNAMES=YES, the default value of SKIPROWS automatically changes to 1. The first row of data and formats after SKIPROWS in a range is used to generate the SAS variable types and formats. However, you can use the SCANTYPE statement to scan the formats of specified rows and use the most common type and format to generate the default SAS variable types and formats.

The SKIPROWS statement is a database-description statement. It must follow the CREATE statement and precede any editing statements when you create a descriptor.

WORKSHEET <=> *worksheet-letter* | <'>*worksheet-name*<'>;
identifies a particular worksheet when you are reading from a WK*n* file that contains more than one worksheet. You can specify a worksheet name or a worksheet letter using the WORKSHEET statement. Worksheet names can be up to 15 characters long and are not case-sensitive. A worksheet letter is a one- or two-letter alpha character. For WK1 files, there is only one worksheet letter: worksheet A. For WK3 and WK4 files, there can be up to 256 different worksheet letters: worksheet A through worksheet Z and worksheet AA through worksheet IV. The default value is A. For example, specifying WORKSHEET=B identifies worksheet B from a group of worksheets.

The WORKSHEET statement is optional. The WORKSHEET statement is a database-description statement. It must follow the CREATE statement and precede any editing statements when you create an access descriptor.

ACCESS Procedure Data Conversions for WK*n* Files

You use PROC ACCESS to define descriptors that identify spreadsheet data and the conversions necessary to use that data in SAS programs. The Lotus 1-2-3 label data type is formatted as a SAS character type, and the Lotus 1-2-3 number data type is formatted as a SAS numeric type.

Fonts, attributes, and colors in the WK*n* files are not read into the SAS data sets. However, the ACCESS procedure supports most of the WK*n* number formats and automatically converts them to the corresponding SAS formats. Any WK*n* data strings longer than 200 characters are truncated while being converted into SAS data sets, and any SAS data file created from WK*n* files can only contain up to 256 variables and 8,192 observations.

Table 17.2 on page 181 shows the default SAS variable formats that the ACCESS procedure assigns to each type of WK*n* file data. WK*n* numeric data includes date and time values. See "Datetime Conversions in the ACCESS Procedure" on page 179 for more information.

If WK*n* file data falls outside of the valid SAS data ranges, you receive an error message in the SAS log when you try to access the data.

The SAS/ACCESS interface does not fully support the Lotus 1-2-3 hidden and text formats. WK*n* data in hidden format is displayed in SAS data sets. However, you can drop the hidden column when you are creating the access descriptor. If you want to display a formula in text format, add a label prefix character to indicate that the formula entry is a label. Otherwise, the results of the formula are displayed.

If you have set the SS_MIXED environment variable to **YES**, the numerical values in WK*n* files are converted to character strings in SAS data sets if the corresponding SAS variable type is specified as character.

Datetime Conversions in the ACCESS Procedure

A Lotus 1-2-3 date value is the integer portion of a number that represents the number of days between January 1, 1900 and a specified date. A Lotus 1-2-3 time value is a decimal portion of a number that represents time as a portion of the day. For example, 0.0 is 12:00:00 a.m. and 0.9999884 is 11:59:59 p.m. While a number can have both a date and a time portion, the formats in Lotus 1-2-3 display a number only in a date format or in a time format. For example, for 1:00 p.m., March 12, 1994, the Lotus 1-2-3 date value is 34405, the time value is 0.5416667, and the datetime value is 34405.5416667.

SAS handles date and time values differently than Lotus 1-2-3 . A SAS date value is an integer that represents the number of days between January 1, 1960 and a specified date. A SAS time value is an integer that represents the number of seconds since midnight of the current day. When a date and a time are both present, SAS stores the value as the number of seconds since midnight, January 1, 1960. For example, for 1:00 p.m., March 12, 1994, the SAS date value is 12489, and the SAS time value is 46800. Therefore, the SAS datetime value is 1079096400.*

To convert a Lotus 1-2-3 datetime format to a SAS datetime format, you need a SAS datetime format in the view descriptor. For example, changing the default SAS numeric format (15.2) to a SAS date format in the descriptor causes the Lotus 1-2-3 date value (based on January 1, 1900) to be converted to an equivalent SAS date value (based on January 1, 1960). In other words, the Lotus 1-2-3 numeric value for January 1, 1960 (which is 21916) is converted to the equivalent SAS representation of January 1, 1960 (which is 0) only if a SAS datetime format is assigned in the descriptor for that column. Otherwise, the Lotus 1-2-3 value of 21916 is treated as a SAS numeric value of 21916.

The table below shows how SAS uses a Lotus 1-2-3 internal datetime value to convert to a SAS internal datetime value.

Table 17.1 Value-to-Format Conversions

For a SAS format	SAS uses
date	if the Lotus datetime value is less than 60: integer portion of the Lotus 1-2-3 datetime value minus 21915
	if the Lotus datetime value is greater than 60: integer portion of the Lotus 1-2-3 datetime value minus 21916
time	decimal portion of the Lotus 1-2-3 datetime value times 86400
date-and-time	if the Lotus datetime value is less than 60: (integer and decimal portion of the Lotus 1-2-3 datetime value minus 21915) times 86400
	if the Lotus datetime value is greater than 60: integer and decimal portion of the Lotus 1-2-3 datetime value minus 21916 times 86400

DBLOAD Procedure: WK*n* Specifics

Chapter 7, "The DBLOAD Procedure for PC Files," on page 91 contains general information about this feature. This section provides WK*n*-specific syntax for the DBLOAD procedure and describes DBLOAD procedure data conversions.

* In this description, datetime (in lowercase) refers to any value or format that represents a date, a time, or both a date and time.

DBLOAD Procedure Syntax for WK*n* Files

To create and load a WK*n* table, the SAS/ACCESS interface to WK*n* uses the following statements:

PROC DBLOAD <DBMS=WK1|WK3|WK4>
<DATA= <*libref.*>*SAS-data-set*>;
PATH='*path-and-filename*<.WK1|.WK3|.WK4>' | <'>*filename*<'> | *fileref*;
PUTNAMES <=> YES | NO | Y | N;
ACCDESC= <*libref.*>*access-descriptor*;
DELETE *variable-identifier-1* <*…variable-identifier-n*>;
ERRLIMIT= *error-limit*;
FORMAT *SAS-variable-name-1 SAS-format-1* <=>
<*…SAS-variable-name-n SAS-format-n*>;
LABEL;
LIMIT= *load-limit* ;
LIST <ALL | COLUMNS | FIELDS | *variable-identifier*>;
RENAME *variable-identifier-1* <=> <'>*column-name-1*<'>
<*…variable-identifier-n* = <'>*column-name-n*<'>>;
RESET ALL | *variable-identifier-1* <*…variable-identifier-n*>;
<*…column-identifier-n* <=> C | N>;
WHERE *SAS-where-expression*;
LOAD;

RUN;

The QUIT statement is also available in PROC DBLOAD. However, it 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 followig list provides detailed information about the WK*n*-specific statements:

PUTNAMES <=> YES | NO | Y | N;
writes column names to the first row of the new WK*n* file. The column names can be default SAS variable names or, if you specify the LABEL statement, SAS variable labels. You can modify the column names using the RENAME statement.

The PUTNAMES statement is optional. If you omit PUTNAMES, data is read from the data set and written to the WK*n* file beginning in the first row of the WK*n* file, and no column names are written to the file.

You can change the default value to YES by setting the SS_NAMES environment variable. See "Setting Environment Variables for WK*n* Files" on page 182 for more information about setting and changing environment variables.

FORMAT *SAS-variable-name-1 SAS-format-1* <*SAS-variable-name-n SAS-format-n*>;
assigns a temporary format to a SAS variable in the input SAS data set. This format temporarily overrides any other format for the variable. The assignment lasts only for the duration of the procedure. Assign formats to as many variables as you want in one FORMAT statement.

Use FORMAT when you want to change the format, column width, or the number of decimal digits for columns being loaded into the PC file. For example, if you change the SAS variable format 12.1 to DOLLAR15.2, the column format of the loaded data changes from a fixed numeric format with a column width of 12 and one decimal digit to a currency format with a column width of 15 and two decimal digits.

DBLOAD Procedure Data Conversions for WK*n* Files

This section explains how SAS data is read into Lotus 1-2-3 data when a table is loaded. In this conversion, the SAS character data type is converted into the Lotus 1-2-3 label type and the SAS numeric type is converted into the Lotus 1-2-3 number type.

The SAS/ACCESS interface automatically converts SAS formats to the same or associated Lotus 1-2-3 formats and column widths. However, you can temporarily assign other formats and column widths to SAS variables by using the FORMAT statement. The following table shows SAS variable types and formats and the WK*n* data types, formats, and column widths that you can assign them to.

Note: The FORMAT statement in PROC DBLOAD only changes the format of SAS variables while you are creating and loading the WK*n* files. When the procedure is completed, the formats of SAS variables return to their original settings. △

WK*n* date and time values are numeric data. See "Datetime Conversions in the DBLOAD Procedure" on page 182 for more information.

Table 17.2 Converting SAS Variable Formats to WK*n* File Data

SAS Variable Format		WK*n* File Data			
Type	Data Format	Data Type	Column Format	Column Width	Number
Char	$*w*.	LABEL	DEFAULT	*w*	
Char	$CHAR*w*.	LABEL	DEFAULT	*w*	
Num	*w.d*	NUMBER	FIXED	*w*	*d*
Num	F*w.d*	NUMBER	FIXED	*w*	*d*
Num	E*w.d*	NUMBER	SCIENTIFIC	*w*	*d*
Num	DOLLAR*w.d*	NUMBER	CURRENCY	*w*	*d*
Num	PERCENT*w.d*	NUMBER	PERCENT	*w*	*d*
Num	COMMA*w.d*	NUMBER	COMMA	*w*	*d*
Num	BEST*w*.	NUMBER	DEFAULT	*w*	
Num	BEST*w*.	NUMBER	GENERAL	*w*	
Num	DATE5.	NUMBER	DD-MON	7	
Num	DATE7.	NUMBER	DD-MON-YY	10	
Num	MONYY5.	NUMBER	MON-YY	7	
Num	MMDDYY5.	NUMBER	MM-DD	6	
Num	MMDDYY8.	NUMBER	MM-DD-YY	9	
Num	TIME5.	NUMBER	HH-MM-SS	6	
Num	TIME8.	NUMBER	HH-MM-SS	9	
Num	TIME9.	NUMBER	HH-MM AM/ PM	9	
Num	TIME12.	NUMBER	HH-MM-SS AM/PM	12	

Datetime Conversions in the DBLOAD Procedure

If a SAS variable is specified with a date, time, or datetime format in the FORMAT statement, the interface view engine converts that SAS datetime format into the equivalent Lotus 1-2-3 datetime format when the new WK*n* file is created.

However, if a SAS datetime format is not specified in the input SAS data set, you have to assign a format by using a PROC DBLOAD FORMAT statement. Doing so assigns a Lotus 1-2-3 datetime format to the SAS variable when the variable is loaded into a WK*n* file. If you do not assign a SAS datetime format, the SAS numeric-datetime value is written to the WK*n* file. Because SAS dates are based on January 1, 1960, and Lotus 1-2-3 dates are based on January 1, 1900, the datetime value in the WK*n* file will be inaccurate.

To maintain a SAS variable format in the input data set, yet change it only while the DBLOAD procedure is in progress, use the FORMAT statement in PROC DBLOAD. This statement enables you to assign a temporary format to a SAS variable for the duration of the procedure without affecting the input SAS data set.

For example, if the SAS format for the BirthDat variable in the MyData.SasEmps access descriptor is left at the default 15.2 format, you can specify the FORMAT statement in the PROC DBLOAD statement. This specification changes the variable's format to DATE7. while you are creating and loading the WK*n* file. When you load the WK*n* file, the DATE7. format becomes an equivalent Lotus 1-2-3 column format, *DD-MON-YY*. When the DBLOAD procedure has completed, the SAS format for the BirthDat variable returns to the 15.2 format.

You can specify the FORMAT statement when you invoke the DBLOAD procedure to assign a temporary format to the variables in your input SAS data set. For more information, see "DBLOAD Procedure Syntax for WK*n* Files" on page 180.

Setting Environment Variables for WK*n* Files

You can change the default behavior of the SAS/ACCESS interface by setting environment variables in your SAS configuration file. You can set four SAS/ACCESS environment variables: SS_MISS NULLS, SS_MIXED, SS_NAMES, and SS_SCAN. Setting these variables in your SAS configuration file changes how the interface works by default.

The configuration file omits these three environment variables by default, which means their default values are NO.

SS_MISS NULLS
: By default, the DBLOAD procedure loads Lotus @NA cell values for missing values. Use this option to specify a null cell value instead. If set, missing values in a SAS data set will be displayed as blanks in the Lotus 1-2-3 table.

SS_MIXED YES | NO
: YES allows both Lotus 1-2-3 numeric and character data in a column to be displayed as SAS character data. The Lotus 1-2-3 numeric data is converted to its character representation when its corresponding SAS variable type is defined as character.

  NO does not convert Lotus 1-2-3 numeric data in a column into SAS character data. Lotus 1-2-3 numeric data is read in as SAS missing values when its corresponding SAS variable type is defined as character. NO is the default.

  Setting the SS_MIXED environment variable changes the default value of the MIXED statement in PROC ACCESS.

SS_NAMES YES | NO

YES in PROC ACCESS generates SAS variable names from column names in the first row of the worksheet or the specified range of the worksheet and reads data from the second row. YES in PROC DBLOAD writes column names using SAS variable names or SAS variable labels to the first row of the new WK*n* file, time reads data from the data set and writes it to the WK*n* file beginning with the second row.

NO in PROC ACCESS generates the SAS variable names VAR0, VAR1, VAR2, and so on, and reads data from the first row of the worksheet or specified range. NO in PROC DBLOAD reads the data from the data set and writes it to the WK*n* file beginning with the first row. NO is the default.

Setting the SS_NAMES environment variable changes the default value of the GETNAMES statement in PROC ACCESS and the PUTNAMES statement in PROC DBLOAD.

SS_SCAN YES | NO | *number-of-rows*

YES scans the data type and format of rows in a worksheet or specified range after skipping the number of rows specified in the SKIPROWS statement. SS_SCAN finds the most common Lotus 1-2-3 data type and format in order to generate the default SAS data type and format. If a number of rows is specified, SAS/ACCESS software scans only the data type and format from these rows.

NO uses the type and format of the first row in a worksheet or specified range, after skipping the number of rows specified in SKIPROWS, to generate the default SAS data type and format. NO is the default.

Number-of-rows scans the type and format of the specified number of rows only. Setting the number of rows is more efficient because data is read only from the specified number of rows rather than from the entire file.

Setting the SS_SCAN environment variable changes the default value of the SCANTYPE statement in PROC ACCESS.

WK*n* Essentials

SAS/ACCESS software for PC files works with WK1, WK3, and WK4 (Releases 4 and 5) files. These files contain data in the form of Lotus 1-2-3 spreadsheets and are referred to collectively in this document as WK*n* files, where *n* stands for 1, 3, or 4. SAS/ACCESS does not support the .123 format for files from Lotus SmartSuite 97 software.

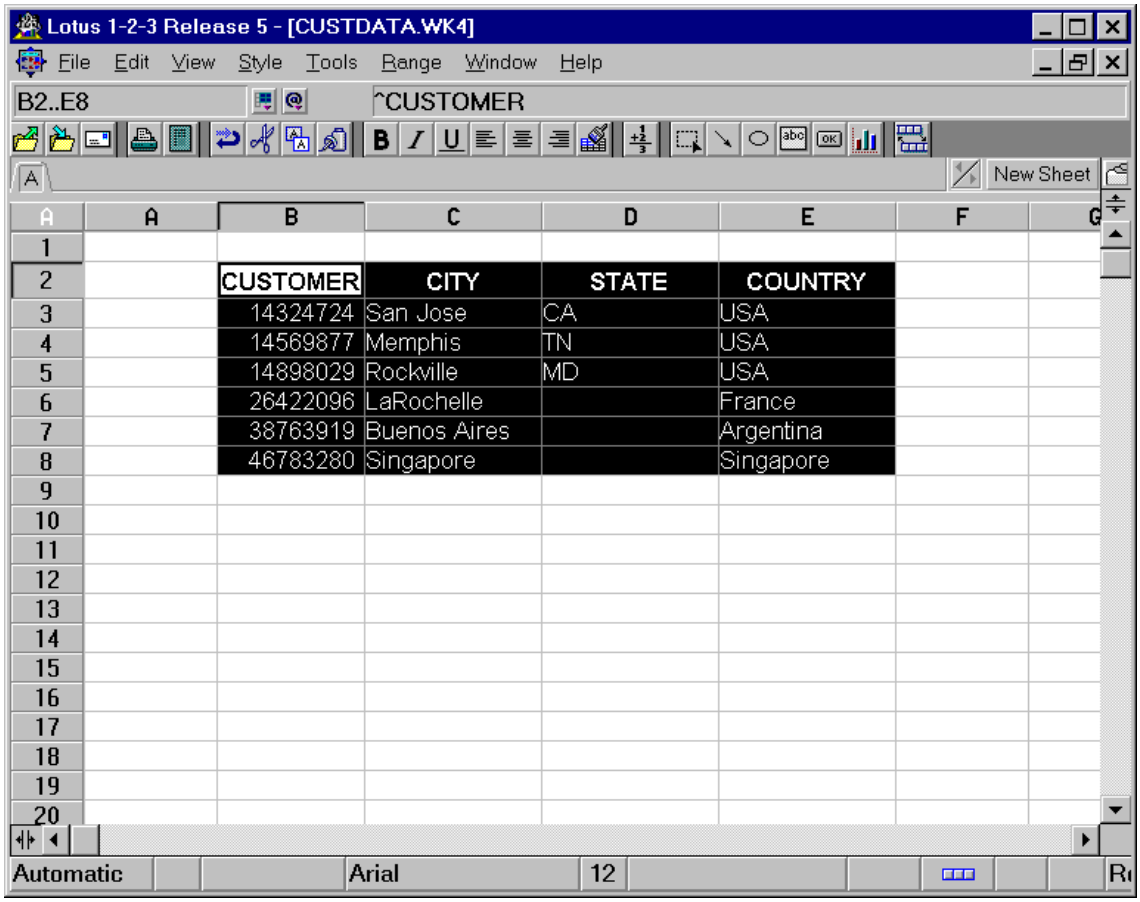
WK*n* Files

Various software products, such as the Lotus 1-2-3 spreadsheet and database system, enable you to use spreadsheet or database files to enter, organize, and perform calculations on data. Spreadsheets are most often used for general ledgers, income statements, and other types of financial record keeping. Database files also enable you to organize related information, such as, the data in an accounts-receivable journal.

In both spreadsheets and database files, the data is organized according to certain relationships among data items. These relationships are expressed in a tabular form, in columns and rows. Each *column* represents one category of data, and each *row* can hold one data value for each column.

A Lotus 1-2-3 worksheet is an electronic spreadsheet consisting of a grid of 256 columns and 8,192 rows. The intersection of a column and a row is called a *cell*. The following display illustrates a portion of a standard 1-2-3 worksheet.

Display 17.1 Columns and Rows of Data in a WK*n* File



Column letters for each column appear above the worksheet. Columns are lettered A through IV (A to Z, AA to AZ, BA to BZ, and so on to IV). Row numbers for each row appear to the left of the worksheet. Rows are numbered 1 to 8,192. For WK1 files, only one worksheet (worksheet A) is allowed per file. For WK3 and WK4 files, up to 256 worksheets (worksheets A-IV) are allowed. The SAS/ACCESS interface to WK*n* files uses only one worksheet, however, and defaults to worksheet A.

A *range* is a subset of cells in a worksheet. A range is identified by its address, which begins with the name of the top left cell and ends with the name of the bottom right cell separated by two periods. For example, the range B2..E8 is the range address for a rectangular block of 28 cells whose top left cell is B2 and whose bottom right cell is E8 (as shaded in the figure).

You can give a name to a range and use the name in commands and formulas instead of the range address in Lotus 1-2-3. A range name can be up to 15 characters long and should not contain any spaces. For example, if the range B3..D6 is named GRADE_TABLE, then the formula @AVG(GRADE_TABLE) has the same value as @AVG(B3..D6).

For more information about ranges and their naming conventions, see the documentation that accompanies your Lotus 1-2-3 software.

WK*n* File Naming Conventions

Filenames must also follow operating environment specific conventions, so check the documentation that comes with your Lotus 1-2-3 product or other software products for further information. The following conventions apply to WK*n* filenames:

- Under Windows 95, Windows 98, and Windows NT, the ACCESS and DBLOAD procedures support long names that are specified in the PATH= statement (such as **`path= 'c:\sasdemo\library\new_customer_1999.wk4';`**). However, WK*n* files with long names might not be accepted by some versions of Lotus 1-2-3.
- Filenames can contain up to eight characters.
- Filenames start with a letter, and they can contain any combination of the letters A through Z, the digits 0 through 9, the underscore (_), the hyphen (-), and spaces (blanks).
- Filenames can contain spaces. Filenames that contain spaces or lowercase letters are supported by the ACCESS and DBLOAD procedures, but they might not be accepted by some versions of Lotus 1-2-3.

WK*n* Data Types

Lotus 1-2-3 software has two data types: character and numeric. Lotus 1-2-3 character data can be entered as labels or formula string. Lotus 1-2-3 numeric data can be entered as numbers or formulas.

Character data is generally considered text and can include dates and numbers if prefixes are used to indicate character data and to align the data in the cell. For example, in Lotus 1-2-3, the value **`"110 Maple Street`** uses the double quote prefix and aligns the label on the right side of the cell.

Numeric data can include numbers (0 through 9), formulas, and cell entries that begin with one of the following symbols: +, $, @, –, or #.

Numeric data also can include *date and time values*. In Lotus 1-2-3 software, a date value is the integer portion of a number that can range from 01 January 1900 to 31 December 2099, that is, 1 to 73,050. A Lotus 1-2-3 software time value is the decimal portion of a number that represents time as a proportion of a day. For example, 0.0 is midnight, 0.5 is noon, and 0.999988 is 23:59:59 (on a 24-hour clock). While a number can have both a date and a time portion, the formats in Lotus 1-2-3 display a number only in a date format or a time format. The conversion of date and time values between SAS data sets and Lotus 1-2-3 spreadsheets is transparent to users. However, you are encouraged to understand the differences between them. For information about how the SAS/ACCESS interface handles date and time values and formats, see “Datetime Conversions in the ACCESS Procedure” on page 179 and “Datetime Conversions in the DBLOAD Procedure” on page 182.

When you create an access descriptor, the interface software uses the column types and formats in the WK*n* file to determine the corresponding SAS variable formats. SAS generates its default formats based on the values that you specify for the SCANTYPE, SKIPROWS, and GETNAMES statements. You can change the formats generated by the software interface. For more information, see “How SAS/ACCESS Works with WK *n* Files” on page 186.

When you browse a view descriptor, any data value that does not match the column type (character or numeric) specified in the descriptor is treated as a missing value. This is the default action. However, you can use the MIXED=YES statement to convert numeric data values in a character column to their character representation when you create an access descriptor.

You can also set the SS_MIXED environment variable to **`YES`** in your SAS configuration file so that both numeric and character data are displayed as SAS character data. Add this line to your SAS configuration file:

```
-SET SS_MIXED YES
```

See “Setting Environment Variables for WK*n* Files” on page 182 for more information about environment variables. For more information about changing the column type, refer to “ACCESS Procedure: WK*n* Specifics” on page 176.

How SAS/ACCESS Works with WK *n* Files

The SAS/ACCESS interface accesses data in the Lotus 1-2-3 WK*n* files directly. It enables you to create SAS data sets from WK*n* files or directly read the WK*n* file data without creating SAS data sets. The interface does not allow you to update, add, or delete data in WK*n* files.

Accessing the Data

To access the data, the interface accesses a range in a worksheet as a table. If the range is not specified, the interface accesses the entire worksheet as a table. By default, the interface uses the Lotus 1-2-3 formats of columns in the first row of the range to determine the formats of variables in SAS/ACCESS descriptors.

However, you can manipulate where the interface begins to read data and what format the interface generates by using the SKIPROWS and SCANTYPE statements in the ACCESS procedure. SKIPROWS skips a specified number of rows before reading data. SCANTYPE finds the most common data type from among a specified number of rows within a WK*n* range (after skipping the number of rows specified in SKIPROWS) and uses it to generate the default format for SAS variables.

The ACCESS procedure enables you to create access descriptors and view descriptors for WK*n* files. You then can use the view descriptors as SAS data sets.

You can retrieve a subset of data using the WHERE statement.

To sort WK*n* file data, you must first extract the data from a WK*n* file and place it in a SAS data file, unless you are using the SQL procedure. (The SQL procedure enables you to present output data in a sorted order using the ORDER BY clause of the SELECT statement.) You can extract and sort WK*n* file data in one step with the OUT= option in the SORT procedure, using a view to the WK*n* file as input to PROC SORT.

Creating and Loading the Data

When you use PROC DBLOAD to create and load WK*n* files, the procedure translates the SAS data set into a WK*n* file. The file is stored in the location specified by the PATH= statement. Only one SAS data set can be loaded into a WK*n* file at one time. The loaded WK*n* file can contain only one worksheet. Lotus 1-2-3 then reads data from the loaded WK*n* file directly.

In the DBLOAD procedure, you can specify the PUTNAMES statement to place the SAS variable names in the first row of the spreadsheet and the first observation in the second row, and so on. If PUTNAMES is not specified, the first observation is placed in the first row, the second observation is placed in the second row, and so on. Columns do not have names. The formats for SAS variables are automatically converted to the corresponding Lotus 1-2-3 types and formats. See the descriptions of individual statements for more information about how the data and columns are read.

CHAPTER

18

dBase DBF Files

How To Access DBF Files from SAS

You can interact with dBASE (DBF) files from SAS by using the following features:

Import/Export wizard or procedures (UNIX and Windows operating environments)
enable you to transfer data between SAS and several PC files.

DBF procedure (UNIX, Windows, and OS/390 operating environments)
enables you to convert data between dBASE (DBF) files and SAS data sets.

ACCESS procedure (Windows operating environments)
creates descriptor files that describe data in a PC file to SAS, enabling you to directly read, update, or extract PC files data into a SAS data file.

DBLOAD procedure (Windows operating environments)
creates PC files and loads them with data from a SAS data set.

This section contains DBF-specific information for the ACCESS and DBLOAD procedures. See Chapter 4, “The Import/Export Wizard and Procedures,” on page 49 and Chapter 5, “The DBF and DIF Procedures,” on page 59 for information about those features.

ACCESS Procedure: DBF Specifics (Windows)

Chapter 6, “The ACCESS Procedure for PC Files,” on page 65 contains general information about this feature. This section provides DBF-specific syntax for the ACCESS procedure and describes ACCESS procedure data conversions.

ACCESS Procedure Syntax 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 SAS and must immediately follow the CREATE statement. In addition to the database-description statement, you can use optional editing statements when you create an access descriptor. These editing statements must follow the database-description statement.

The database-description statement is 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.

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

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

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, it causes the procedure to terminate. QUIT is used most often in the interactive line and non-interactive 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;
```

ACCESS Procedure Data Conversions for DBF Files

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

Table 18.1 Default SAS Variable Formats for DBF File Data Types

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 V and later. Check with other software products' documentation to see if this data type applies.

DBLOAD Procedure: DBF Specifics (Windows)

Chapter 7, "The DBLOAD Procedure for PC Files," on page 91 contains general information about this feature. This section provides DBF-specific syntax for the DBLOAD procedure and describes DBLOAD procedure data conversions .

DBLOAD Procedure Syntax for DBF Files

To create and load a DBF table, the SAS/ACCESS interface to PC files 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, it causes the procedure to terminate. QUIT is used most often in the interactive line and non-interactive modes to exit the procedure without exiting SAS.

The following list provides detailed information about the DBF-specific statements:

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, and 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.

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;
```

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;
```

DBLOAD Procedure Data Conversions for DBF Files

The following table shows the default DBF file data types that the DBLOAD procedure assigns to each SAS variable format.

Table 18.2 Default DBF File Data Types for SAS Variable Formats

SAS Variable Formats	DBF File Data Types
$*w*.	CHAR(*n*)
w.	NUMERIC
w.d.	NUMERIC
datetime*w.d*	DATE
date*w*.	DATE
e*w*.	FLOAT
binary*w*.	NUMERIC

DBF Essentials

The SAS/ACCESS interface to PC files 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. SAS/ACCESS cannot access DBF files created by Visual dBASE 7.

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

DBF Files

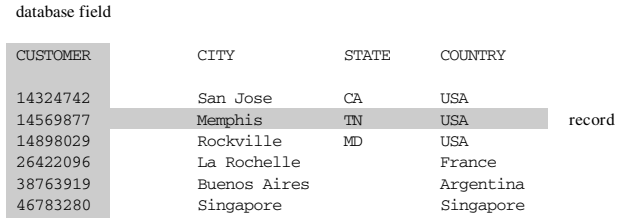
DBF files are a file format created by dBASE, a relational database management system for microcomputer systems. DBF files can be created using a variety of microcomputer software programs.

A DBF file contains data that is organized in a tabular format of database fields and records. Each *database field* can contain one type of data, and each *record* can hold one data value for each field. Figure 18.1 on page 192 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 product's documentation for information about creating DBF files and assigning field names, field types, and other attributes.

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

Figure 18.1 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

File names 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 file names and field names:

- Under Windows 95, Windows 98, and Windows NT, 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.
- File names 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.
- File names 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 is 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 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 as follows:

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, SAS 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 SAS. If a field in SAS indicates a missing value (represented by a period), SAS writes a nine 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 nines, 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 might 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 files.

Chapter 7, "The DBLOAD Procedure for PC Files," on page 91 describes how the DBLOAD procedure determines data types when creating DBF files.

Handling Missing Values in DBF Files

Missing numeric values are filled in with nines by default. The DBFMISCH environment variable 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 SAS.

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**.

How SAS/ACCESS Works with DBF Files

For DBF files, the SAS/ACCESS interface is a *read-write interface*. When you use the ACCESS procedure to create an access descriptor, SAS retrieves descriptive information about the database fields directly from the DBF file. When you create a view descriptor, SAS 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 were created previously on that access descriptor and, therefore, invalidate the view descriptors.

However, if the data in the DBF file changes, the updated data does appear when it is 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.

SAS 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 procedures 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 DBF file interface view engine reads from or writes to the DBF file that is stored in the path you specified.

CHAPTER
19

Lotus DIF Files

How To Access DIF Files from SAS

You can interact with data interchange format (DIF) files from SAS by using the following features:

DIF procedure (UNIX and Windows operating environments)
enables you to convert between DIF files and SAS data sets.

ACCESS procedure (Windows operating environments)
creates descriptor files that describe data in a PC file to SAS, enabling you to directly read, update, or extract PC files data into a SAS data file.

DBLOAD procedure (Windows operating environments)
creates PC files and loads them with data from a SAS data set.

This section contains DIF-specific information for the ACCESS and DBLOAD procedures. See “The DIF Procedure” on page 62 for information about the DIF procedure.

ACCESS Procedure: DIF Specifics

Chapter 6, “The ACCESS Procedure for PC Files,” on page 65 contains general information about this feature. This section provides DIF-specific syntax for the ACCESS procedure and describes ACCESS procedure data conversions.

ACCESS Procedure Syntax for DIF Files

To create an access descriptor, you use the DBMS=DIF option and the database-description statements PATH=, DIFLABEL, and SKIPROWS. These statements supply DIF-specific information to SAS, 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 DIF 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 DIF uses the following procedure statements:

PROC ACCESS *options*;
CREATE <*libref.*>*member-name.*ACCESS | VIEW;

UPDATE <*libref.*>*member-name.*ACCESS | VIEW;

PATH= *'path-and-filename*<.DIF>' | <'>*filename*<'>| *fileref*;
DIFLABEL;
SKIPROWS <=> *number-of-rows-to-skip*;
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* ;
TYPE <'>*column-identifier-1*<'><=> C | N
<…*column-identifier-n* <=> C | N>;
UNIQUE <=> YES | NO;

RUN;

The QUIT statement is also available in PROC ACCESS. However, it 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 list provides detailed information about the DIF-specific statements:

DIFLABEL
indicates whether variable names are generated from the first row of the columns. If you omit this statement, variable names that are generated are based on the columns' placement in the first row. That is, SAS labels each column as **COL0**, **COL1**, **COL2**, and so on. These labels become the names of SAS variables in the access descriptor.

If you specify DIFLABEL, the ACCESS procedure reads column labels from the first row of the DIF file and uses them as the SAS variable names in the access descriptor. You provide the DIF file column labels; they are not the letters (for

example, A, B, and so on) that identify the columns in a worksheet. If you specify DIFLABEL, the SKIPROWS statement automatically changes to 1.

Always specify DIFLABEL after the PATH= statement and before any editing statements. When you update a descriptor, you are not allowed to specify the DIFLABEL statement.

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

```
options linesize=80;
libname difdliba 'SAS-data-library';
libname diflibv 'SAS-data-library';

proc access dbms=dif;
/* create access descriptor */
   create difliba.custs.access;
   path='c:\difiles\dbcusts.dif';
   diflabel;
   skiprows=2;
   assign=yes;
   rename customer = custnum;
   format firstorder date9.;
   list all;

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

SKIPROWS <=> *number-of-rows-to-skip*;

specifies the number of rows, beginning at the top of the DIF file, to ignore when you read data from the file. The default value for SKIPROWS is **0**. The skipped (or ignored) rows often contain information such as column labels or names, or underscores rather than input data.

If you specify the DIFLABEL statement, the default value of SKIPROWS automatically changes to **1**. The SKIPROWS statement should always follow the PATH= statement and precede any editing statements when you are creating a descriptor. The first row of data after SKIPROWS is used to generate the SAS variable types and formats. If there is no data in the first row of a column after SKIPROWS, the data in the rest of the column is assumed to be character data, even if the data in the next row is numeric.

By default, any data value in a column that does not match the type is treated as a missing value. However, if you set the DIFNUMS environment variable to **YES** in your SAS configuration file, any numeric data values in a character column are converted to the character representation of the number and are not treated as missing values. Add the following line to your SAS configuration file to set the DIFNUMS environment variable to **YES**:

```
-SET DIFNUMS YES
```

The default for the DIFNUMS environment variable is **NO**. Refer to the SAS documentation for your operating environment for more information about environment variables.

You can change the column type from the type determined by SAS/ACCESS software when you create an access descriptor.

ACCESS Procedure Data Conversions for DIF Files

The following table shows the default SAS variable formats that the ACCESS procedure assigns to each type of DIF file data. DIF file numeric data includes date and time values. See "Datetime Conversions in the ACCESS Procedure" on page 198 for more information.

Table 19.1 Default SAS Variable Formats for DIF File Data

DIF File Data	SAS Variable Format
C (Character)	$20.
N (Numeric)	15.2

If DIF file data falls outside of the valid SAS data ranges, you get an error message in the SAS log when you try to read the data.

Datetime Conversions in the ACCESS Procedure

When you create an access descriptor, SAS cannot distinguish a Lotus datetime value from other numeric data. SAS stores the Lotus datetime value as a number and displays it like other Lotus numeric data by using the SAS variable format 15.2 (the default format for this interface).

To convert a Lotus datetime value to a SAS datetime value, you must specify a SAS datetime format in the access descriptor. A Lotus datetime value is a number that represents the number of days between January 1, 1900, and a specified date; changing the default SAS format (15.2) to a datetime format in the descriptor causes the Lotus value to be converted to an equivalent SAS datetime value based on January 1, 1960. In other words, the Lotus numeric value for January 1, 1960 (which is 21,916) is converted to the equivalent SAS representation of January 1, 1960 (which is 0) only if a SAS datetime format is stored in the descriptor for that column. Otherwise, the Lotus value of 21,916 is treated as a SAS numeric value of 21,916.

The following table shows how SAS uses a Lotus datetime value to convert to a SAS datetime format.

Table 19.2 Value-to-Format Conversions

For a SAS format	SAS uses
date	integer portion of the Lotus number
time	decimal portion of the Lotus number
date-and-time	integer and decimal portion of the Lotus number

DBLOAD Procedure: DIF Specifics

Chapter 7, "The DBLOAD Procedure for PC Files," on page 91 contains general information about this feature. This section provides DIF-specific syntax for the DBLOAD procedure and describes DBLOAD procedure datetime conversions .

DBLOAD Procedure Syntax for DIF Files

To create and load a DIF table, the SAS/ACCESS interface to PC files uses the following statements.

PROC DBLOAD DBMS=DIF <DATA=<*libref.*>*SAS-data-set*>;

PATH=*'path-and-filename*<.DIF>' | <'>*filename*<'>| *fileref*;

DIFLABEL;

ACCDESC=<*libref.*>*access-descriptor*;

DELETE *variable-identifier-1* <*…variable-identifier-n*>;

ERRLIMIT=*error-limit*;

FORMAT *SAS-variable-name-1 SAS-format-1* <*…SAS-variable-name-n SAS-format-n*>;

LABEL;

LIMIT=*load-limit*;

LIST <ALL | COLUMNS | FIELDS | *variable-identifier*>;

LOAD;

RENAME *variable-identifier-1*= <'>*column-name-1*<'> <*…variable-identifier-n*=<'>*column-name-n*<'>>;

RESET ALL | *variable-identifier-1* <*…variable-identifier-n*>;

WHERE *SAS-where-expression*;

RUN;

The QUIT statement is also available in PROC DBLOAD. However, it 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 list provides detailed information about the DIF-specific statements:

DIFLABEL
writes column labels to the first row of the new DIF file and follows the column labels with a blank row. The column labels can be default SAS variable names or, if you specify the LABEL statement, SAS labels. You can modify the column labels using the RENAME statement.

If this statement is omitted, data is read from the data set and written to the DIF file beginning in the first row of the DIF file, and no column labels are written to the file.

FORMAT *SAS-variable-name-1 SAS-format-1*< *…SAS-variable-name-n sas-format-n*>;
assigns a temporary format to a SAS variable in the input SAS data set. This format temporarily overrides any other format for the variable. The assignment lasts only for the duration of the procedure. Assign formats to as many variables as you want in one FORMAT statement.

Use FORMAT when you want to change the format, column width, or the number of decimal digits for columns being loaded into the PC file. For example, if you change the SAS variable format 12.1 to DOLLAR15.2, the column format of the loaded data changes from a fixed numeric format with a column width of 12 and one decimal digit to a currency format with a column width of 15 and two decimal digits.

The following example creates a new DIF table, Exchange.dif, from the data file Dlib.RateOfex. An access descriptor AdLib.Exchange is also created, based on the new

DIF table. You must be granted the appropriate privileges in order to create new DIF files.

```
libname difdliba 'SAS-data-library';
libname diflibv 'SAS-data-library';

proc dbload dbms=dif data=dlib.rateofex;
   accdesc=adlib.exchange;
   path='c:\difiles\sasdemo\exchange.dif';
   diflabel;
   rename fgnindol=fgnindolar 4=dolrsinfgn;
   load;
run;
```

Datetime Conversions in the DBLOAD Procedure

If a SAS variable is specified with a date, time, or datetime format in the FORMAT statement, the interface view engine converts that datetime value into the equivalent Lotus datetime value when the new DIF file is created. However, the DIF file has no way of relating this formatting information to Lotus products. Therefore, when you load the DIF file into a Lotus 1-2-3 worksheet, the datetime values are represented as numbers. You should assign (from within Lotus) a Lotus datetime format to any datetime column that you load from a DIF file.

If a SAS variable represents a date, time, or datetime value, but it has not been assigned a SAS datetime format — the SAS datetime value is represented as a number — the number is *not converted* into an equivalent Lotus datetime value in the DIF file. Rather, the number is written to the new DIF file as is. Because SAS dates are based on January 1, 1960, and Lotus dates are based on January 1, 1900, if you assign a Lotus datetime format to an unconverted Lotus column, the datetime values in that column are inaccurate.

To maintain a SAS variable format in the input data set, yet change it only while the DBLOAD procedure is in progress, use the DBLOAD FORMAT statement. This statement enables you to assign a temporary format to a SAS variable for the duration of the procedure without affecting how SAS stores the variable.

For example, if the SAS format for the BirthDat variable in the MyData.SasEmps data set is left at the default 15.2 format, you can specify the FORMAT statement to change the variable's format to DATE7. while you are creating and loading the DIF file. When you load the DIF file into a Lotus 1-2-3 worksheet, you can specify an equivalent Lotus date format. When the DBLOAD procedure has completed, the SAS format for the BirthDat variable reverts to the 15.2 format.

You can specify the FORMAT statement in the PROC DBLOAD statement when you invoke the procedure using any of the methods of processing.

Note: There are certain display restrictions on the SAS datetime values that are loaded into Lotus 1-2-3 worksheets through DIF files. If you load a SAS variable with a DATETIME*w.d* format into a DIF file, Lotus stores the number with both integer and decimal portions. However, when you load the DIF file into a Lotus 1-2-3 worksheet and specify a format for the column, you can only specify a date format (that uses the integer portion) or time format (that uses the decimal portion) for that column, not both at the same time. △

DIF Essentials

Data interchange format (DIF) files are used by the SAS/ACCESS interface to PC files to access data indirectly from other software products, such as data in Lotus 1-2-3 spreadsheets and database files.

DIF Files

DIF files can be created using software under a variety of microcomputer software packages (such as Lotus 1-2-3). These software products enable you to use spreadsheet or database files to enter, organize, and perform calculations on data. Spreadsheets are most often used for general ledgers, income statements, and other types of financial record keeping. Database files also enable you to organize related information, such as, the data in an accounts-receivable journal.

In both spreadsheets and database files, the data is organized according to certain relationships among data items. These relationships are expressed by files in a tabular form, that is, in columns and rows. DIF files allow both character and numeric data in the same column. See "DIF File Data Types" on page 202 in this chapter for more information. Each *row* can hold one data value for each column. The spreadsheet and database files can be translated to DIF files that the SAS/ACCESS interface can process.

A spreadsheet consists of columns and rows, and their intersection is called a *cell*. The following figure illustrates four columns from the spreadsheet Customers and highlights a column and a row.

Figure 19.1 Columns and Rows of Data in a DIF File

column

CUSTOMER	CITY	STATE	COUNTRY	
14324742	San Jose	CA	USA	
14569877	Memphis	TN	USA	row
14898029	Rockville	MD	USA	
26422096	La Rochelle		France	
38763919	Buenos Aires		Argentina	
46783280	Singapore		Singapore	

DIF File Naming Conventions

DIF file names must follow operating environment specific conventions, so check the documentation that comes with your application or operating system software for further information.

- Under Windows 95, 98, NT, 2000, or XP the ACCESS and DBLOAD procedures support long names that are specified in the PATH= statement (such as **`path= 'c:\sasdemo\library\new_customers_1999.dif';`**). However, some applications that support DIF files might not accept files with long names.
- File names start with a letter, and they can contain any combination of the letters A through Z, the digits 0 through 9, and the underscore (_).

DIF File Data Types

Every column in a DIF file has a name and one or two data types. A DIF file allows columns that include both character and numeric data.

Character data is generally considered text and can include dates and numbers if prefixes are used to indicate character data and to align the data in the cell. For example, in Lotus 1-2-3, the value **"110 Maple Street** uses the double quote prefix and aligns the label on the right side of the cell.

Numeric data includes numbers (0 through 9), formulas, and cell entries that begin with one of the following symbols: +, $, @, –, or #. When you create and load a DIF file with PROC DBLOAD, the SAS/ACCESS engine supplies **NA** for a missing, numeric value. For decimal numbers, the SAS/ACCESS engine queries the operating environment for the current setting of the decimal separator and uses it when reading or creating DIF files.

Numeric data also include *date and time values*. In Lotus software, a date value is the integer portion of a number that can range from 01 January 1900 to 31 December 2099, that is, 1 to 73,050. A Lotus software time value is the decimal portion of a number that represents time as a proportion of a day. For example, 0.0 is midnight, 0.5 is noon, and 0.999988 is 23:59:59 (on a 24-hour clock). While a number can have both a date and a time portion, the formats in Lotus 1-2-3 display a number only in a date format or a time format. For information about how the SAS/ACCESS interface handles date and time values and formats, see “Datetime Conversions in the ACCESS Procedure” on page 198 and “Datetime Conversions in the DBLOAD Procedure” on page 200.

When you create an access descriptor, the interface software determines the column type by the value in the first row of data (excluding any rows that are defined for column names, blank rows for readability, and so on). If the first row in the column has no data value, the column type defaults to character data.

By default, any data value in a column that does not match the type is treated as a missing value. However, if you set the DIFNUMS environment variable to **YES** in your SAS configuration file, any numeric data values in a character column are converted to the character representation of the number and are not treated as missing values. Add the following line to your SAS configuration file to set the DIFNUMS environment variable to **YES**:

```
-SET DIFNUMS YES
```

The default for the DIFNUMS environment variable is **NO**. Refer to the SAS Companion for your operating system for more information about environment variables.

You can change the column type from the type determined by SAS/ACCESS software when you create an access descriptor.

How SAS/ACCESS Works With DIF Files

The SAS/ACCESS interface to DIF files accesses data in spreadsheets and databases indirectly. Spreadsheet and database data must be translated into a DIF file format before it can be read by SAS. A DIF file is an ASCII text file with a file header section and a data section. DIF files, not spreadsheets or databases, are specified in the ACCESS and DBLOAD procedures. You use your software product’s utilities to translate your spreadsheets and databases into DIF files. For example, you can use the Lotus 1-2-3 Translate Utility to translate a Lotus 1-2-3 worksheet or database to a DIF file. Once your spreadsheet or database is translated into a DIF file, the file is stored in

a directory that you specify. You then enter this path and DIF filename with the PATH= statement in the ACCESS procedure.

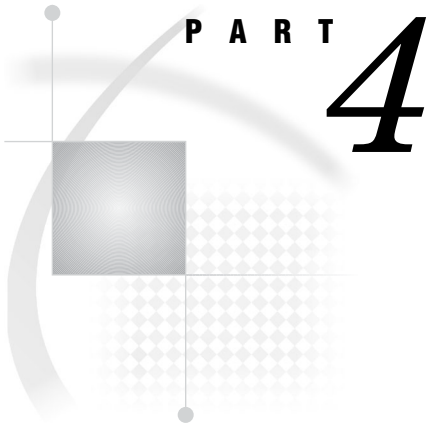
If you change a spreadsheet or database file after translating the file to DIF format, retranslate the modified file and save it in a new DIF file. If you do not, the DIF file and SAS/ACCESS view based on the DIF file will not reflect your changes to the original.

The SAS/ACCESS interface to DIF files is read-only: it cannot be used to modify a DIF file.

To sort data in a DIF file, you must first extract the data into a data file. You can do this in one step with the SORT procedure's OUT= option. Or you can use the SQL procedure's SELECT statement with an ORDER BY clause.

The DBLOAD procedure translates a SAS data set into a DIF file format and stores the DIF file in the path specified by the PATH= statement. Software products such as Lotus 1-2-3 can then read data from the DIF file.

When you use a view descriptor to a DIF file in a DATA step or procedure, you provide a path to the DIF file. The DIF file interface view engine retrieves data from this file.



PART 4

Sample Code

CHAPTER

20

Accessing PC Files Data with the LIBNAME Statement

Introduction to Accessing PC Files Data with the LIBNAME Statement

One advantage of using SAS/ACCESS software is that it enables SAS to read and write PC files data directly from SAS programs. This section presents examples in which PC files data accessed through LIBNAME statements is used as input data for SAS programs. It also shows you how to use SAS procedures and the DATA step to review PC file data that is directly accessed by SAS/ACCESS LIBNAME statements.

The examples in this section use Microsoft Access data. The PC file is identified in each example and any file-specific issues are described in the example.

Running the LIBNAME Examples

The examples in this chapter use data in different PC files. The PC files data is identified in each example and any file-specific issues are described in the example.

The examples in this chapter show the following:

- how to create LIBNAME statements
- how to use the LIBNAME statements in SAS procedures and DATA steps.

The files that create the PC files tables and the examples are shipped with your SAS/ACCESS software. See “Sample Data in This Document” on page 4 for more information about these files.

Charting PC Files Data with the LIBNAME Statement

This example shows how to use the GCHART procedure with DBMS data by using the LIBNAME statement to accomplish the task in an easy and direct way. The LIBNAME statement accommodates member names and variable names of up to 32 characters.

The GCHART Procedure with a SAS/ACCESS LIBNAME Statement

This example uses the GCHART procedure to chart data from the Microsoft Access table Orders. The LIBNAME statement is used to define a SAS libref that references Microsoft Access data.

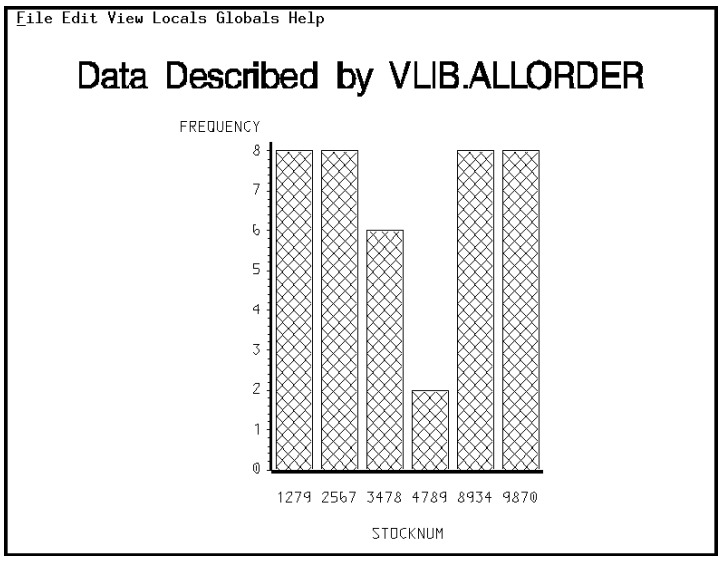
Note: Using this procedure requires a SAS/GRAPH license at your site. △

```
libname mydblib access 'c:/sampdata/samples.mdb' user=dmitry pw=elvis

proc gchart data=mydblib.orders;
vbar stocknum / discrete;
title 'Data Described by VLIB.ALLORDER';
run;
```

The following output shows the output for this example. STOCKNUM represents each product. The number of orders for each product is represented by the height of the bar.

Output 20.1 Vertical Bar Chart of Number of Orders per Product



For more information about the GCHART procedure, see *SAS/GRAPH Reference, Volumes 1 and 2*.

Calculating Statistics with the PC Files LIBNAME Statement

This example shows how to use the FREQ procedure with DBMS data. The LIBNAME statement supports long names of up to 32 characters.

The FREQ Procedure with a SAS/ACCESS LIBNAME Statement

This example uses the FREQ procedure to calculate statistics on the Microsoft Access table Invoice. The LIBNAME statement is used to define a SAS libref that references Microsoft Access data.

```
libname mydblib access 'c:/sampdata/samples.mdb';

proc freq data=mydblib.invoice(keep=invoicenum amtbilled country billedby paidon);
tables country;
title 'Data Described by VLIB.INV'
run;
```

The following output shows the one-way frequency table that this example generates.

Output 20.2 Frequency Table for Variable COUNTRY Described by View Descriptor VLib.Inv

Data Described by VLIB.INV 6

COUNTRY

COUNTRY	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Argentina	2	11.76	2	11.76
Australia	1	5.88	3	17.65
Brazil	4	23.53	7	41.18
USA	10	58.82	17	100.00

For more information about the FREQ procedure, see *Step-by-Step Programming with Base SAS Software* and *Base SAS Procedures Guide*.

Selecting and Combining PC Files Data with the LIBNAME Statement

This example shows how to use the WHERE statement to subset DBMS data.

The WHERE Statement with a SAS/ACCESS LIBNAME Statement

This example uses a WHERE statement directly in the PRINT procedure to print only unpaid bills over $300,000. The LIBNAME statement is used to define a SAS libref that references the Microsoft Access data. Column names can be up to 32 characters.

```
libname mydblib access 'c:/sampdata/samples.mdb' user=dmitry pw=elvis;

proc print data=mydblib.invoice(where=(paidon is null and amountinus>=300000.00)
   drop=paidon);
format amountinus dollar20.2;
title 'High Bills---Not Paid';
run;
```

Output 20.3 Work.NotPaid Data File Created Using a SAS WHERE Statement

High Bills--Not Paid

OBS	INVNUM	BILLEDTO	AMTINUS	BILLEDON
1	11271	18543489	$11,063,836.00	05OCT1998
2	12102	18543489	$11,063,836.00	17NOV1998
3	11286	43459747	$11,063,836.00	10OCT1998
4	12051	39045213	$2,256,870.00	02NOV1998
5	12471	39045213	$2,256,870.00	27DEC1998
6	12476	38763919	$2,256,870.00	24DEC1998

CHAPTER

21

Accessing PC Files with Descriptors

Introduction to Accessing PC Files with Descriptors

One advantage of using SAS/ACCESS software is that it enables SAS to read and write PC files data directly from SAS programs. This section presents examples in which PC files data accessed through view descriptors is used as input data for SAS programs, and it also shows you how to use SAS procedures and the DATA step to review and update PC files data that is described by SAS/ACCESS view descriptors.

The examples in this section use DIF, DBF, WK*n*, and XLS data. The PC file format is identified in each example and any file-specific issues are described in the example. Throughout the examples, the SAS terms *variable* and *observation* are used instead of column and row because this section illustrates SAS procedures and the SAS DATA step. The examples show how to create access descriptors and view descriptors and then use the view descriptors in SAS procedures and DATA steps. For more information about the SAS language and procedures that are used in the examples, refer to the documents listed at the end. For information about using view descriptors efficiently in SAS programs, see “Performance and Efficient View Descriptors for PC Files” on page 70.

In examples that update DBF file data, examples that are rerun will not work the same because the data has been modified. In this case, submit the PcfFdbl.sas file to re-create the PC files tables.

See Appendix 1, "Sample Data," on page 243 for all the PC files on which the access and view descriptors are defined. This appendix also includes the SAS data files that are used in this section, as well as the SAS statements that created them.

Running the Descriptor Examples

The examples in this section use data in different PC file formats. The PC files data is identified in each example and any file-specific issues are described in the example.

The examples show the following:

- how to create access descriptors and view descriptors
- how to use the view descriptors in SAS procedures and DATA steps.

As you work through the examples, notice that you can create the descriptors in a number of ways. In some cases, the ASSIGN=YES statement is specified and SAS variable names and formats are assigned when the access descriptor is created. In other cases, the ASSIGN statement is omitted and editing statements, such as RENAME and UNIQUE, are specified when the view descriptors are created. How you create descriptors depends on your site's needs and practices. When you run the examples, you only need to create an access descriptor or a view descriptor one time per example. If you rerun the examples, you do not need to re-create the descriptors.

The macro file (PcfFmac.sas) provided with the files contains macros that enable any SAS/ACCESS interface for a PC format to create database-description statements. Use the macro file with PcfFdbl.sas (creates PC files), PcfFsamp.sas (contains samples) and PcfFscl.sas (contains SAS/AF examples). To adapt the PcfFmac.sas file for use at your site, insert your PC file format in the first line of the code. See the comments in the PcfFmac.sas file for more information.

If you run the examples individually instead of running the entire examples file, you must preface them with LIBNAME statements to identify where your SAS data libraries are stored. In these examples, the libref Dlib is used for SAS data files; the libref SLib is used for PROC SQL views; the libref AdLib is used when creating access descriptors; and the libref VLib is used when creating view descriptors.

The files that create the PC files tables, descriptors, and the examples are shipped with your SAS/ACCESS software. See "Sample Data in This Document" on page 4 for more information about these files.

Reviewing Variables

Before retrieving or updating the PC files data that is described by a view descriptor, you might want to review the attributes of the data's variables. You can use the CONTENTS or DATASETS procedure to display a view descriptor's variable and format information. You can use these procedures with view descriptors in much the same way you use them with other SAS data sets.

This example uses the DATASETS procedure to display information about the view descriptor VLib.UsaCust, which describes the data in the Customers.wk3 file.

```
options linesize=80;

proc access dbms=wk3;
   create adlib.customr.access;
```

```
   /* create access descriptor  */
   path="c:\sasdemo\customer.wk3";
   worksheet=a;
   range='a1..j22';
   getnames=yes;
   scantype=5;
   mixed=yes;
   assign=yes;
   rename customer=custnum;
   format firstorder date9.;
   list all;

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

proc datasets library=vlib memtype=view;
   /* example             */
   contents data=usacust;
run;
```

The following output shows the results of this example.

Output 21.1 Using the DATASETS Procedure with a View Descriptor

```
                          DATASETS PROCEDURE

       Data Set Name: VLIB.USACUST          Observations:         21
       Member Type:   VIEW                  Variables:            5
       Engine:        SASIOWK3              Indexes:              0
       Created:       .                     Observation Length:   83
       Last Modified: .                     Deleted Observations: 0
       Protection:                          Compressed:           NO
       Data Set Type:                       Sorted:               NO
       Label:

           -----Alphabetic List of Variables and Attributes-----

   #    Variable    Type    Len    Pos    Format    Informat    Label
   ------------------------------------------------------------------------
   1    CUSTNUM     Char      8      0    $8.       $8.         CUSTOMER
   5    FIRSTORD    Num       8     75    DATE9.    DATE9.      FIRSTORDER
   4    NAME        Char     60     15    $60.      $60.        NAME
   2    STATE       Char      2      8    $2.       $2.         STATE
   3    ZIPCODE     Char      5     10    $5.       $5.         ZIPCODE
```

As you can see from the DATASETS procedure output, the VLib.UsaCust view descriptor has five variables: CustNum, FirstOrd, Name, State, and Zipcode. The variables are listed in alphabetic order, and the # column in the listing shows the order of each variable in VLib.UsaCust.

The **`Label`** field in the DATASETS procedure lists the names of the PC files columns. The FirstOrder column name has been truncated to the eight-character SAS variable name FirstOrd because SAS uses only the first eight characters of a PC files column name when it assigns a default SAS variable name. See "ASSIGN Statement" on page 73 for more information about how SAS variable names are assigned for PC files column names.

The information displayed by the DATASETS procedure does not include any selection criteria that might be specified for the view descriptor. To see selection criteria, you must review the code that created the view descriptor.

You cannot use the MODIFY statement in the DATASETS procedure to change the attributes of a view descriptor.

For more information about the DATASETS procedure, see *Base SAS Procedures Guide*.

Charting PC Files Data with Descriptors

PROC GCHART programs work with data that is described by view descriptors just as they do with other SAS data sets. (Using this procedure requires a SAS/GRAPH license at your site.) The following example uses the view descriptor VLib.AllOrdr to create a vertical bar chart of the number of orders per product. VLib.AllOrdr describes the data in the Orders.xls file.

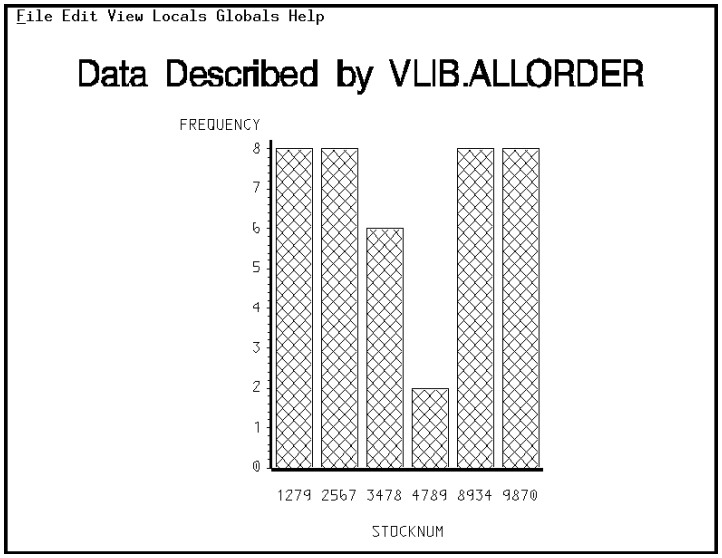
```
proc access dbms=xls;
   create adlib.order.access;
   /* create access descriptor   */
   path="c:\sasdemo\orders.xls";
   worksheet=sheet1;
   range='a1..j39';
   getnames=yes;
   scantype=5;
   mixed=yes;
   assign=yes;
   rename dateorderd = dateord
          processdby = procesby;
   format dateorderd date9.
          shipped date9.
          ordernum      5.0
          length        4.0
          stocknum      4.0
          takenby       6.0
          processdby    6.0
          fabcharges    12.2;
   list all;

   create vlib.allordr.view;
   /* create vlib.allordr view */
   select all;
run;

proc gchart data=vlib.allordr;
   /* example                    */
   vbar stocknum / discrete;
title 'Data Described by VLIB.ALLORDR';
run;
```

The following output shows the results for this example. StockNum represents each product. The number of orders for each product is represented by the height of the bar.

Output 21.2 Vertical Bar Chart of Number of Orders per Product



For more information about the GCHART procedure, see *SAS/GRAPH Reference, Volumes 1 and 2*.

Calculating Statistics with PC Files Descriptors

You can also use SAS statistical procedures on PC files data. This section shows examples using the FREQ, MEANS, and RANK procedures.

Using the FREQ Procedure

Suppose you want to find the percentages of your invoices that went to each country so that you can decide where to increase your overseas marketing. The following example uses the view descriptor VLib.Inv to calculate the percentage of invoices for each country that appears in the Invoice.dbf file.

```
proc access dbms=dbf;
   /* create access descriptor   */
   create adlib.invoice.access;
      path="c:\sasdemo\invoice.dbf";
   assign;
   rename invoicenum = invnum
          amtbilled = amtbilld ;
   format paidon      date9.
          invoicenum  5.0
          billedby    6.0;
   assign=yes;

   create vlib.inv.view;
   /* create vlib.inv view      */
   select invoicenum amtbilled
```

```
          country billedby paidon;
   list all;
run;

proc freq data=vlib.inv;
   /* example             */
   tables country;
   title 'Data Described by VLIB.INV';
run;
```

The following output shows the one-way frequency table that this example generates.

Output 21.3 Frequency Table for Variable Country Described by View Descriptor VLib.Inv

Data Described by VLIB.INV 6

COUNTRY

COUNTRY	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Argentina	2	11.76	2	11.76
Australia	1	5.88	3	17.65
Brazil	4	23.53	7	41.18
USA	10	58.82	17	100.00

For more information about the FREQ procedure, see *Step-by-Step Programming with Base SAS Software* and *Base SAS Procedures Guide*.

Using the MEANS Procedure

In your analysis of recent orders, suppose you want to calculate some statistics for each U.S. customer. From the Orders.xls file, the view descriptor VLib.UsaOrdr selects a subset of observations that have a ShipTo value beginning with a 1, indicating a U.S. customer.

Using the OUT= option in the SORT procedure, the data from the DBF file is extracted, placed in a SAS data file, and then sorted.

The following example generates the means and sums of the length of material ordered (in yards) and the fabric charges (in dollars) for each U.S. customer. Also included are the number of observations (N) and the number of missing values (NMiss). The MAXDEC= option specifies the number of decimal places (0-8) for PROC MEANS to use in printing the results.

```
proc access dbms=xls;
    create adlib.order.access;
    /* create access descriptor    */
    path="c:\sasdemo\orders.xls";
    worksheet=shee1;
    getnames=yes;
    skiprows=2;
    scantype=5;
    mixed=yes;
    assign=yes;
    rename dateorderd = dateord
           processdby = procesby;
    format dateorderd  date9.
```

```
           shipped     date9.
           ordernum    5.0
           length      4.0
           stocknum    4.0
           takenby     6.0
           processdby  6.0
           fabcharges  12.2;
    list all;

    create vlib.usaordr.view;
    /* create vlib.usaordr view  */
    select ordernum stocknum length
           fabcharges shipto;
    subset where shipto like '1%';
run;

proc sort data=vlib.usaordr out=work.usaorder;
    by shipto;
run;

proc means data=work.usaordr mean
     /* example             */
     sum n nmiss maxdec=0;
   by shipto;
   var length fabcharg;
title 'Data Described by VLIB.USAORDR';
run;
```

The following output shows the results for this example.

Output 21.4 PROC MEANS Statistics on Fabric Length and Charges for Each U.S. Customer

```
                        Data Described by VLIB.USAORDR                        7

-------------------------------- SHIPTO=14324742 --------------------------------


          Variable  Label               Mean           Sum   N  Nmiss
          ------------------------------------------------------------
          LENGTH    LENGTH              1095          4380   4      0
          FABCHARG  FABCHARGES       1934460       3868920   2      2
          ------------------------------------------------------------

-------------------------------- SHIPTO=14898029 --------------------------------


          Variable  Label               Mean           Sum   N  Nmiss
          ------------------------------------------------------------
          LENGTH    LENGTH              2500          5000   2      0
          FABCHARG  FABCHARGES       1400825       2801650   2      0
          ------------------------------------------------------------

-------------------------------- SHIPTO=15432147 --------------------------------


          Variable  Label               Mean           Sum   N  Nmiss
          ------------------------------------------------------------
          LENGTH    LENGTH               725          2900   4      0
          FABCHARG  FABCHARGES        252149        504297   2      2
          ------------------------------------------------------------

-------------------------------- SHIPTO=18543489 --------------------------------


          Variable  Label               Mean           Sum   N  Nmiss
          ------------------------------------------------------------
          LENGTH    LENGTH               303          1820   6      0
          FABCHARG  FABCHARGES      11063836      44255344   4      2
          ------------------------------------------------------------

-------------------------------- SHIPTO=19783482 --------------------------------


          Variable  Label               Mean           Sum   N  Nmiss
          ------------------------------------------------------------
          LENGTH    LENGTH               450          1800   4      0
          FABCHARG  FABCHARGES        252149       1008594   4      0
          ------------------------------------------------------------

-------------------------------- SHIPTO=19876078 --------------------------------


          Variable  Label               Mean           Sum   N  Nmiss
          ------------------------------------------------------------
          LENGTH    LENGTH               690          1380   2      0
          FABCHARG  FABCHARGES            .             .   0      2
          ------------------------------------------------------------
```

For more information about the MEANS procedure, see *Base SAS Procedures Guide*.

Using the RANK Procedure

You can use advanced statistical procedures on PC files data. The following example uses the RANK procedure to calculate the order of birthdays for a set of employees who are listed in the Employees.dbf file. The OUT= option creates a SAS data file, DLib.RankExam, from the view descriptor VLib.Emps so that the data in the SAS file

can be sorted by the SORT procedure. The RANKS statement assigns the name DateRank to the new variable (in the SAS data file) that is created by the procedure. The PRINT procedure then prints the data that is described by DLib.RankExam. You can also use the PRINT procedure to print all or some of the PC file data values described by view descriptors.

```
proc access dbms=dbf;
   create adlib.employ.access;
   /* create access descriptor  */
   path="c:\sasdemo\employees";
   drop salary;
   list all;

   create vlib.emps.view;
   /* create vlib.emps view     */
   select empid jobcode birthdate
      lastname jobcode;
   format birthdate date9.
          empid     6.0;
   subset where jobcode=602;
run;

proc rank data=vlib.emps out=dlib.rankexam;
   /* example          */
   var birthdat;
   ranks daterank;
run;

proc sort data=dlib.rankexam;
   by lastname;
run;

proc print data=dlib.rankexam(drop=jobcode);
   title 'Order of Dept 602 Employee Birthdays';
run;
```

Data stored in the DBF file must be extracted and placed in a SAS data set before it can be sorted with a SAS procedure. (This restriction also applies to data from other PC files.) The DROP= data set option is used in the PROC PRINT statement because the JobCode variable is not needed in the output. The JobCode variable is required in the SELECT statement so it can be used in the WHERE statement. The JobCode variable is then included in the view descriptor, even though it is not needed in the output. The following output shows the result of this example.

Output 21.5 Ranking of Employee Birthdays with PROC RANK

Order of Dept 602 Employee Birthdays

OBS	EMPID	BIRTHDAT	LASTNAME	DATERANK
1	456910	24SEP1953	ARDIS	5
2	237642	13MAR1954	BATTERSBY	6
3	239185	28AUG1959	DOS REMEDIOS	7
4	321783	03JUN1935	GONZALES	2
5	120591	12FEB1946	HAMMERSTEIN	4
6	135673	21MAR1961	HEMESLY	8
7	456921	12MAY1962	KRAUSE	9
8	457232	15OCT1963	LOVELL	11
9	423286	31OCT1964	MIFUNE	12
10	216382	24JUL1963	PURINTON	10
11	234967	21DEC1967	SMITH	13
12	212916	29MAY1928	WACHBERGER	1
13	119012	05JAN1946	WOLF-PROVENZA	3

When you use the PRINT procedure, you might want to take advantage of the SAS data set option OBS=, which enables you to limit the number of observations to be processed. This option is especially useful when the view descriptor describes a large amount of data, the SAS data file is large, or when you just want to see an example of the output. You cannot use OBS= if the view descriptor contains a WHERE clause in the SUBSET statement.

For more information about RANK, about other advanced statistical procedures, and about the PRINT procedure, see *Base SAS Procedures Guide*. For more information about the OBS= and FIRSTOBS= options, see *SAS Language Reference: Dictionary*.

Selecting and Combining PC Files Data with Descriptors

For many of your SAS programs, you might need to combine data from more than one view descriptor or to manipulate data that is accessed by a specific view descriptor. The following sections describe how you can select and combine data using the following language elements:

- the WHERE statement in a DATA step
- the SQL procedure to create a new PROC SQL view
- the SQL procedure to join data from various sources
- a summary function in a PROC SQL query to create a new column in the output.

Using the WHERE Statement

Suppose you have a view descriptor VLib.AllInv that lists invoices for all customers. VLib.AllInv is based on the Invoice.dbf file. You can use a SET statement to create a SAS data file that contains information on customers who have not paid their bills and whose bills amount to at least $300,000.

```
proc access dbms=dbf;
   create adlib.invoice.access;
   /* create access descriptor   */
   path="c:\sasdemo\invoice.dbf";
   assign=yes;
   rename invoicenum = invnum
```

```
          amtbilled  = amtbilld
          amountinus = amtinus;
   format paidon date9.
          billedon date9.
          invoicenum  5.0
          billedby    6.0
          amtbilled   15.2
          amountinus  15.2;
   list all;

   create vlib.allinv.view;
   /* create vlib.allinv view  */
   select all;
run;

data notpaid(keep=invnum billedto amtinus billedon);
    /* example  */
   set vlib.allinv;
   where paidon is missing and amtinus>=300000;
run;
```

In the DATA step's WHERE statement, be sure to use SAS variable names, not PC files column names. The following output shows the result of the new temporary SAS data file Work.NotPaid.

```
proc print data=notpaid;
   format amtinus dollar20.2;
title 'High Bills--Not Paid';
run;
```

Output 21.6 Word.NotPaid Data File Created Using a SAS WHERE Statement

High Bills--Not Paid

OBS	INVNUM	BILLEDTO	AMTINUS	BILLEDON
1	11271	18543489	$11,063,836.00	05OCT1998
2	12102	18543489	$11,063,836.00	17NOV1998
3	11286	43459747	$11,063,836.00	10OCT1998
4	12051	39045213	$2,256,870.00	02NOV1998
5	12471	39045213	$2,256,870.00	27DEC1998
6	12476	38763919	$2,256,870.00	24DEC1998

The first line of the DATA step uses the KEEP= data set option. This option works with view descriptors just as it works with other SAS data sets; it specifies that you want to include only the listed variables in the new SAS data file Work.NotPaid. However, you can still use the other view descriptor variables in other statements within the DATA step.

The SAS WHERE statement includes two conditions to be met. First, it selects only observations that have a missing value for the PAIDON variable. Second, it requires that the amount in each bill be higher than a certain figure. You need to be familiar with the PC files data so that you can determine reasonable values for these expressions. For information about the SAS WHERE statement, refer to *SAS Language: Reference.*

Using the SQL Procedure

The SQL procedure implements the Structured Query Language in SAS. The SQL procedure follows the SQL convention of using the terms column and row for variable and observation.

Joining Data from Various Sources

The SQL procedure provides another way to select and combine data. For example, suppose you have three data sets: two view descriptors, VLib.CusPhon and VLib.CusOrdr, which are based on the Customers.wk3 and Orders.xls files, respectively, and a SAS data file, DLib.OutOfStk, which contains product names and numbers that are out of stock. You can use the SQL procedure to create a view that joins the data from these three sources and displays their output. The SAS WHERE or subsetting IF statements would not be appropriate in this case because you want to compare variables from several sources, rather than simply merging or concatenating the data.

The following SAS statements select and combine data from the view descriptors and the SAS data file to create an SQL view, SLib.BadOrdr. SLib.BadOrdr retrieves customer and product information that the sales department uses to notify customers of unavailable products.

```
proc access dbms=wk3;
   create adlib.customr.access;
   /* create access descriptor  */
   path="c:\sasdemo\customers.wk3";
   worksheet=v;
   range='cus_phone';
   getnames=yes;
   skiprows=2;
   scantype=5;
   mixed=yes;
   list all;

   create vlib.cusphon.view;
   /* create vlib.cusphon view */
   select customer phone name;
   rename customer=custnum;
run;

proc access dbms=xls;
   create adlib.orders.access;
   /* create access descriptor  */
   path="c:\sasdemo\orders.xls";
   worksheet='sheet1';
   range='a1..j39';
   getnames=yes;
   skiprows=2;
   scantype=5;
   mixed=yes;
   list all;

   create vlib.cusordr.view;
   /* create vlib.cusordr view  */
   select ordernum stocknum shipto;
```

```
   rename ordernum ordnum;
   format ordernum 5.0
          stocknum 4.0;
run;

proc sql;
   /* example              */
   create view slib.badordr as
      select distinct cusphon.custnum,
             cusphon.name, cusphon.phone,
             cusordr.stocknum,
              outofstk.fibernam
             as product
         from vlib.cusphon, vlib.cusordr,
              dlib.outofstk
         where cusordr.stocknum=
           outofstk.fibernum
              and cusphon.custnum=
                cusordr.shipto;
```

The CREATE VIEW statement incorporates a WHERE clause as part of its SELECT clause. The DISTINCT keyword eliminates any duplicate rows of customer numbers that occur when companies order an unavailable product more than once.

It is recommended that you *not* include an ORDER BY clause in a CREATE VIEW statement. Doing so causes the output data to be sorted every time the PROC SQL view is submitted, which can have a negative impact on performance. It is more efficient to add an ORDER BY clause to a SELECT statement that displays your output data, as shown below.

```
options linesize=120;
title 'Data Described by SLIB.BADORDR';

select * from slib.badordr
   order by custnum, product;
quit;
```

This SELECT statement uses the SQL view SLib.BadOrdr to display joined WK3, XLS, and SAS data in ascending order by the CustNum column and then by the Product (that is, FiberNam) column. The data is ordered by Product because one customer might have ordered more than one product. To select all the columns from the view, use an asterisk (*) in place of column names. When an asterisk is used, the columns are displayed in the order specified in the SLib.BadOrdr view. The following output shows the data that is described by the SLib.BadOrdr view.

Output 21.7 Data Described by the SQL View SLib.BadOrdr

Data Described by SLIB.BADORDER

CUSTOMER	NAME	PHONE	STOCKNUM	PRODUCT
15432147	GREAT LAKES LABORATORY EQUIPMENT MANUFACTURERS	616/582-3906	4789	dacron
18543489	LONE STAR STATE RESEARCH SUPPLIERS	512/478-0788	8934	gold
29834248	BRITISH MEDICAL RESEARCH AND SURGICAL SUPPLY	(0552)715311	3478	olefin
31548901	NATIONAL COUNCIL FOR MATERIALS RESEARCH	406/422-3413	8934	gold
43459747	RESEARCH OUTFITTERS	03/734-5111	8934	gold

Although the query uses SAS variable names like CustNum, you might notice that the output uses PC files column names like Customer. By default, PROC SQL displays SAS variable labels, which default to PC files column names. (You can use the NOLABEL option to change this default.)

Creating New Columns with the GROUP BY Clause

Instead of creating a new PROC SQL view, you might want to summarize your data and create new columns in a report. Although you cannot use the ACCESS procedure to create new columns, you can easily do this by using the SQL procedure with data that is described by a view descriptor.

This example uses the SQL procedure to retrieve and manipulate data from the view descriptor VLib.AllEmp, which is based on the Employee.dbf file. When this *query* (as a SELECT statement is often called) is submitted, it calculates and displays the average salary for each department. The query enables you to manipulate your data and display the results without creating a SAS data set.

Because this example reports on employees' salaries, the view descriptor VLib.AllEmp is assigned a SAS password (MONEY) using the DATASETS procedure. Because of the READ= level of protection, the password must be specified in the PROC SQL SELECT statement before you can see the DBF file data that is accessed by Work.AllEmp.

In the following example, the DISTINCT keyword in the SELECT statement removes duplicate rows. The AVG function in the SQL procedure is equivalent to the SAS MEAN function.

```
options linesize=80;

proc access dbms=dbf;
   /* create access descriptor */
   create adlib.employ.access;
   path="c:\sasdemo\employee.dbf";
   assign=yes;
   format empid     6.0
          salary    dollar12.2
          jobcode   5.0
          birthdate date9.
          hiredate  date9.;
   list all;
run;

/* create work.allemp view  */
proc access dbms=dbf
            accdesc=adlib.employ;
   create work.allemp.view;
   select all;
run;

/* assign a password */
proc datasets library=work  memtype=view;
   modify allemp (read=money);
run;

/* example */
   title 'Average Salary Per ACC Department';

proc sql;
```

```
   select distinct dept,
          avg(salary) label='Average Salary'
           format=dollar12.2
      from work.allemp(pw=money)
      where dept like 'ACC%'
      group by dept;
quit;
```

The columns are displayed in the order specified in the SELECT clause of the query. The following output shows the result of the query.

Output 21.8 Data Retrieved by an SQL Procedure Query

```
              Average Salary Per ACC Department

                              Average
                    DEPT       Salary
                    --------------------
                    ACC013    $54,591.33
                    ACC024    $55,370.55
                    ACC043    $75,000.34
```

To delete a password on an access descriptor or any SAS data set, put a slash after the password:

```
/* delete the password */
proc datasets library=work  memtype=view;
   modify allemp (read=money/);
run;
```

For more information about SAS passwords, see "SAS Passwords for Descriptors" on page 68.

Using the SAS Viewer on PC Files Data

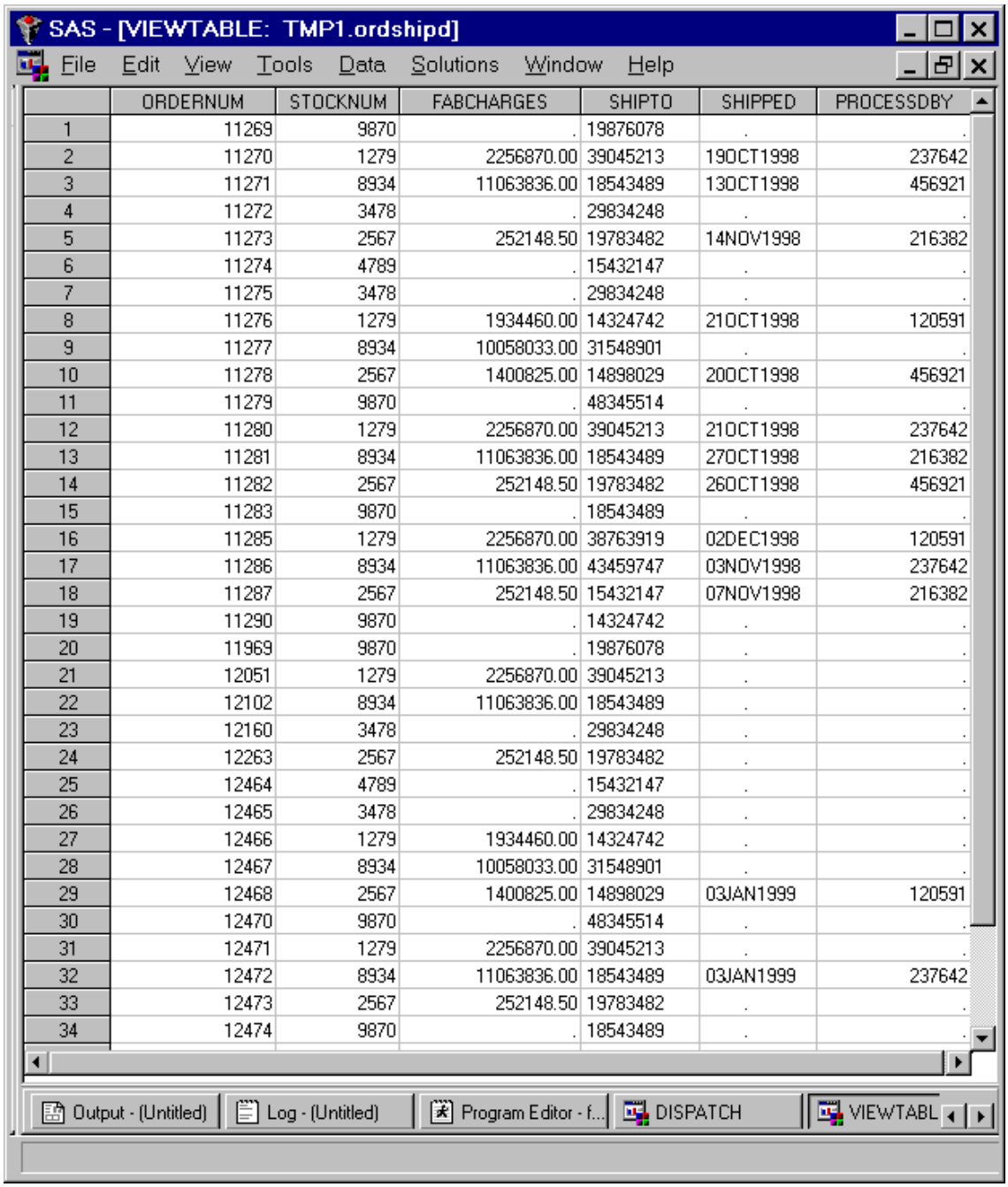
While your DBF data is displayed in an FSVIEW window, for example, you can save it to a data file and then re-open that file using the SAS Viewer (VIEWTABLE window). Take these steps to save your FSVIEW output to a data file:

1. Select the SAS icon in the top, left corner and then select the **Menu** item. Doing so opens a listing of menus.
2. Select the **File** menu and then the **Save As** item.
3. A Save As window opens and asks you for the directory and filename information for the file that you want to save. In the **Save as type** field, click the down arrow to select **Data Files**.
4. Click Save. In this example, the output is stored to a file named VLib.OrdShp.

When your file is saved, you can go to the SAS Explorer window and double-click the *libref.name* of your new file, in this case, VLib.OrdShp. Doing so opens the VIEWTABLE window, as shown in the following display:

You can browse or edit the PC files data from the VIEWTABLE window. For information about using this window, select the **Using this Window** item from the **Help** menu.

Display 21.1 Browsing or Editing Data through the VIEWTABLE Window



	ORDERNUM	STOCKNUM	FABCHARGES	SHIPTO	SHIPPED	PROCESSDBY
1	11269	9870	.	19876078	.	.
2	11270	1279	2256870.00	39045213	19OCT1998	237642
3	11271	8934	11063836.00	18543489	13OCT1998	456921
4	11272	3478	.	29834248	.	.
5	11273	2567	252148.50	19783482	14NOV1998	216382
6	11274	4789	.	15432147	.	.
7	11275	3478	.	29834248	.	.
8	11276	1279	1934460.00	14324742	21OCT1998	120591
9	11277	8934	10058033.00	31548901	.	.
10	11278	2567	1400825.00	14898029	20OCT1998	456921
11	11279	9870	.	48345514	.	.
12	11280	1279	2256870.00	39045213	21OCT1998	237642
13	11281	8934	11063836.00	18543489	27OCT1998	216382
14	11282	2567	252148.50	19783482	26OCT1998	456921
15	11283	9870	.	18543489	.	.
16	11285	1279	2256870.00	38763919	02DEC1998	120591
17	11286	8934	11063836.00	43459747	03NOV1998	237642
18	11287	2567	252148.50	15432147	07NOV1998	216382
19	11290	9870	.	14324742	.	.
20	11969	9870	.	19876078	.	.
21	12051	1279	2256870.00	39045213	.	.
22	12102	8934	11063836.00	18543489	.	.
23	12160	3478	.	29834248	.	.
24	12263	2567	252148.50	19783482	.	.
25	12464	4789	.	15432147	.	.
26	12465	3478	.	29834248	.	.
27	12466	1279	1934460.00	14324742	.	.
28	12467	8934	10058033.00	31548901	.	.
29	12468	2567	1400825.00	14898029	03JAN1999	120591
30	12470	9870	.	48345514	.	.
31	12471	1279	2256870.00	39045213	.	.
32	12472	8934	11063836.00	18543489	03JAN1999	237642
33	12473	2567	252148.50	19783482	.	.
34	12474	9870	.	18543489	.	.

Reading and Updating PC Files Data with the SQL Procedure

The SQL procedure enables you to retrieve data from PC files and update data in DBF files. You can read and display PC files data by specifying a view descriptor or other SAS data set in the SQL procedure's SELECT statement.

To update DBF data, you can specify view descriptors in the SQL procedure's INSERT, DELETE, and UPDATE statements. You can also use these statements to

modify SAS data files. However, the ability to update data in a DBF file is subject to the following conditions:

- As in other PROC and DATA steps, you can use only a view descriptor or other SAS data set in an SQL procedure statement, not an access descriptor.
- If you did not create the DBF file, you must be granted the appropriate file access privileges before you can select, insert, delete, or update the data.
- You must also be granted the appropriate file access privileges before you select the data from MDB, DIF, WK*n*, or XLS files. The SAS/ACCESS interface to these files is read-only, so the SELECT statement is the only one of the four PROC SQL statements (in this section) that can reference a view descriptor based on MDB, DIF, WK*n*, or XLS data.

A summary of some of the SQL procedure statements follows:

SELECT	retrieves, manipulates, and displays PC files data that is described by a view descriptor. SELECT can also use data that is described by a PROC SQL or DATA step view, or data in a SAS data file. A SELECT statement is usually referred to as a *query* because it queries the table for information.
DELETE	deletes rows from a SAS data file or from a DBF file that is described by a view descriptor. When you reference a view descriptor that is based on a DBF file in the DELETE statement, the records in the DBF file are marked for deletion.
INSERT	inserts rows into a DBF file or a SAS data file.
UPDATE	updates the data values in a DBF file or in a SAS data file.

Because the SQL procedure is based on the Structured Query Language, it works somewhat differently than some SAS procedures. For example, the SQL procedure executes without a RUN statement when a procedure statement is submitted. The SQL procedure also displays any output automatically without using the PRINT procedure.

By default, PROC SQL uses the LABEL option to display output. LABEL displays SAS variable labels, which default to PC files column names. If you prefer to use SAS variable names in your output, specify NOLABEL in the OPTIONS statement.

For more information about this procedure, its options, and example, see the SQL procedure chapter in *Base SAS Procedures Guide*.

Reading Data with the SQL Procedure

You can use the SQL procedure's SELECT statement to display PC files data that is described by a view descriptor or by another SAS data set. In the following example, the query uses the VLib.Product view descriptor to retrieve a subset of the data in the SpecProd.dif file.

The asterisk (*) in the SELECT statement indicates that all the columns in VLib.Product are retrieved. The WHERE clause retrieves a subset of the rows. The ORDER BY clause causes the data to be presented in ascending order according to the table's FiberName column.

```
proc access dmbs=dif;
   create adlib.product.access;
   /* create access descriptor  */
   path="c:\sasdemo\specprod.dif";
   diflabel;
   assign=yes;
   rename productid prodid;
```

```
   format productid  4.
          weight     e16.9
          fibersize  e20.13
          width      e16.9;
   list all;

   create vlib.product.view;
   /* create view descriptor   */
   select all;
   list view;
 run;

options nodate linesize=120;
  title 'DIF File Data Retrieved with a SELECT
         Statement';

proc sql;
  select *
  from vlib.product
  where cost is not null
  order by fibernam;
quit;
```

The following output displays the query's results. Notice that the SQL procedure displays the DIF file's column names, not the SAS variable names.

Output 21.9 PC Files Data Retrieved with a PROC SQL Query

DIF File Data Retrieved with a SELECT Statement

PRODUCTID	WEIGHT	FIBERNAME	FIBERSIZE	COST	PERUNIT	WIDTH
1279	1.278899910E-01	asbestos	6.3476000000000E-10	1289.64	m	2.227550050E+02
2567	1.258500220E-01	fiberglass	5.1880000000000E-11	560.33	m	1.205000000E+02
8934	1.429999950E-03	gold	2.3800000000000E-12	100580.33	cm	2.255999760E+01

Updating Data with the SQL Procedure

You can use the SQL procedure's UPDATE statement to update the data in a DBF file, as illustrated by the following example. Because you are referencing a view descriptor, you use the SAS variable names in the UPDATE statement. However, the SQL procedure displays the DBF column names.

```
proc sql;
 update vlib.empeeoc
   set salary=26678.24,
       gender='M',
       birthdat='28AUG1959'd
   where empid=123456;

 options linesize=120;
 title 'Updated Data in EMPLOYEES Table';
 select empid, hiredate, salary, dept, jobcode,
```

```
        gender, birthdat, lastname
    from vlib.empeeoc
    where empid=123456;
quit;
```

The following output displays the updated row of data retrieved from the view descriptor VLib.EmpEEoc.

Output 21.10 DBF File Data Updated with the UPDATE Statement

```
                          Updated Data in EMPLOYEES Table

 EMPID  HIREDATE        SALARY  DEPT    JOBCODE  GENDER  BIRTHDATE  LASTNAME
-------------------------------------------------------------------------------
123456   04APR1989    $26,678.24  ACC043     1204  M         28AUG1959  VARGAS
```

Deleting Data with the SQL Procedure

You can use the SQL procedure's DELETE statement to delete rows from a DBF file. In the following example, the row that contains the value 346,917 in the EmpID column is deleted from the Employee.dbf.

```
proc sql;
  delete from vlib.empeeoc
   where empid=346917;
quit;
```

The deleted observation is marked with an asterisk (*) in the DELETE_FLG field. This is the only indicator you have that a record in a DBF field has been marked for deletion. If you have a number of rows to delete, you could use a macro variable EmpID instead of the individual EmpID values. Doing so would enable you to change the values more easily.

```
%let empid=346917;

proc sql;
  delete from vlib.empeeoc
   where empid=&empid;
quit;
```

CAUTION:

Use a WHERE clause in the DELETE statement. If you omit the WHERE clause from the DELETE statement, you delete *all* the data in the SAS data file or DBF file. △

Inserting Data with the SQL Procedure

You can use the SQL procedure's INSERT statement to add rows to a DBF file. In the following example, the row that contains the value 346,917 in the EmpID column is inserted back into the Employee.dbf file.

```
proc sql;
  insert into vlib.allemp
   values('',346917,'02MAR1987'd,46000.33,'SHP013',204,
          'F','15MAR1950'd,'SHIEKELESLAM','SHALA',
```

```
          'Y.','8745');
quit;
```

A message is written to the SAS log to indicate that the row has been inserted, as shown in the following output:

Output 21.11 Message Displayed in the SAS Log When a Row Is Inserted

```
6698
6699
6700
6701
6702
6703  proc sql;
6704  insert into vlib.allemp
6705     values('',346917,'02MAR1987'd,46000.33,
6706            'SHP013',204,'F','15MAR1950'd,
6707            'SHIEKELESLAM','SHALA','Y.',
6708            '8745');

NOTE: 1 row was inserted into VLIB.ALLEMP.

6709  quit;
```

Updating PC Files Data with the MODIFY Statement

The MODIFY statement extends the capabilities of the DATA step by enabling you to modify DBF file data that is accessed by a view descriptor or a SAS data file without creating an additional copy of the data. To use the MODIFY statement with a view descriptor, you must have update privileges on the view's underlying DBF file.

You can specify either a view descriptor or a SAS data file as the master data set in the MODIFY statement. In the following example, the master data set is the view descriptor VLib.Master, which describes data in the Orders.dbf file. You also create a transaction data file, DLib.Trans, that you use to update the master data set (and therefore, the Orders.dbf table). The SAS variable names, formats, and informats of the transaction data file must correspond to those described by the view descriptor VLib.Master.

Using the VLib.Master view descriptor, the MODIFY statement updates the Orders.dbf table with data from the DLib.Trans data file. SAS reads one observation (or row) of the Orders.dbf table for each iteration of the DATA step, and performs any operations that the code specifies. In this case, the IF-THEN statements specify whether the information for an order is to be updated, added, or deleted.

```
proc access dmbs=dbf;
   /* create access descriptor  */
   create adlib.orders.access;

   path="c:\sasdemo\orders.dbf";
   assign=yes;
   rename dateorderd = dateord;
          processdby = procesby;
   format dateorderd date9.
          shipped date9.
```

```
          ordernum      5.0
          length        4.0
          stocknum      4.0
          takenby       6.0
          processdby    6.0
          fabcharges    12.2;

   /* create vlib.master view   */
   create vlib.master.view;
   select all;
run;

data dlib.trans;
   ordernum=12102;
   /*Obs. 1 specifies Update for ORDERNUM=12102*/
      shipped='05DEC1998'd;
      type='U';
      output;

   ordernum=12160;
   /*Obs. 2 specifies Update for ORDERNUM=12160*/
      shipped=.;
      takenby=456910;
      type='U';
      output;

   ordernum=13000;
   /*Obs. 3 specifies Add for new ORDERNUM=13000*/
      stocknum=9870;
      length=650;
      fabcharg=.;
      shipto='19876078';
      dateord='18JAN1999'd;
      shipped='29JAN1999'd;
      takenby=321783;
      procesby=120591;
      specinst='Customer agrees to certain
                limitations.';
      type='A';
      output;

   ordernum=12465;
   /*Obs. 4 specifies Delete for
     ORDERNUM=12465*/
      type='D';
      output;
run;

data vlib.master;
   /* MODIFY statement example */
   modify vlib.master dlib.trans;
   by ordernum;
   select (_iorc_);
      when (%sysrc(_dsenmr)) do;
```

```
   /* No match in MASTER - Add */
        if type='A'
           then output vlib.master;
        _error_ = 0;
      end;
      when (%sysrc(_sok)) do;
   /* Match located - Update or Delete */
        if type='U'
          then replace vlib.master;
        else if type='D'
          then remove vlib.master;
      end;
      otherwise do;
   /* Traps unexpected outcomes */
        put 'Unexpected ERROR condition:
            _IORC_ = ' _iorc_ ;
        put _all_;
   /* This dumps all vars in the PDV */
         _error_ = 0;
      end;
   end;
run;

options linesize=120;
/* prints the example's output */

proc print data=vlib.master;
   where ordernum in(12102 12160 13000 12465);
   title 'DBF File Data Updated with the MODIFY
          Statement';
run;
```

The DATA step uses the SYSRC macro to check the value of the _IORC_ automatic variable. It also prevents an error message from being generated when no match is found in the VLib.Master file for an observation that is being added. It prevents the error message by resetting the _ERROR_ automatic variable to 0. The PRINT procedure specifies a WHERE statement so that it displays only the observations that are included in the transaction data set. The observation with OrderNum **12465** is deleted by the MODIFY statement, so it does not appear in the results. The results of this example are shown in the following output.

Output 21.12 Revising PC Files Data with a MODIFY Statement

DBF File Data Updated with the MODIFY Statement

OBS	DELETE_F	ORDERNUM	STOCKNUM	LENGTH	FABCHARG	SHIPTO	DATEORD	SHIPPED	TAKENBY	PROCESBY
22		12102	8934	110	11063836.00	18543489	15NOV1998	05DEC1998	456910	.
23		12160	3478	1000	.	29834248	19NOV1998	.	456910	.
26	*	12465	3478	1000	.	29834248	23DEC1998	.	234967	.
39		13000	9870	650	.	19876078	18JAN1999	29JAN1999	321783	120591

OBS	SPECINST
22	
23	Customer agrees to pay in full.
26	
39	Customer agrees to certain limitations.

In this example, any column value that you specify in the transaction data set carries over to any subsequent observations if the values for the subsequent observations are missing. For example, the first observation sets the value of Shipped to **05DEC1998**. The second observation sets the value to missing. If the value of Shipped was not set to missing in the second observation, the value **05DEC1998** would be incorrectly supplied. Therefore, you might want to create your transaction data set in a particular order to minimize having to reset variables.

There are some differences in the way you use a MODIFY statement to update a SAS data file and to update DBF file data through a view descriptor. When you use a view descriptor as the master data set in a MODIFY statement, the following conditions apply:

- You cannot use the POINT= option because observation numbers are not available in a DBF file.
- The NOBS= option displays the largest positive integer value available on the host operating system.
- Each PC files statement that is issued, whether an INSERT, DELETE, or UPDATE, is a separate transaction and is saved in the DBF file. You cannot undo (or reverse) these changes without re-editing.

For more information about the MODIFY statement, see *SAS Language Reference: Dictionary*.

Updating a SAS Data File with PC Files Data

You can update a SAS data file with PC file data that is described by a view descriptor just as you can update a SAS data file with data from another SAS data file.

Suppose you have a SAS data set, DLib.Birthday, that contains employee ID numbers, last names, and birthdays. (See Appendix 1, “Sample Data,” on page 243 for a description of DLib.Birthday.) You want to update this data set with data described by VLib.EmpBday, a view descriptor that is based on the Employee.dbf file. To perform this update, enter the following SAS statements:

```
options linesize=80;

proc access dbms=dbf;
   create adlib.employ.access;
   /* create access descriptor   */
```

```
   path="c:\sasdemo\employee.dbf";
   assign=yes;
   format empid 6.
          salary dollar12.2
          jobcode 5.
          hiredate date9.
          birthdate date9.;
   list all;

   create vlib.empbday.view;
   /* create view descriptor   */
   select empid birthdate lastname
          firstname phone;
run;

proc sort data=dlib.birthday;
   by lastname;
run;

proc print data=dlib.birthday;
   /* examples                */
   format birthdat date9.;
   title 'DLIB.BIRTHDAY Data File';
run;

proc print data=vlib.empbday;
   format birthdat date9.;
   title 'Data Described by VLIB.EMPBDAY';
run;

proc sort data=vlib.empbday out=work.empbirth;
   by lastname;
run;

data dlib.newbday;
   update dlib.birthday work.empbirth;
   by lastname;
run;

proc print;
   format birthdat date9.;
   title 'DLIB.NEWBDAY Data File';
run;
```

In this example, a PROC SORT statement with the OUT= option extracts DBF file data, places it in the SAS data file Work.EmpBirth, and sorts the data by the LastName variable. (When you use a DATA step, PC files data must be extracted before you can sort it.) When the UPDATE statement references the SAS data file Work.EmpBirth and you use a BY statement in the DATA step, the BY statement causes the interface view engine to generate an ORDER BY clause for the variable LastName. Thus, the ORDER BY clause causes the DBF data to be presented to SAS in sorted order for use in updating the DLib.NewBday data file. However, the SAS data file DLib.Birthday must be sorted before the update because the UPDATE statement expects both the original file and the transaction file to be sorted by the same BY variable.

The following outputs show the results of the PRINT procedures.

Output 21.13 Data File to Be Updated, DLib.Birthday

DLIB.BIRTHDAY Data File

OBS	EMPID	BIRTHDAT	LASTNAME
1	254196	06APR1949	CHANG
2	459288	05JAN1934	JOHNSON
3	127815	25DEC1943	WOLOSCHUK

Output 21.14 DBF File Data Described by the View Descriptor VLib.EmpBday

Data Described by VLIB.EMPBDAY

OBS	EMPID	BIRTHDAT	LASTNAME	FIRSTNAM	PHONE
1	119012	05JAN1946	WOLF-PROVENZA	G.	3467
2	120591	12FEB1946	HAMMERSTEIN	S.	3287
3	123456	28AUG1959	VARGAS	CHRIS	
4	127845	25DEC1943	MEDER	VLADIMIR	6231
5	129540	31JUL1960	CHOULAI	CLARA	3921
6	135673	21MAR1961	HEMESLY	STEPHANIE	6329
7	212916	29MAY1928	WACHBERGER	MARIE-LOUISE	8562
8	216382	24JUL1963	PURINTON	PRUDENCE	3852
9	234967	21DEC1967	SMITH	GILBERT	7274
10	237642	13MAR1954	BATTERSBY	R.	8342
11	239185	28AUG1959	DOS REMEDIOS	LEONARD	4892
12	254896	06APR1949	TAYLOR-HUNYADI	ITO	0231
13	321783	03JUN1935	GONZALES	GUILLERMO	3642
14	328140	02JUN1951	MEDINA-SIDONIA	MARGARET	5901
15	346917	15MAR1950	SHIEKELESLAM	SHALA	8745
16	356134	25OCT1960	DUNNETT	CHRISTINE	4213
17	423286	31OCT1964	MIFUNE	YUKIO	3278
18	456910	24SEP1953	ARDIS	RICHARD	4351
19	456921	12MAY1962	KRAUSE	KARL-HEINZ	7452
20	457232	15OCT1963	LOVELL	WILLIAM	6321
21	459287	05JAN1934	RODRIGUES	JUAN	5879
22	677890	24APR1965	NISHIMATSU-LYNCH	CAROL	6245
23	346917	15MAR1950	SHIEKELESLAM	SHALA	

Output 21.15 Data in the Updated Data File DLib.NewBday

DLIB.NEWBDAY Data File

OBS	EMPID	BIRTHDAT	LASTNAME	FIRSTNAM	PHONE
1	456910	24SEP1953	ARDIS	RICHARD	4351
2	237642	13MAR1954	BATTERSBY	R.	8342
3	254196	06APR1949	CHANG		
4	129540	31JUL1960	CHOULAI	CLARA	3921
5	239185	28AUG1959	DOS REMEDIOS	LEONARD	4892
6	356134	25OCT1960	DUNNETT	CHRISTINE	4213
7	321783	03JUN1935	GONZALES	GUILLERMO	3642
8	120591	12FEB1946	HAMMERSTEIN	S.	3287
9	135673	21MAR1961	HEMESLY	STEPHANIE	6329
10	459288	05JAN1934	JOHNSON		
11	456921	12MAY1962	KRAUSE	KARL-HEINZ	7452
12	457232	15OCT1963	LOVELL	WILLIAM	6321
13	127845	25DEC1943	MEDER	VLADIMIR	6231
14	328140	02JUN1951	MEDINA-SIDONIA	MARGARET	5901
15	423286	31OCT1964	MIFUNE	YUKIO	3278
16	677890	24APR1965	NISHIMATSU-LYNCH	CAROL	6245
17	216382	24JUL1963	PURINTON	PRUDENCE	3852
18	459287	05JAN1934	RODRIGUES	JUAN	5879
19	346917	15MAR1950	SHIEKELESLAM	SHALA	8745
20	234967	21DEC1967	SMITH	GILBERT	7274
21	254896	06APR1949	TAYLOR-HUNYADI	ITO	0231
22	123456	28AUG1959	VARGAS	CHRIS	
23	212916	29MAY1928	WACHBERGER	MARIE-LOUISE	8562
24	119012	05JAN1946	WOLF-PROVENZA	G.	3467
25	127815	25DEC1943	WOLOSCHUK		

Appending Data with the APPEND Procedure

You can append data from any data set to a SAS data file or view descriptor. Specifically, you can append PC files data that is described by one view descriptor to another, or you can append a SAS data file. Because the SAS/ACCESS interface to MDB, DIF, WK*n*, and XLS files is read-only, you cannot append data *to* those files. You can however, append data *from* them to a DBF file or to a SAS data file.

The following example uses the APPEND procedure's FORCE option to append a SAS data file with extra variables to the data file referenced by the view descriptor VLib.SqlEmps. You must have DBF insert privileges in order to add rows to the Employees.dbf file.

You can append data to a table that is referenced by a view descriptor even if the view descriptor contains a subset of columns and a subset of rows. If a PC files column is defined as NOT NULL, some restrictions apply when appending data. For more information, see the APPEND procedure in *Base SAS Procedures Guide*.

The FORCE option forces PROC APPEND to concatenate two data sets even though they might have some different variables or variable attributes. The SAS data file, DLib.TempEmps, has Dept, FamilyID, and Gender variables that have not been selected in the view descriptor VLib.SqlEmps. The extra variables are dropped from DLib.TempEmps when it and the BASE= data set, VLib.SqlEmps, are concatenated. A message is displayed in the SAS log indicating that the variables are dropped.

```
proc access dbms=dbf;
  /* create access descriptor    */
  create adlib.employ.access;
```

```
  path='c:\sasdemo\employee.dbf';
  assign=no;
  drop salary;
  list all;

  create vlib.sqlemps.view;
  /* create view descriptor    */
  select empid hiredate lastname
         firstname middlename;
  format empid 6.0
         hiredate date9.;
run;

proc print data=vlib.sqlemps;
  /* examples                  */
  title 'Data Described by VLIB.SQLEMPS';
run;

proc print data=dlib.tempemps;
  title 'Data in DLIB.TEMPEMPS Data File';
run;
```

The view descriptor VLib.SqlEmps is displayed in the following output, and the SAS data file DLib.Temps is displayed in Output 21.17.

Output 21.16 Data Described by VLib.SqlEmps

Data Described by VLIB.SQLEMPS

OBS	EMPID	HIREDATE	LASTNAME	FIRSTNAM	MIDDLENA
1	119012	01JUL1968	WOLF-PROVENZA	G.	ANDREA
2	120591	05DEC1980	HAMMERSTEIN	S.	RACHAEL
3	123456	04APR1989	VARGAS	CHRIS	J.
4	127845	16JAN1967	MEDER	VLADIMIR	JORAN
5	129540	01AUG1982	CHOULAI	CLARA	JANE
6	135673	15JUL1984	HEMESLY	STEPHANIE	J.
7	212916	15FEB1951	WACHBERGER	MARIE-LOUISE	TERESA
8	216382	15JUN1985	PURINTON	PRUDENCE	VALENTINE
9	234967	19DEC1988	SMITH	GILBERT	IRVINE
10	237642	01NOV1976	BATTERSBY	R.	STEPHEN
11	239185	07MAY1981	DOS REMEDIOS	LEONARD	WESLEY
12	254896	04APR1985	TAYLOR-HUNYADI	ITO	MISHIMA
13	321783	10SEP1967	GONZALES	GUILLERMO	RICARDO
14	328140	10JAN1975	MEDINA-SIDONIA	MARGARET	ROSE
15	346917	02MAR1987	SHIEKELESLAM	SHALA	Y.
16	356134	14JUN1985	DUNNETT	CHRISTINE	MARIE
17	423286	19DEC1988	MIFUNE	YUKIO	TOSHIRO
18	456910	14JUN1978	ARDIS	RICHARD	BINGHAM
19	456921	19AUG1987	KRAUSE	KARL-HEINZ	G.
20	457232	15JUL1985	LOVELL	WILLIAM	SINCLAIR
21	459287	02NOV1964	RODRIGUES	JUAN	M.
22	677890	12DEC1988	NISHIMATSU-LYNCH	CAROL	ANNE
23	346917	02MAR1987	SHIEKELESLAM	SHALA	Y.

Output 21.17 Data in DLib.TempEmps

Data in DLIB.TEMPEMPS Data File

OBS	EMPID	HIREDATE	DEPT	GENDER	LASTNAME	FIRSTNAM	MIDDLENA	FAMILYID
1	765111	04MAY1998	CSR011	M	NISHIMATSU-LYNCH	RICHARD	ITO	677890
2	765112	04MAY1998	CSR010	M	SMITH	ROBERT	MICHAEL	234967
3	219776	15APR1998	ACC024	F	PASTORELLI	ZORA		.
4	245233	10APR1998	ACC013		ALI	SADIQ	H.	.
5	245234	10APR1998	ACC024	F	MEHAILESCU	NADIA	P.	.
6	326721	01MAY1998	SHP002	M	CALHOUN	WILLIS	BEAUREGARD	.

The APPEND procedure also accepts a WHERE= data set option or a SAS WHERE statement to retrieve a subset of the observations. In this example, a subset of the observations from DLib.TempEmps is added to VLib.SqlEmps by using a SAS WHERE statement; the WHERE statement applies only to the DATA= data set.

```
proc append base=vlib.sqlemps
            data=dlib.tempemps force;
   where hiredate >= '01JAN1998'd;
run;

proc print data=vlib.sqlemps;
  title 'Subset of SAS Data Appended
         to a View Descriptor';
run;
```

The following output shows VLib.SqlEmps with three rows from DLib.TempEmps appended to it.

Output 21.18 Subset of Data Appended with the FORCE Option

Subset of SAS Data Appended to a View Descriptor

OBS	EMPID	HIREDATE	LASTNAME	FIRSTNAM	MIDDLENA
1	119012	01JUL1968	WOLF-PROVENZA	G.	ANDREA
2	120591	05DEC1980	HAMMERSTEIN	S.	RACHAEL
3	123456	04APR1989	VARGAS	CHRIS	J.
4	127845	16JAN1967	MEDER	VLADIMIR	JORAN
5	129540	01AUG1982	CHOULAI	CLARA	JANE
6	135673	15JUL1984	HEMESLY	STEPHANIE	J.
7	212916	15FEB1951	WACHBERGER	MARIE-LOUISE	TERESA
8	216382	15JUN1985	PURINTON	PRUDENCE	VALENTINE
9	234967	19DEC1988	SMITH	GILBERT	IRVINE
10	237642	01NOV1976	BATTERSBY	R.	STEPHEN
11	239185	07MAY1981	DOS REMEDIOS	LEONARD	WESLEY
12	254896	04APR1985	TAYLOR-HUNYADI	ITO	MISHIMA
13	321783	10SEP1967	GONZALES	GUILLERMO	RICARDO
14	328140	10JAN1975	MEDINA-SIDONIA	MARGARET	ROSE
15	346917	02MAR1987	SHIEKELESLAM	SHALA	Y.
16	356134	14JUN1985	DUNNETT	CHRISTINE	MARIE
17	423286	19DEC1988	MIFUNE	YUKIO	TOSHIRO
18	456910	14JUN1978	ARDIS	RICHARD	BINGHAM
19	456921	19AUG1987	KRAUSE	KARL-HEINZ	G.
20	457232	15JUL1985	LOVELL	WILLIAM	SINCLAIR
21	459287	02NOV1964	RODRIGUES	JUAN	M.
22	677890	12DEC1988	NISHIMATSU-LYNCH	CAROL	ANNE
23	346917	02MAR1987	SHIEKELESLAM	SHALA	Y.
24	765111	04MAY1994	NISHIMATSU-LYNCH	RICHARD	ITO
25	765112	04MAY1998	SMITH	ROBERT	MICHAEL
26	219776	15APR1998	PASTORELLI	ZORA	
27	245233	10APR1998	ALI	SADIQ	H.
28	245234	10APR1998	MEHAILESCU	NADIA	P.
29	326721	01MAY1998	CALHOUN	WILLIS	BEAUREGARD

See the following output for a copy of the SAS log screen and the messages about the FORCE option.

Output 21.19 SAS Log with Messages about the FORCE Option

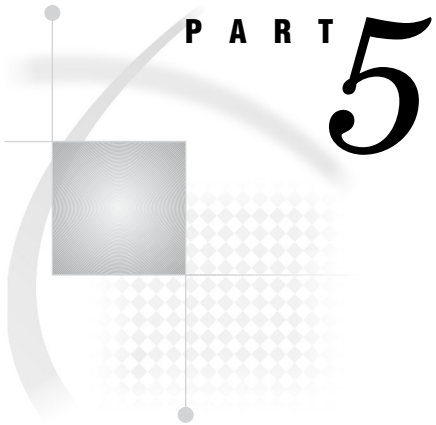
```
10504
10505
10506
10507
10508
10509  proc append base=vlib.sqlemps
10510       data=dlib.tempemps force;
10511     where hiredate <= '30APR1998'd;
10512  run;

NOTE: Appending DLIB.TEMPEMPS to VLIB.SQLEMPS.
WARNING: Variable DEPT was not found on BASE
file.
WARNING: Variable GENDER was not found on BASE
file.
WARNING: Variable FAMILYID was not found on
BASE file.
NOTE: FORCE is specified, so dropping/
truncating will occur.
NOTE: 3 observations added.
NOTE: The data set VLIB.SQLEMPS has .
observations and 5 variables.
```

Because the BASE= data set is a view descriptor in this example, PROC APPEND generates a SQL INSERT statement for the rows to be appended to the DBF file.

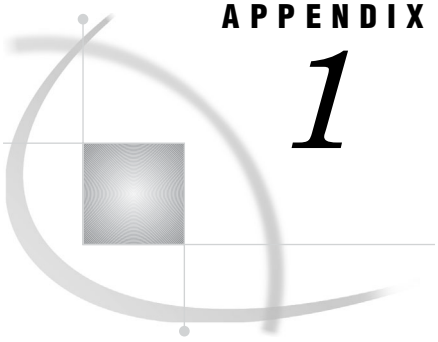
The number of observations in the Employees.dbf file is not displayed in the SAS log because when the view descriptor is opened by the DBF engine, the number of rows in the underlying file is not known.

For more information about the APPEND procedure, see *Base SAS Procedures Guide*.



PART 5

Appendixes



APPENDIX

1

Sample Data

Introduction to Sample Data

This section provides information about the PC files and SAS data files that are used in the examples in this document. In addition, it lists the SAS statements used to create the SAS data files and the data in those files. If you want to run the examples in this document, access your online help system or contact your SAS Software Representative for information about how to access the sample library files.

Sample PC Files

The following sections provide the data in the PC files that are used in the examples in this document.

Customers Data

The data in the Customers file is shown in the following table.

Table A1.1 Customer Data

CUSTOMER	STATE	ZIP CODE	COUNTRY	PHONE	NAME
14324742	CA	95123	USA	408629-0589	SANTA CLARA VALLEY TECHNOLOGY SPECIALISTS
14569877	NC	27514	USA	919/ 489-6792	PRECISION PRODUCTS
14898029	MD	20850	USA	301/ 760-2541	UNIVERSITY BIOMEDICAL MATERIALS
15432147	MI	49001	USA	616/ 582-3906	GREAT LAKES LABORATORY EQUIPMENT MANUFACTURERS
18543489	TX	78701	USA	512/ 478-0788	LONE STAR STATE RESEARCH SUPPLIERS
19783482	VA	22090	USA	703/ 714-2900	TWENTY-FIRST CENTURY MATERIALS
19876078	CA	93274	USA	209/ 686-3953	SAN JOAQUIN SCIENTIFIC AND INDUSTRIAL SUPPLY, INC.
26422096		75014	France	4268-54-72	SOCIETE DE RECHERCHES POUR LA CHIRURGIE ORTHOPEDIQUE
26984578		5110	Austria	43-57-04	INSTITUT FUER TEXTIL-FORSCHUNG
27654351		5010	Belgium	02/215-37-32	INSTITUT DE RECHERCHE SCIENTIFIQUE MEDICALE
28710427		3607	Netherlands	(021)570517	ANTONIE VAN LEEUWENHOEK VERENIGING VOOR MICROBIOLOGIE
29834248		—-	Britain	(0552)715311	BRITISH MEDICAL RESEARCH AND SURGICAL SUPPLY
31548901		—-	Canada	406/ 422-3413	NATIONAL COUNCIL FOR MATERIALS RESEARCH
38763919		1405	Argentina	244-6324	INSTITUTO DE BIOLOGIA Y MEDICINA NUCLEAR
39045213		01051	Brazil	012/ 302-1021	LABORATORIO DE PESQUISAS VETERINARIAS DESIDERIO FINAMOR
43290587		—-	Japan	(02)933-3212	HASSEI SAIBO GAKKAI
43459747		3181	Australia	03/734-5111	RESEARCH OUTFITTERS
46543295		—-	Japan	(03)022-2332	WESTERN TECHNOLOGICAL SUPPLY

46783280	2374	Singapore	3762855	NGEE TECHNOLOGICAL INSTITUTE
48345514	—-	United Arab Emirates	213445	GULF SCIENTIFIC SUPPLIES

CUSTOMER	CONTACT	ADDRESS	CITY	FIRSTORDER
14324742	A. BAUM	5089 CALERO AVENUE	SAN JOSE	05FEB1970
14569877	CHARLES BARON	198 FAYETTEVILLE ROAD	MEMPHIS	15AUG1988
14898029	S. TURNER	1598 PICCARD DRIVE	ROCKVILLE	12NOV1981
15432147	D.W. KADARAUCH	103 HARRIET STREET	KALAMAZOO	28APR1991
18543489	A. SILVERIA	5609 RIO GRANDE	AUSTIN	10SEP1984
19783482	M.R. HEFFERNAN	4613 MICHAEL FARADAY DRIVE	RESTON	18JUL1973
19876078	J.A. WHITTEN	1095 HIGHWAY 99 SOUTH	TULARE	11MAY1984
26422096	Y. CHAVANON	40 RUE PERIGNON	LA ROCHELLE	14JUN1988
26984578	GUNTHER SPIELMANN	MECHITARISTENGASSE 5	VIENNA	25MAY1992
27654351	I. CLEMENS	103 RUE D'EGMONT	BRUSSELS	14OCT1991
28710427	M.C. BORGSTEEDE	BIRMOERSTRAAT 34	THE HAGUE	10OCT1990
29834248	A.D.M. BRYCESON	44 PRINCESS GATE, HYDE PARK	LONDON, SW7 1PU	29JAN1991
31548901	W.E. MACDONALD	5063 RICHMOND MALL	VANCOUVER, V5T 1L2	19MAR1989
38763919	JORGE RUNNAZZO	SALGUERO 2345	BUENOS AIRES	10DEC1989
39045213	ELISABETE REGIS GUILLAUMON	RUA DONA ANTONIA DE QUEIROS 381	SAO PAULO	18AUG1987
43290587	Y. FUKUDA	3-2-7 ETCHUJMA, KOTO-KU	TOKYO 101	08FEB1979
43459747	R.G. HUGHES	191 LOWER PLENTY ROAD	PRAHRAN, VICTORIA	28JUL1977
46543295		4-3-8 ETCHUJMA, KOTO-KU	TOKYO 102	19APR1989

46783280	LING TAO SOON	356 CLEMENTI ROAD	SINGAPORE	27SEP1984
48345514	J.Q. RIFAII	POB 8032	RAS AL KHAIMAH	10SEP1991

Employees Data

The data in the Employees file is shown in the following table.

Table A1.2 Employee Data

EMPID	HIREDATE	SALARY	DEPT	JOBCODE	SEX
119012	01JUL1973	42340.58	CSR010	602	F
120591	05DEC1985	31000.55	SHP002	602	F
127845	16JAN1972	75320.34	ACC024	204	M
129540	01AUG1987	56123.34	SHP002	204	F
135673	15JUL1989	46322.58	ACC013	602	F
212916	15FEB1958	52345.58	CSR010	602	F
216382	15JUN1990	34004.65	SHP013	602	F
234967	19DEC1993	17000.00	CSR004	602	M
237642	01NOV1981	43200.34	SHP013	602	M
239185	07MAY1986	57920.66	ACC024	602	M
254896	04APR1990	35000.74	CSR011	204	M
321783	10SEP1972	48931.58	CSR011	602	M
328140	10JAN1980	75000.34	ACC043	1204	F
346917	02MAR1992	46000.33	SHP013	204	F
356134	14JUN1990	62450.75	ACC013	204	F
423286	19DEC1993	32870.66	ACC024	602	M
456910	14JUN1983	45000.58	CSR010	602	M
456921	19AUG1992	33210.04	SHP002	602	M
457232	15JUL1990	55000.66	ACC013	602	M
459287	02NOV1969	50000.00	SHP024	204	M
677890	12DEC1993	37610.00	CSR010	204	F
123456	04APR1994	——	ACC043	1204	—-

Employee Data

EMPID	BIRTHDATE	LASTNAME	FIRSTNAME	MIDDLENAME	PHONE
119012	05JAN1951	WOLF-PROVENZA	G.	ANDREA	3467
120591	12FEB1951	HAMMERSTEIN	S.	RACHAEL	3287

EMPID	BIRTHDATE	LASTNAME	FIRSTNAME	MIDDLENAME	PHONE
127845	25DEC1948	MEDER	VLADIMIR	JORAN	6231
129540	31JUL1965	CHOULAI	CLARA	JANE	3921
135673	21MAR1966	HEMESLY	STEPHANIE	J.	6329
212916	29MAY1935	WACHBERGER	MARIE-LOUISE	TERESA	8562
216382	24JUL1968	PURINTON	PRUDENCE	VALENTINE	3852
234967	21DEC1972	SMITH	GILBERT	IRVINE	7274
237642	13MAR1959	BATTERSBY	R.	STEPHEN	8342
239185	28AUG1964	DOS REMEDIOS	LEONARD	WESLEY	4892
254896	06APR1954	TAYLOR-HUNYADI	ITO	MISHIMA	1231
321783	03JUN1940	GONZALES	GUILLERMO	RICARDO	3642
328140	02JUN1956	MEDINA-SIDONIA	MARGARET	ROSE	5901
346917	15MAR1955	SHIEKELESLAM	SHALA	Y.	8745
356134	25OCT1965	DUNNETT	CHRISTINE	MARIE	4213
423286	31OCT1969	MIFUNE	YUKIO	TOSHIRO	3278
456910	24SEP1958	ARDIS	RICHARD	BINGHAM	4351
456921	12MAY1967	KRAUSE	KARL-HEINZ	G.	7452
457232	15OCT1968	LOVELL	WILLIAM	SINCLAIR	6321
459287	05JAN1939	RODRIGUES	JUAN	M.	5879
677890	24APR1970	NISHIMATSU-LYNCH	CAROL	ANNE	6245
123456	———	VARGAS	CHRIS	J.	—-

Invoice Data

The data in the Invoice file is shown in the following table.

Table A1.3 Invoice Data

INVOICENUM	BILLEDTO	AMTBILLED	COUNTRY	AMOUNTINUS	BILLEDBY	BILLEDON	PAIDON
11270	39045213	1340738760.90	Brazil	2256870.00	239185	05OCT1998	18OCT1998
11271	18543489	11063836.00	USA	11063836.00	457232	05OCT1998	11OCT1998
11273	19783482	252148.50	USA	252148.50	239185	06OCT1998	11NOV1998
11276	14324742	1934460.00	USA	1934460.00	135673	06OCT1998	20OCT1998
11278	14898029	1400825.00	USA	1400825.00	239185	06OCT1998	19OCT1998
11280	39045213	1340738760.90	Brazil	2256870.00	423286	07OCT1998	20OCT1998
11282	19783482	252148.50	USA	252148.50	457232	07OCT1998	25OCT1998

INVOICENUM	BILLEDTO	AMTBILLED	COUNTRY	AMOUNTINUS	BILLEDBY	BILLEDON	PAIDON
11285	38763919	34891210.20	Argentina	2256870.00	239185	10OCT1998	30NOV1998
11286	43459747	12679156.00	Australia	11063836.00	423286	10OCT1998	———
11287	15432147	252148.50	USA	252148.50	457232	11OCT1998	04NOV1998
12051	39045213	1340738760.90	Brazil	2256870.00	457232	02NOV1998	——-
12102	18543489	11063836.00	USA	11063836.00	239185	17NOV1998	——-
12263	19783482	252148.50	USA	252148.50	423286	05DEC1998	——-
12468	14898029	1400825.00	USA	1400825.00	135673	24DEC1998	02JAN1999
12471	39045213	1340738760.90	Brazil	2256870.00	457232	27DEC1998	——-
12476	38763919	34891210.20	Argentina	2256870.00	135673	24DEC1998	——-
12478	15432147	252148.50	USA	252148.50	423286	24DEC1998	02JAN1999

Orders Data

The data in the Orders file is shown in the following table.
The data in the SpecInstr column is shown truncated in some cases. The full text is

```
Customer agrees to accept any liabilities
that may arise from the use of this product.
If the customer is sued, the customer agrees
not to countersue us.
```

Table A1.4 Orders Data

ORDERNUM	STOCKNUM	LENGTH	FABCHARGES	SHIPTO	DATEORDERED
11269	9870	690	- - –19876078	03OCT1998	- - –
11270	1279	1750	2256870	39045213	03OCT1998
11271	8934	110	11063836.00	18543489	03OCT1998
11272	3478	1000	—-	29834248	03OCT1998
11273	2567	450	252148.50	19783482	04OCT1998
11274	4789	1000	—-	15432147	04OCT1998
11275	3478	1000	—-	29834248	04OCT1998
11276	1279	1500	1934460.00	14324742	04OCT1998
11277	8934	100	10058033.00	31548901	05OCT1998
11278	2567	2500	1400825.00	14898029	05OCT1998
11279	9870	650	—-	48345514	05OCT1998
11280	1279	1750	2256870.00	39045213	06OCT1998
11281	8934	110	11063836.00	18543489	06OCT1998
11282	2567	450	252148.50	19783482	06OCT1998
11283	9870	690	—-	18543489	07OCT1998
11284	3478	1000	—-	24589689	07OCT1998
11285	1279	1750	256870.00	38763919	07OCT1998

ORDERNUM	STOCKNUM	LENGTH	FABCHARGES	SHIPTO	DATEORDERED
11286	8934	110	11063836.00	43459747	07OCT1998
11287	2567	450	252148.50	15432147	07OCT1998
11290	9870	690	—-	14324742	10OCT1998
11969	9870	690	—-	19876078	25OCT1998
12051	1279	1750	2256870.00	39045213	
12102	8934	110	11063836.00	18543489	15NOV1998
12160	3478	1000	—-	29834248	19NOV1998
12263	2567	450	252148.50	19783482	01DEC1998
12464	4789	—-	15432147	23DEC1998	212916
	1000				
12465	3478	1000	—-	29834248	23DEC1998
12466	1279	1500	1934460.00	14324742	23DEC1998
12467	8934	100	10058033.00	31548901	23DEC1998
12468	2500	1400825.00	14898029	23DEC1998	03JAN1999
2567					
12470	9870	650	—-	48345514	23DEC1998
12471	1279	1750	2256870.00	39045213	23DEC1998
12472	8934	110	11063836.00	18543489	23DEC1998
12473	2567	450	252148.50	19783482	23DEC1998
12474	9870	690	—-	18543489	23DEC1998
12475	3478	1000	—-	24589689	23DEC1998
12476	1279	1750	2256870.00	38763919	23DEC1998
12477	8934	110	11063836.00	43459747	23DEC1998
12478	2567	450	252148.50	15432147	23DEC1998
12479	9870	690	—-	14324742	23DEC1998

ORDERNUM	SHIPPED	TAKENBY	PROCESSDBY	SPECINSTR
11269	212916			
11270	19Oct1998	321783	237642	Customer agrees to a
11271	13OCT1998	456910	456921	—-
11272	—-	234967	—-	—-
11273	14NOV1998	119012	216382	—-
11274	—-	212916	—-	—-
11275	—-	234967	—-	—-
11276	21OCT1998	321783	120591	Customer agrees to a
11277	—-	456910	—-	—-
11278	20OCT1998	119012	456921	—-
11279	—-	212916	—-	—-

ORDERNUM	SHIPPED	TAKENBY	PROCESSDBY	SPECINSTR
11280	21OCT1998	321783	237642	Customer agrees to a
11281	27OCT1998	456910	216382	—-
11282	26OCT1998	119012	456921	—-
11283	—-	212916	—-	—-
11284	—-	234967	—-	—-
11285	02DEC1998	321783	120591	Customer agrees to a
11286	03NOV1998	456910	237642	—-
11287	07NOV1998	119012	216382	—-
11290	—-	212916	—-	Customer agrees to certain limitations.
11969	—-	212916	—-	—-
12051	31OCT1998	—-	321783	—-
12102	—-	456910	—-	—-
12160	—-	234967	—-	Customer agrees to pay in full.
12263	—-	119012	—-	—-
12464	—-	—-		
12465	—-	234967	—-	—-
12466	—-	321783	—-	Customer agrees to a
12467	—-	456910	—-	—-
12468	119012	120591	—-	
2567				
12470	—-	212916	—-	—-
12471	—-	321783	—-	Customer agrees to a
12472	03JAN1999	456910	237642	—-
12473	—-	119012	—-	—-
12474	—-	212916	—-	—-
12475	—-	234967	—-	—-
12476	03JAN1999	321783	456921	Customer agrees to a
12477	—-	456910	—-	—-
12478	03JAN1999	119012	216382	—-
12479	—-	212916	—-	—-

SpecProd Data

The data in the SpecProd file is shown in the following table.

Table A1.5 Special Products Data

PRODUCED	WEIGHT		FIBER NAME	FIBER SIZE	
2356	+8.967499730	E– 01	nylon	+6.780000000	E-13
4789	+7.967500090	E– 01	dacron	+5.780000000	E-13
9870	+6.967499850	E– 01	polyester	+4.760000000	E-13
3478	+9.949499960	E– 01	olefin	+9.880000000	E-13
9678	+6.942499880	E– 01	cotton	+3.420000000	E-13
3456	+1.675000040	E– 02	silk	+2.678000000	E-11
8934	+1.429999950	E– 03	gold	+2.380000000	E-12
2567	+1.258500220	E– 01	fiberglass	+5.188000000	E-11
1279	+1.278899910	E– 01	asbestos	+6.347600000	E-10

PRODUCED	COST	PER UNIT	WIDTH
2356	—-	—-	—-
4789	—-	—-	—-
9870	—-	—-	—-
3478	—-	sq yd	—-
9678	—-	—-	—-
3456	—-	—-	—-
8934	100580.33	cm	+2.255999760
2567	560.33	m	+1.205000000
1279	1289.64	m	+2.227550050

SAS Data Files

This section describes the SAS data files used in the examples in this PC files documentation. It gives the SAS statements that created each data file and shows the output from the PRINT procedure.

DLib.Birthday SAS Data File

The SAS data file DLib.Birthday is created with the following SAS statements:

```
libname dlib 'Your-SAS-data-library';
data dlib.birthday;
   input empid birthdat date7.
         lastname $18.
         firstnam $15.
         phone $4.;
datalines;
459287 05JAN39 RODRIGUES       JUAN     5879
```

```
127845 25DEC48 MEDER           VLADIMIR 6231
254896 06APR54 TAYLOR-HUNYADI  ITO      0231
;
run;
```

This PRINT procedure lists the data shown in the following output:

```
proc print data=dlib.birthday;
   format birthdat date9.;
   title 'DLIB.BIRTHDAY Data File';
run;
```

Output A1.1 Data in SAS Data File DLib.Birthday

DLIB.BIRTHDAY Data File

OBS	EMPID	BIRTHDAT	LASTNAME	FIRSTNAM	PHONE
1	459287	05JAN1939	RODRIGUES	JUAN	5879
2	127815	25DEC1948	MEDER	VLADIMIR	6231
3	254196	06APR1954	TAYLOR-HUNYADI	ITO	0231

DLib.OutOfStk SAS Data File

The SAS data file DLib.OutOfStk is created with the following SAS statements:

```
libname dlib 'Your-SAS-data-library';
data dlib.outofstk;
   input fibernam $8. fibernum;
   datalines;
olefin   3478
gold     8934
dacron   4789
;
```

This PRINT procedure lists the data shown in the following output:

```
proc print data=dlib.outofstk;
   title 'SAS Data File DLIB.OUTOFSTK';
run;
```

Output A1.2 Data in SAS Data File DLib.OutOfStk

SAS Data File DLIB.OUTOFSTK

OBS	FIBERNAM	FIBERNUM
1	olefin	3478
2	gold	8934
3	dacron	4789

DLib.TempEmps SAS Data File

The SAS data file DLib.TempEmps is created with the following PROC SQL statements:

```
libname dlib 'Your-SAS-data-library';
proc sql;
 create table dlib.tempemps
   (label='Student interns',
    empid num, hiredate date format=date9.,
    dept char(6), gender char(1),
    lastname char(18), firstnam char(15),
    middlena char(15), familyid num);

 insert into dlib.tempemps
   values(765111,'04MAY1998'd,'CSR011','M',
         'NISHIMATSU-LYNCH','RICHARD',
         'ITO',677890)
   values(765112,'04MAY1998'd,'CSR010','M',
         'SMITH','ROBERT','MICHAEL',234967)
   values(219776,'15APR98'd,'ACC024','F',
         'PASTORELLI','ZORA',null,.)
   values(245233,'10APR1998'd,'ACC013','
         ','ALI','SADIQ','H.',.)
   values(245234,'10APR1998'd,'ACC024','F',
         'MEHAILESCU','NADIA','P.',.)
   values(326721,'01MAY1998'd,'SHP002','M',
         'CALHOUN','WILLIS','BEAUREGARD',.);

quit;
```

This PRINT procedure lists the data shown in the following output:

```
options ls=120;
proc print data=dlib.tempemps;
   title 'DLIB.TEMPEMPS Data File';
run;
```

Output A1.3 Data in DLib.TempEmps

DLIB.TEMPEMPS Data File

OBS	EMPID	HIREDATE	DEPT	GENDER	LASTNAME	FIRSTNAM	MIDDLENA	FAMILYID
1	765111	04MAY1998	CSR011	M	NISHIMATSU-LYNCH	RICHARD	ITO	677890
2	765112	04MAY1998	CSR010	M	SMITH	ROBERT	MICHAEL	234967
3	219776	15APR1998	ACC024	F	PASTORELLI	ZORA		.
4	245233	10APR1998	ACC013		ALI	SADIQ	H.	.
5	245234	10APR1998	ACC024	F	MEHAILESCU	NADIA	P.	.
6	326721	01MAY1998	SHP002	M	CALHOUN	WILLIS	BEAUREGARD	.

DLib.RateOfex SAS Data File

The SAS data file DLib.RateOfex is created with the following SAS statements:

```
libname dlib 'Your-SAS-data-library';
data dlib.rateofex;
   input updated date9.
         currency & $15.
         fgnindol : 8.
         dolinfgn : 11.
         country & $20.;
   format updated date9.
          currency $15.
          fgnindol 8.6
          dolinfgn 11.6
          country $20.;
datalines;
28JUL1998 peso    1.01 0.99 Argentina
28JUL1998 dollar    0.7457 1.3410 Australia
...more data lines...
;
```

This PRINT procedure lists the data shown in the following output:

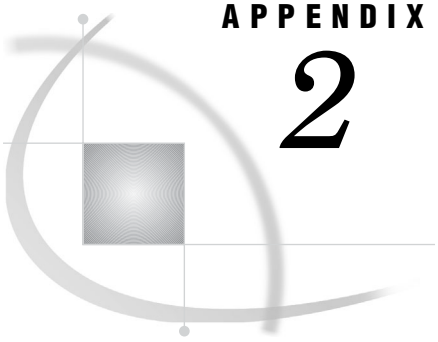
```
proc print data=dlib.rateofex;
   title 'Data in SAS Data File DLIB.RATEOFEX';
run;
```

Output A1.4 Data in SAS Data File DLib.RateOfex

Data in SAS Data File DLIB.RATEOFEX

OBS	UPDATED	CURRENCY	FGNINDOL	DOLINFGN	COUNTRY
1	28JUL1998	peso	1.010000	0.990000	Argentina
2	28JUL1998	dollar	0.745700	1.341000	Australia
3	28JUL1998	schilling	0.095940	10.420000	Austria
4	28JUL1998	dinar	2.652200	0.377100	Bahrain
5	28JUL1998	franc	0.032780	30.510000	Belgium
6	28JUL1998	cruzeiro	0.000260	3872.000000	Brazil
7	28JUL1998	pound	1.919500	0.521000	Britain
8	28JUL1998	dollar	0.841400	1.188500	Canada
9	28JUL1998	peso	0.002835	352.750000	Chile
10	28JUL1998	renminbi	0.182815	5.470000	China
11	28JUL1998	peso	0.001722	580.640000	Columbia
12	28JUL1998	krone	0.175400	5.700500	Denmark
13	28JUL1998	sucre	0.000693	1443.000000	Ecuador
14	28JUL1998	markka	0.246490	4.057000	Finland
15	28JUL1998	franc	0.199980	5.000500	France
16	28JUL1998	mark	0.675400	1.480500	Germany
17	28JUL1998	drachma	0.005491	182.100000	Greece
18	28JUL1998	dollar	0.129190	7.740500	Hong Kong
19	28JUL1998	forint	0.013139	76.110000	Hungary
20	28JUL1998	rupee	0.035650	28.050000	India
21	28JUL1998	rupiah	0.000493	2029.510000	Indonesia
22	28JUL1998	punt	1.802600	0.554800	Ireland
23	28JUL1998	shekel	0.406500	2.460000	Israel
24	28JUL1998	lira	0.000892	1120.500000	Italy
25	28JUL1998	yen	0.007843	127.500000	Japan
26	28JUL1998	dinar	1.527700	0.654600	Jordan
27	28JUL1998	dinar	3.420600	0.292400	Kuwait
28	28JUL1998	pound	0.000503	1990.000000	Lebanon
29	28JUL1998	ringgit	0.400000	2.500300	Malaysia
30	28JUL1998	lira	3.344500	0.299000	Malta
31	28JUL1998	peso	0.000321	3114.510000	Mexico
32	28JUL1998	guilder	0.598900	1.669800	Netherlands
33	28JUL1998	dollar	0.546100	1.831200	New Zealand
34	28JUL1998	krone	0.171800	5.819500	Norway
35	28JUL1998	rupee	0.040000	25.000000	Pakistan
36	28JUL1998	new sol	0.839000	1.190000	Peru
37	28JUL1998	peso	0.040900	24.450000	Philippines
38	28JUL1998	zloty	0.000077	12959.01000	Poland
39	28JUL1998	escudo	0.007949	125.800000	Portugal
40	28JUL1998	riyal	0.267380	3.740000	Saudi Arabia
41	28JUL1998	dollar	0.619000	1.615500	Singapore
42	28JUL1998	rand	0.362300	2.759800	South Africa
43	28JUL1998	won	0.001270	787.400000	South Korea
44	28JUL1998	peseta	0.010612	94.230000	Spain
45	28JUL1998	krona	0.186000	5.376000	Sweden
46	28JUL1998	franc	0.763400	1.310000	Switzerland
47	28JUL1998	dollar	0.039714	25.180000	Taiwan
48	28JUL1998	baht	0.039450	25.350000	Thailand
49	28JUL1998	lira	0.000144	6938.000000	Turkey
50	28JUL1998	dirham	0.272300	3.672500	United Arab
51	28JUL1998	new peso	0.000321	3120.010000	Uruguay
52	28JUL1998	bolivar	0.015130	66.090000	Venezuela

The DLib.RateOfex data is used primarily in the Version 6 compatibility examples.



APPENDIX

2

Recommended Reading

Recommended Reading

Here is the recommended reading list for this title:

- *SAS/ACCESS for Relational Databases: Reference*
- *SAS Language Reference: Concepts*
- *SAS Language Reference: Dictionary*
- *Base SAS Procedures Guide*
- SAS Companion that is specific to your operating environment

For a complete list of SAS publications, see the current *SAS Publishing Catalog*. To order the most current publications or to receive a free copy of the catalog, contact a SAS representative at

SAS Publishing Sales
SAS Campus Drive
Cary, NC 27513
Telephone: (800) 727-3228*
Fax: (919) 677-8166
E-mail: **sasbook@sas.com**
Web address: **support.sas.com/pubs**
* For other SAS Institute business, call (919) 677-8000.

Customers outside the United States should contact their local SAS office.

Glossary

This glossary defines SAS software terms that are used in this document as well as terms that relate specifically to SAS/ACCESS software.

access descriptor
a SAS/ACCESS file that describes data that is managed by a data management system. After creating an access descriptor, you can use it as the basis for creating one or more view descriptors.

browsing data
the process of viewing the contents of a file. Depending on how the file is accessed, you can view SAS data either one observation (row) at a time or as a group in a tabular format. You cannot update data that you are browsing.

column
in relational databases, a vertical component of a table. Each column has a unique name, contains data of a specific type, and has certain attributes. A column is analogous to a variable in SAS terminology.

column function
an operation that is performed for each value in the column that is named as an argument of the function. For example, AVG(SALARY) is a column function.

commit
the process that ends a transaction and makes permanent any changes to the database that the user made during the transaction. When the commit process occurs, locks on the database are released so that other applications can access the changed data. The SQL COMMIT statement initiates the commit processs.

data type
an attribute of every column in a table or database. The data type tells the operating system how much physical storage to set aside for the column and specifies what type of data the column will contain. It is similar to the type attribute of SAS variables.

data value
in SAS software, a unit of character or numeric information in a SAS data set. A data value represents one variable in an observation.

database
an organized collection of related data. A database usually contains named files, named objects, or other named entities such as tables, views, and indexes.

database field
a vertical component of a dBASE .DBF file that contains data of a specific type with certain attributes. A database field is analogous to a variable in SAS terminology.

database file
a two-dimensional system of representing data in records and fields.

database management system (DBMS)
a software application that enables you to create and manipulate data in the form of databases. See also relational database management system.

editing data
the process of viewing the contents of a file with the intent and the ability to change its data. Depending on how the file is accessed, you can view the details either one observation at a time or in a tabular format.

engine
a component of SAS software that reads from or writes to a file. Each engine enables SAS to access files that are in a particular format. There are several types of engines.

file
a collection of related records that are treated as a unit. SAS files are processed and controlled through SAS software and are stored in SAS data libraries.

format
an instruction that SAS uses to display or write each value of a variable (or column). Some formats are supplied by SAS software. You can create other formats by using the FORMAT procedure in Base SAS software. In SAS/ACCESS software, the default formats vary according to the interface product.

index
in SAS software, a component of a SAS data set that enables SAS to access observations in the SAS data set quickly and efficiently. The purpose of SAS indexes is to optimize WHERE-clause processing and to facilitate BY-group processing.

informat
a pattern or set of instructions that SAS uses to determine how data values in an input file should be interpreted. SAS provides a set of standard informats and also enables you to define your own informats.

interface view engine
a SAS engine that is used by SAS/ACCESS software to retrieve data from files that have been formatted by another vendor's software. Each SAS/ACCESS interface has its own interface view engine, which reads the interface product data and returns the data in a form that SAS can understand (that is, in a SAS data set). SAS automatically uses an interface view engine; the engine name is stored in SAS/ACCESS descriptor files so that you do not need to specify the engine name in a LIBNAME statement.

libref
a name that is temporarily associated with a SAS data library. The complete name of a SAS file consists of two words, separated by a period. The libref, which is the first word, indicates the library. The second word is the name of the specific SAS file. For example, in VLIB.NEWBDAY, the libref VLIB tells SAS which library contains the file NEWBDAY. You assign a libref with a LIBNAME statement or with an operating system command.

member
a name that represents a particular data item within a dimension. For example, September 1996 might be a member of the Time dimension. A member can be either

unique or non-unique. For example, 1997 and 1998 represent unique members in the Year level of a Time dimension. January represents non-unique members in the Month level, because there can be more than one January in the Time dimension if the Time dimension contains data for more than one year.

member name
a name that is assigned to a SAS file in a SAS library.

member type
a SAS name that identifies the type of information that is stored in a SAS file. Member types include ACCESS, DATA, CATALOG, ITEMSTOR, MDDB, PROGRAM, and VIEW.

missing value
in SAS, a term that describes the contents of a variable that contains no data for a particular row or observation. By default, SAS prints or displays a missing numeric value as a single period, and it prints or displays a missing character value as a blank space.

observation
a row in a SAS data set. All of the data values in an observation are associated with a single entity such as a customer or a state. Each observation contains one data value for each variable. In a database product table, an observation is analogous to a row. Unlike rows in a database product table or file, observations in a SAS data file have an inherent order.

PROC SQL view
a SAS data set (of type VIEW) that is created by the SQL procedure. A PROC SQL view contains no data. Instead, it stores information that enables it to read data values from other files, which can include SAS data files, SAS/ACCESS views, DATA step views, or other PROC SQL views. A PROC SQL view's output can be either a subset or a superset of one or more files.

record
a logical unit of information that consists of fields of related data. A collection of records are stored in a file. A record is analogous to a SAS observation.

relational database management system
a database management system that organizes and accesses data according to relationships between data items. The main characteristic of a relational database management system is the two-dimensional table. Examples of relational database management systems are DB2, INGRES, ORACLE, and SQL/DS.

rollback
in most databases, the process that restores a database to its state when changes were last committed, voiding any changes. The SQL ROLLBACK statement initiates the rollback processs.

row
in relational database management systems, the horizontal component of a table. A row is analogous to a SAS observation.

SAS data file
a type of SAS data set that contains data values as well as descriptor information that is associated with the data. The descriptor information includes information such as the data types and lengths of the variables, as well as the name of the engine that was used to create the data. A PROC SQL table is a SAS data file. SAS data files are of member type DATA.

SAS data library
a collection of one or more SAS files that are recognized by SAS and which are referenced and stored as a unit. Each file is a member of the library.

SAS data set
a file whose contents are in one of the native SAS file formats. There are two types of SAS data sets: SAS data files and SAS data views. SAS data files contain data values in addition to descriptor information that is associated with the data. SAS data views contain only the descriptor information plus other information that is required for retrieving data values from other SAS data sets or from files whose contents are in other software vendors' file formats.

SAS data view
a type of SAS data set that retrieves data values from other files. A SAS data view contains only descriptor information such as the data types and lengths of the variables (columns), plus other information that is required for retrieving data values from other SAS data sets or from files that are stored in other software vendors' file formats. SAS data views are of member type VIEW.

Structured Query Language (SQL)
a standardized, high-level query language that is used in relational database management systems to create and manipulate database management system objects. SAS implements SQL through the SQL procedure.

variable
a column in a SAS data set or in a SAS data view. The data vlues for each variable describe a single characteristic for all observations. In the ACCESS procedure, variables are created from the PC files' columns or fields.

view
a definition of a virtual data set. The definition is named and stored for later use. A view contains no data but describes or defines data that are stored elsewhere. See also PROC SQL view, SAS data view, and view descriptor.

view descriptor
a SAS/ACCESS file that defines part or all of the DBMS data that is described by an access descriptor.

windowing procedure
a SAS procedure that you can use by entering information in one or more windows or dialog boxes. For example, the FSVIEW procedure is a windowing procedure. Some procedures, such as ACCESS and DBLOAD, can be used either as windowing procedures or in batch mode.

Index

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

Your Turn

If you have comments or suggestions about *SAS/ACCESS® 9.1 Interface to PC Files: Reference*, please send them to us on a photocopy of this page, or send us electronic mail.

For comments about this book, please return the photocopy to

SAS Publishing
SAS Campus Drive
Cary, NC 27513
E-mail: **yourturn@sas.com**

For suggestions about the software, please return the photocopy to

SAS Institute Inc.
Technical Support Division
SAS Campus Drive
Cary, NC 27513
E-mail: **suggest@sas.com**