

# **SAS/ACCESS<sup>®</sup> 9.3 Interface to PC Files Reference**



The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2011. *SAS/ACCESS® 9.3 Interface to Files: Reference*. Cary, NC: SAS Institute Inc.

**SAS/ACCESS® 9.3 Interface to PC Files: Reference**

Copyright © 2011, SAS Institute Inc., Cary, NC, USA

All rights reserved. Produced in the United States of America.

**For a hardcopy book:** No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

**For a Web download or e-book:** Your use of this publication shall be governed by the terms established by the vendor at the time you acquire this publication.

The scanning, uploading, and distribution of this book via the Internet or any other means without the permission of the publisher is illegal and punishable by law. Please purchase only authorized electronic editions and do not participate in or encourage electronic piracy of copyrighted materials. Your support of others' rights is appreciated.

**U.S. Government Restricted Rights Notice:** Use, duplication, or disclosure of this software and related documentation by the U.S. government is subject to the Agreement with SAS Institute and the restrictions set forth in FAR 52.227–19 Commercial Computer Software-Restricted Rights (June 1987).

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

1st electronic book, July 2011

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/publishing](http://support.sas.com/publishing) or call 1-800-727-3228.

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

---

# Contents

<i>Using This Document</i> . . . . .	<i>vii</i>
<i>What's New in SAS/ACCESS 9.3 Interface to PC Files</i> . . . . .	<i>ix</i>
<i>Recommended Reading</i> . . . . .	<i>xi</i>

## PART 1 Introduction to SAS/ACCESS 9.3 Interface to PC Files 1

<b>Chapter 1 • Working with SAS/ACCESS 9.3 Interface to PC Files</b> . . . . .	<b>3</b>
Methods for Accessing PC Files Data . . . . .	3
Using This Document . . . . .	4
Sample Data in This Document . . . . .	4

## PART 2 Import and Export Wizards and Procedures 5

<b>Chapter 2 • Supported Data Sources and Environments</b> . . . . .	<b>7</b>
Reading and Writing Data between SAS Data Sets and PC Files . . . . .	7
Supported Data Sources and Environments . . . . .	7
<b>Chapter 3 • Using the SAS Import and Export Wizards</b> . . . . .	<b>11</b>
Importing and Exporting Data with the Wizards . . . . .	11
Using the External File Interface (EFI) . . . . .	11
<b>Chapter 4 • The IMPORT Procedure</b> . . . . .	<b>13</b>
Overview: PROC IMPORT . . . . .	13
Syntax: The IMPORT Procedure . . . . .	14
<b>Chapter 5 • The EXPORT Procedure</b> . . . . .	<b>21</b>
Overview: PROC EXPORT . . . . .	21
Syntax: The EXPORT Procedure . . . . .	22
<b>Chapter 6 • File Format-Specific Reference for the IMPORT and EXPORT Procedures</b> . . . . .	<b>27</b>
Delimited Files . . . . .	29
Microsoft Excel Workbook Files . . . . .	33
Microsoft Access Database Files . . . . .	52
Lotus 1-2-3 WKn Files . . . . .	62
dBase DBF Files . . . . .	66
dBase DBF MEMO Files . . . . .	70
JMP Files . . . . .	71
Paradox DB File Formats . . . . .	74
SPSS SAV Files . . . . .	75
Stata DTA Files . . . . .	78

## PART 3 LIBNAME Access and Excel Engines on Microsoft Windows 83

<b>Chapter 7 • Interaction and Functionality</b> . . . . .	<b>85</b>
Overview of LIBNAME Statement for Access and Excel on Microsoft Windows . . . . .	85
Sorting PC Files Data . . . . .	85
Using SAS Functions with PC Files Data . . . . .	86
Assigning a Libref Interactively . . . . .	86
<b>Chapter 8 • The LIBNAME Engines</b> . . . . .	<b>89</b>
Overview: LIBNAME Engines . . . . .	89
Software Requirements . . . . .	90
Macro Variables . . . . .	90
System Options . . . . .	91
Dictionary . . . . .	91
<b>Chapter 9 • The LIBNAME Statement for Access and Excel on Microsoft Windows</b> . . . . .	<b>97</b>
Dictionary . . . . .	97
<b>Chapter 10 • Data Set Options</b> . . . . .	<b>103</b>
Overview of Data Set Options . . . . .	103
Dictionary . . . . .	104
<b>Chapter 11 • LIBNAME Options</b> . . . . .	<b>121</b>
Dictionary . . . . .	121
<b>Chapter 12 • Pass-Through Facility for Access and Excel on Microsoft Windows</b> . . . . .	<b>129</b>
Pass-Through Facility on Microsoft Windows . . . . .	129
Dictionary . . . . .	130
<b>Chapter 13 • File-Specific Reference for Access and Excel on Microsoft Windows</b> . . . . .	<b>147</b>
Microsoft Excel Workbook Files . . . . .	147
Microsoft Access Files . . . . .	153

## PART 4 LIBNAME PCFILES Engine and PC Files Server on Microsoft Windows 159

<b>Chapter 14 • PC Files Server Administration</b> . . . . .	<b>161</b>
Overview . . . . .	162
Windows Service . . . . .	164
Desktop Application . . . . .	164
PC Files Server Configuration . . . . .	166
Authentication . . . . .	167
PC Files Server Autostart . . . . .	169
Local Security Policy Configuration . . . . .	169
Constraints . . . . .	170
Shared Information . . . . .	171
<b>Chapter 15 • LIBNAME Statement for PCFILES Engine on Linux, UNIX, and Microsoft Windows</b> . . . . .	<b>173</b>
LIBNAME Statement for PCFILES Engine on Linux, UNIX, and Microsoft Windows . . . . .	174
Dictionary . . . . .	174

<b>Chapter 16 • Pass-Through Facility: PCFILES on Linux, UNIX, and Microsoft Windows . . . .</b>	<b>205</b>
Overview: Pass-Through Facility for PCFILES on Linux, UNIX, and Microsoft Windows . . . . .	.205
Dictionary . . . . .	.206

<b>Chapter 17 • Special Query Support . . . . .</b>	<b>219</b>
Special PCFILES Queries . . . . .	.219

## PART 5 ACCESS and DBLOAD Procedures 223

<b>Chapter 18 • The ACCESS Procedure for PC Files . . . . .</b>	<b>225</b>
Overview: The ACCESS Procedure for PC Files . . . . .	.226
SAS/ACCESS Descriptors for PC Files . . . . .	.227
Syntax: The ACCESS Procedure for PC Files . . . . .	.229
SAS Passwords for Descriptors . . . . .	.246
Performance and Efficient View Descriptors for PC Files . . . . .	.246

<b>Chapter 19 • The DBLOAD Procedure . . . . .</b>	<b>249</b>
Overview: DBLOAD Procedure . . . . .	.249
Syntax: The DBLOAD Procedure . . . . .	.250

<b>Chapter 20 • File-Specific Reference for the ACCESS and DBLOAD Procedures . . . . .</b>	<b>259</b>
Overview: ACCESS Procedure and DBLOAD Procedure . . . . .	.260
ACCESS Procedure: XLS Files . . . . .	.260
DBLOAD Procedure: XLS Specifics . . . . .	.266
ACCESS Procedure: WK <sub>n</sub> Specifics . . . . .	.271
DBLOAD Procedure: WK <sub>n</sub> Specifics . . . . .	.275
ACCESS Procedure: DBF Specifics . . . . .	.279
DBLOAD Procedure: DBF Specifics (Windows) . . . . .	.280
ACCESS Procedure: DIF Specifics . . . . .	.283
DBLOAD Procedure: DIF Specifics . . . . .	.286

## PART 6 Appendixes 289

<b>Appendix 1 • LIBNAME Statement for the JMP Engine . . . . .</b>	<b>291</b>
Dictionary . . . . .	.291

<b>Appendix 2 • The DBF and DIF Procedures . . . . .</b>	<b>293</b>
Overview: DBF and DIF Procedures . . . . .	.293
Dictionary . . . . .	.293

<b>Glossary . . . . .</b>	<b>301</b>
<b>Index . . . . .</b>	<b>307</b>



# Using This Document

---

## **Audience**

This document is intended for applications programmers and users who know how to use their operating environment, PC files, and basic SAS commands and statements. This document provides a general reference, as well as specific details, and SAS code examples that show how to access and use data in PC files directly from within SAS.



# What's New in SAS/ACCESS 9.3 Interface to PC Files

---

## Overview

SAS/ACCESS 9.3 Interface to PC Files enables you to exchange (import and export) PC files between the original source format and SAS data sets. Files are moved between the native PC format and SAS data sets via Import and Export procedures and wizards or through the use of LIBNAME statements.

---

## General Enhancements

General enhancements found in this release include the following.

- In SAS/ACCESS 9.2 Interface to PC Files, the Import and Export procedures and wizards were updated to include support for JMP files. This support is now included, by default, in Base SAS. This means that you no longer need a SAS/ACCESS Interface to PC Files license to access JMP files through the use of Import and Export procedures and wizards.
  - Support for the FMTLIB= option is available for JMP, SPSS, and Stata files.
  - The default port number used for TCP/IP server connections has changed from 8621 to 9621. LIBNAME and the Import and Export procedure commands default to PORT=9621 if the PORT option is omitted.
- 

## LIBNAME Engines

Starting in SAS 9.3, SAS/ACCESS 9.3 Interface to PC Files supports these LIBNAME engines:

- LIBNAME ACCESS engine for 32- and 64-bit Microsoft Windows operating systems
- LIBNAME EXCEL engine for 32- and 64-bit Microsoft Windows operating systems
- LIBNAME PCFILES engine for 32- and 64-bit Microsoft Windows operating system and for Linux and UNIX operating systems

- LIBNAME JMP engine for Linux, UNIX, and Microsoft Windows operating systems

---

## Import and Export Procedures and Wizards

- In this release, the Import procedure supports source type XLSX to read the Microsoft Excel 2007 and 2010 default file format (.xlsx) on Linux, UNIX, and Microsoft Windows operating systems.
- Import and Export procedures and wizards support the following source types in both 32- and 64-bit Microsoft Windows operating system.
  - Microsoft Access database files (\*.accdb, \*.mdb)
  - Microsoft Excel files (workbook: \*.xlsx, \*.xlsm, \*.xlsb; spreadsheet: \*.xls)
  - Microsoft Access database on PC Files Server
  - Microsoft Excel workbook on PC Files Server

---

## PC Files Server

Beginning with SAS/ACCESS 9.3 Interface to PC Files, the PC Files Server can be operated as a Windows service or as a Windows application on the 64-bit Windows operating system. This allows the server to take advantage of the associated 64-bit features and to operate more efficiently. The PC Files Server continues to operate on and support the Windows 32-bit operating system.

# Recommended Reading

---

- *SAS/ACCESS for Relational Databases: Reference*
- *SAS Language Reference: Concepts*
- *SAS Component Objects: Reference*
- *SAS Data Set Options: Reference*
- *SAS Formats and Informats: Reference*
- *SAS Functions and CALL Routines: Reference*
- *SAS Statements: Reference*
- *SAS System Options: Reference*
- *Base SAS Utilities: Reference*
- *SAS Macro Language: Reference*
- *Base SAS Procedures Guide*

For a complete list of SAS publications, go to [support.sas.com/bookstore](http://support.sas.com/bookstore). If you have questions about which titles you need, please contact a SAS Publishing Sales Representative:

SAS Publishing Sales  
SAS Campus Drive  
Cary, NC 27513-2414  
Phone: 1-800-727-3228  
Fax: 1-919-677-8166  
E-mail: [sasbook@sas.com](mailto:sasbook@sas.com)  
Web address: [support.sas.com/bookstore](http://support.sas.com/bookstore)



## **Part 1**

---

# Introduction to SAS/ACCESS 9.3 Interface to PC Files

### *Chapter 1*

**Working with SAS/ACCESS 9.3 Interface to PC Files** ..... 3



## Chapter 1

# Working with SAS/ACCESS 9.3 Interface to PC Files

<b>Methods for Accessing PC Files Data</b> .....	<b>3</b>
<b>Using This Document</b> .....	<b>4</b>
<b>Sample Data in This Document</b> .....	<b>4</b>

## Methods for Accessing PC Files Data

You can use SAS/ACCESS Interface to PC Files to read data from PC files for use in SAS reports or applications. You can use SAS data sets to create PC files in various formats. SAS/ACCESS Interface to PC Files includes accessing data in Microsoft Access database files and Excel workbook files. The Import and Export Wizards guide you through the steps to access your data.

The Import and Export Wizards as well as the IMPORT and EXPORT procedures facilitate data transfer between SAS data sets and several PC file formats including:

- Microsoft Access
- Microsoft Excel
- Lotus 1-2-3
- DBF
- JMP
- SPSS
- Stata
- Paradox

*Note:* JMP files support is included in Base SAS software. A SAS/ACCESS Interface to PC Files license is no longer needed.

Not every PC file format is available under every operating environment. For a list of supported data source and environments, see [“Supported Data Sources and Environments” on page 7](#).

To use LIBNAME ACCESS statement or LIBNAME EXCEL statement, you must have Microsoft ACE (Access Database Engine) software installed. See [“LIBNAME Statement Syntax” on page 97](#) for additional information.

To use LIBNAME PCFILES statement, you must also install SAS PC Files Server on Microsoft Windows.

To use the pass-through facility, see “[Pass-Through Facility on Microsoft Windows](#)” on [page 129](#).

You can directly read, update, or extract PC files data into a SAS data file. Use the ACCESS procedure with Microsoft Excel 4, 5, 95, Lotus 1-2-3 (WK1, WK3, WK4), DBF, and DIF file formats. See [Chapter 18, “The ACCESS Procedure for PC Files,”](#) on [page 226](#) for additional information.

The ACCESS procedure creates PC files and loads them with the data from a SAS data set. Use the DBLOAD procedure with any file formats that the ACCESS procedure supports. Both procedures are supported only for compatibility with SAS 6. Such SAS 6 limitations as the 8 character-long variable names apply. See [Chapter 19, “The DBLOAD Procedure,”](#) on [page 249](#) for additional information.

On Linux, UNIX, and Windows, using the DBF procedure, you can convert formatted data between dBase (DBF) files and SAS data sets. The DIF procedure enables you to convert between data interchange format (DIF) and SAS data sets. See “[Overview: DBF and DIF Procedures](#)” on [page 293](#) for additional information.

*Note:* The DBF procedure is also available under IBM z/OS.

---

## Using This Document

This document is intended for applications programmers and users with these skills.

- Know how to use their operating environment.
- Are familiar with their PC files.
- Know how to use basic SAS commands and statements.

This document provides a general reference, as well as specific details, and SAS code examples that show how to access and use data in PC files directly from within SAS.

---

## Sample Data in This Document

Examples in this document show how you can use SAS/ACCESS Interface to PC Files to read and write PC file data directly from SAS programs. They are not meant as examples for you to follow in designing files for any purpose. Sample data is available from <http://support.sas.com/kb/?ct=51000>. The data is based on a fictitious international textile manufacturer whose product line includes some special fabrics that they make to precise specifications. All data is fictitious.

## Part 2

---

# Import and Export Wizards and Procedures

<i>Chapter 2</i>	
<b>Supported Data Sources and Environments</b> .....	<i>7</i>
<i>Chapter 3</i>	
<b>Using the SAS Import and Export Wizards</b> .....	<i>11</i>
<i>Chapter 4</i>	
<b>The IMPORT Procedure</b> .....	<i>13</i>
<i>Chapter 5</i>	
<b>The EXPORT Procedure</b> .....	<i>21</i>
<i>Chapter 6</i>	
<b>File Format-Specific Reference for the IMPORT and EXPORT Procedures</b> .....	<i>27</i>



## Chapter 2

# Supported Data Sources and Environments

---

Reading and Writing Data between SAS Data Sets and PC Files . . . . .	7
Supported Data Sources and Environments . . . . .	7

---

## Reading and Writing Data between SAS Data Sets and PC Files

To read and write data between SAS data sets and external PC files see; [Chapter 4, “The IMPORT Procedure,”](#) on page 13, [Chapter 5, “The EXPORT Procedure,”](#) on page 21, and [“Using the SAS Import and Export Wizards”](#) on page 11.

Although the procedures provide similar capabilities, the wizards have a user interface. The procedures are code-based and support additional features.

---

## Supported Data Sources and Environments

The IMPORT and EXPORT procedures work within the limited range of available PC file formats if they reside locally on UNIX. The procedures work with a wider range of PC file formats if they reside locally on a PC.

The Import and Export Wizards and the IMPORT and EXPORT procedures are part of Base SAS software. If SAS/ACCESS Interface to PC files is not licensed, access is limited to JMP, CSV, TXT, and delimited files.

Alternate methods to access data are described in [“Methods for Accessing PC Files Data”](#) on page 3.

*Note:* The PC File Server version must match the SAS version.

**Table 2.1** Data Source & Environment Support Summary

Data Source		Import		Export		Supported Platforms
Description	Identifier	Wizard	PROC	Wizard	PROC	
Excel using LIBNAME EXCEL engine	EXCEL <sup>1</sup> EXCEL97 <sup>2</sup>	Yes	Yes	Yes	Yes	Microsoft Windows
Excel using PC Files Server	EXCELCS	Yes	Yes	Yes	Yes	Microsoft Windows, Linux, UNIX
Excel using .XLSX file formats	XLSX	No	Yes	No	Yes	Microsoft Windows, Linux, UNIX
Excel using .XLS file formats	XLS	No	Yes	No	Yes	Microsoft Windows, Linux, UNIX
Excel 4 using PROC ACCESS and PROC DBLOAD	EXCEL4	Yes	Yes	Yes	Yes	Microsoft Windows
Excel5 using PROC ACCESS and PROC DBLOAD	EXCEL5	Yes	Yes	Yes	Yes	Microsoft Windows
Access using LIBNAME ACCESS engine	ACCESS <sup>3</sup> ACCESS2000 <sup>4</sup>	Yes	Yes	Yes	Yes	Microsoft Windows
Access using PC Files Server	ACCESSCS	No <sup>5</sup>	Yes	No <sup>5</sup>	Yes	Microsoft Windows, Linux, UNIX
dBase using .DBF file formats	DBF	Yes	Yes	Yes	Yes	Microsoft Windows, Linux, UNIX
dBase file format with MEMO support	DBFMEMO	No	Yes	No	Yes	Microsoft Windows, Linux, UNIX
JMP using .JMP file formats	JMP	Yes	Yes	Yes	Yes	Microsoft Windows, Linux, UNIX
Paradox using .DB file formats	DB	Yes	Yes	Yes	Yes	Microsoft Windows, Linux, UNIX

Data Source		Import		Export		Supported Platforms
Description	Identifier	Wizard	PROC	Wizard	PROC	
SPSS using .SAV file formats	SAV	Yes	Yes	Yes	Yes	Microsoft Windows, Linux, UNIX
Stata using .DTA file formats	DTA	Yes	Yes	Yes	Yes	Microsoft Windows, Linux, UNIX
Lotus 1-2-3 using .WK1 file formats	WK1	Yes	Yes	Yes	Yes	Microsoft Windows
Lotus 1-2-3 using .WK3 file formats	WK3	Yes	Yes	Yes	Yes	Microsoft Windows
Lotus 1-2-3 using .WK4 file formats	WK4	Yes	Yes	Yes	Yes	Microsoft Windows
Comma-separated file	CSV	Yes	Yes	Yes	Yes	Microsoft Windows, Linux, UNIX, Open VMS
Tab-separated file	TAB	Yes	Yes	Yes	Yes	Microsoft Windows, Linux, UNIX, Open VMS
Delimiter-separated file	DLM	Yes	Yes	Yes	Yes	Microsoft Windows, Linux, UNIX, Open VMS
Data files using client/server model	PCFS	No	Yes	No	Yes	Microsoft Windows, Linux, UNIX

<sup>1</sup>Data Source Identifier EXCEL has alias names EXCEL2007 and EXCEL2010.

<sup>2</sup>Data Source Identifier EXCEL97 has alias names EXCEL2000, EXCEL2002, and EXCEL2003.

<sup>3</sup>Data Source Identifier ACCESS has alias names ACCESS2007 and ACCESS2010.

<sup>4</sup>Data Source Identifier ACCESS2000 has alias names ACCESS2002 and ACCESS2003.

<sup>5</sup>Yes for Microsoft Windows, for SAS 9.2 Phase 3 up to and excluding this release.



## Chapter 3

# Using the SAS Import and Export Wizards

---

<b>Importing and Exporting Data with the Wizards</b> .....	<b>11</b>
<b>Using the External File Interface (EFI)</b> .....	<b>11</b>

---

## Importing and Exporting Data with the Wizards

With the Import and Export Wizards, you can transfer data between external data sources and SAS data sets. Each wizard presents a series of windows with simple choices to guide you through the import or export process. Since there is no coding required, use of the wizards makes it easy to complete a complex or infrequently performed task.

- The Import Wizard guides you through the steps to read data from an external data source and write it to a SAS data set
- The Export Wizard guides you through the steps to read data from a SAS data set and write it to an external data source.

To start the Import Wizard, select **File** ⇒ **Import Data**.

To start the Export Wizard, select **File** ⇒ **Export Data**.

If you indicate that the source data type (for import or for export) is a user-defined format, after stepping through a series of windows, the External File Interface (EFI) opens to enable you to specify your data. Regardless of the type of source data, you can request that the wizard generate the procedure statements, which you can save to a file for subsequent use.

For each step, each wizard displays a window with available selections. To get help for any window or window item, select **Help**.

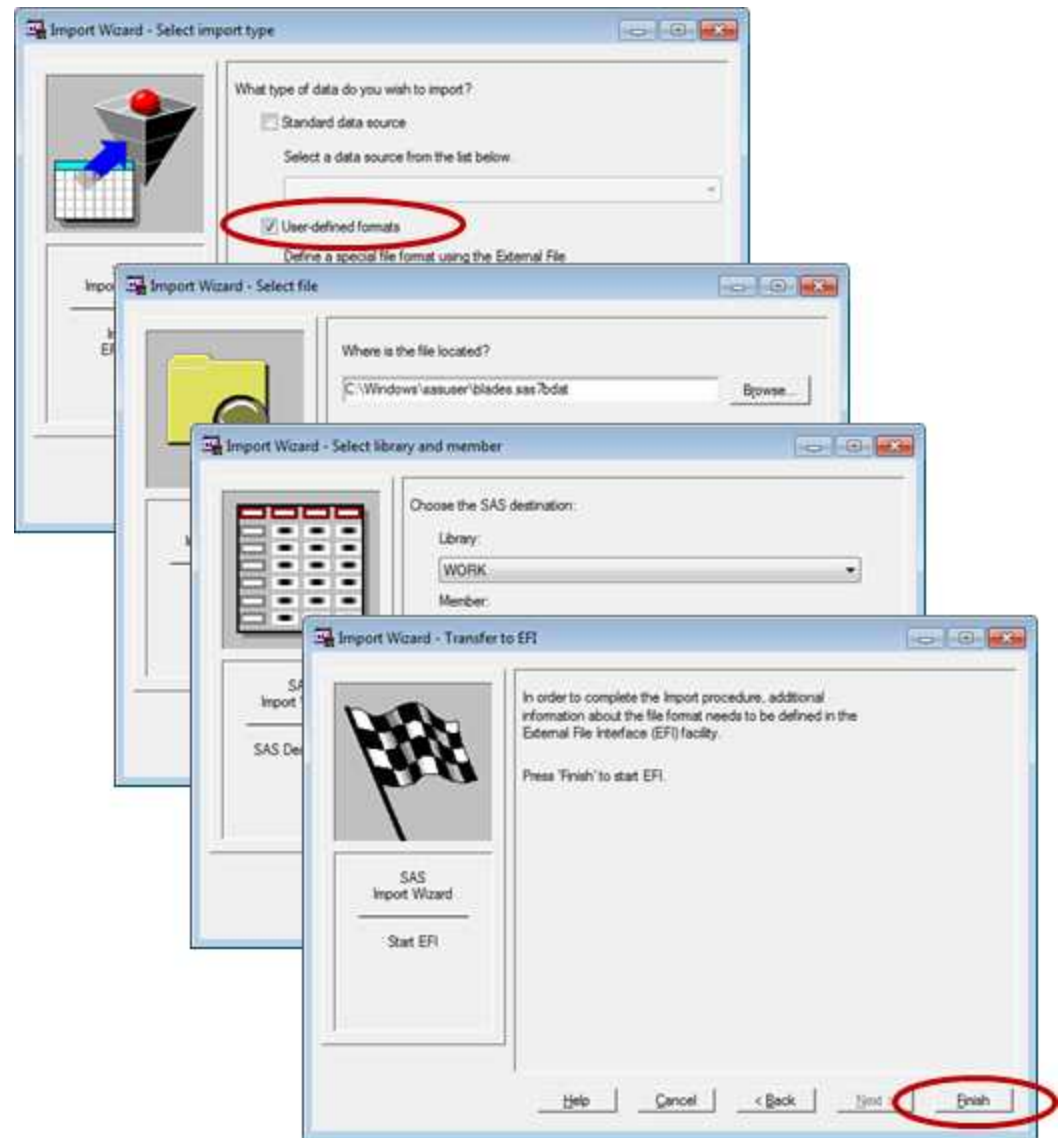
*Note:* Although you can use JAWS® to facilitate the use of the Import and Export wizard screens, the Import and Export wizards are not Section 508 compliant.

---

## Using the External File Interface (EFI)

The External File Interface (EFI) is a graphical user interface that enables you to read and write data that is not in a SAS internal format (that is, the source data is in a user-defined format).

From the Import Wizard or from the Export Wizard, if you select **User-defined format** from the data source selection window, after a series of windows that request specific information, the Import or Export Wizard prompts you to select **Finish** to invoke the EFI.



*Note:* To return to the Import or Export Wizard without invoking the EFI, select **Back**.

To close the Import or Export Wizard without invoking the EFI, select **Cancel**.

For instructions about using the EFI, select **Help** ⇒ **Using This Window**.

## Chapter 4

# The IMPORT Procedure

---

<b>Overview: PROC IMPORT</b> .....	<b>13</b>
<b>Syntax: The IMPORT Procedure</b> .....	<b>14</b>
PROC IMPORT Statement .....	14

---

## Overview: PROC IMPORT

The IMPORT procedure reads data from an external data source and writes it to a SAS data set. External data sources can include:

- Microsoft Access databases
- Microsoft Excel workbooks
- Lotus 1-2-3 spreadsheets
- Paradox files
- SPSS files
- Stata files
- dBase
- JMP files
- delimited files

Delimited files contain columns of data values that are separated by a delimiter, such as a blank, a comma, or a tab.

The SAS variable definitions are based on the input records. The IMPORT procedure imports the data using one of the following methods:

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

Customize the results with statements and options that are specific to the input data source. The IMPORT procedure generates a SAS data set and writes information about the import to the SAS log. The DATA step or the SAS/ACCESS code that the IMPORT procedure generates is written to the log. If a translation engine is used, no code is submitted.

To import data, you can use the Import Wizard. This windowing tool guides you through the steps to import an external data source. You can request that the Import Wizard generate IMPORT procedure statements. You can save these statements for subsequent use. To open the Import Wizard, select **File** ⇒ **Import Data** from the SAS windowing environment.

---

## Syntax: The IMPORT Procedure

```
PROC IMPORT
DATAFILE =<'filename'> | DATATABLE= <'tablename'>
```

---

### PROC IMPORT Statement

The IMPORT procedure reads external data and writes the data to a SAS data set.

---

#### Syntax

```
PROC IMPORT
DATAFILE= <'filename'> | DATATABLE= <'tablename'> (Not used for Microsoft Excel files)
<DBMS>= <data-source-identifier>
<OUT>= <libref.SAS data-set-name> <SAS data-set-option(s)>
<REPLACE>;
<file-format-specific-statements>
```

#### Required Arguments

##### **DATAFILE=filename**

specifies the complete path and filename or fileref for the input file. A fileref is a SAS name that is associated with the physical location of the output file. To assign a fileref, use the FILENAME= statement. You can omit the quotation marks if the *filename* does not include certain characters such as these:

- backslash
- lowercase characters
- spaces

**Alias:** FILE

**Default:** character

##### **Restrictions:**

The IMPORT procedure does not support device types or access methods for the FILENAME statement except for DISK. For example, the IMPORT procedure does not support the TEMP device type, which creates a temporary external file.

When running SAS/ACCESS on UNIX, to access data stored on a PC server, specify the full path and filename of the import file. The use of a fileref is not supported.

The IMPORT procedure can import data if the data type is supported by SAS. SAS supports numeric and character types of data but not (for example, binary objects). If the data that you want to import is a type that SAS does not support, the IMPORT procedure might not import it correctly. In many cases, the

procedure attempts to convert the data to the best of its ability. However, at times this is not possible.

**Interaction:** For some input data sources such as a Microsoft Excel workbook, the first eight rows of data are scanned. The most prevalent data type (numeric or character) is used for a column. This is the default. If most of the data in the first eight rows is missing, SAS defaults to data type (character) and any subsequent numeric data for that column is set to missing.

**Notes:**

For information about how SAS converts data types, see the specific information for the data source file format that you are importing.

To import DBF files created with Microsoft Visual FoxPro, you must export to an appropriate dBASE format using Visual FoxPro. Import the dBASE file to SAS.

**See:** The FILENAME statement in *SAS Statements: Reference*.

**DATATABLE= 'table-name'**

specifies the table name of the input DBMS table. If the name does not include special characters, such as question marks, lowercase characters, or spaces, you can omit the quotation marks. The DBMS table name might be case sensitive and is generally used for MSACCESS tables, but not for Microsoft Excel sheets.

**Alias:** TABLE

**Requirements:**

When importing Microsoft Access tables, SAS/ACCESS converts the table name to a SAS member name. SAS does not support member names longer than 32 bytes.

When you import a DBMS table, you must specify the DBMS= option.

## Optional Arguments

### *SAS data-set-option(s)*

Specify SAS data set options. For example, to assign a password to the resulting SAS data set, you can use the ALTER=, PW=, READ=, or WRITE= data set options. To import only data that meets a specified condition, you can use the WHERE data set option. For information about all SAS data set options, see *SAS Data Set Options: Reference*.

### **DBMS= data-source-identifier**

specifies the type of data to import. To import a DBMS table, specify DBMS= using a supported database identifier listed in [Table 4.1 on page 15](#). For example, DBMS=ACCESS specifies to import a Microsoft Access 2000, 2002, 2003, or 2007 database. All DBMS= specifications refer to local access, except where noted in this table.

*Note:* Transcoding is not supported for DBMS=XLS. Attempted execution of this operation yields unpredictable results. As an alternative, use DBMS=EXCEL or DBMS=EXCELCS with PC Files Server.

**Table 4.1** DBMS Specifications

Data Source Identifier	Output Data Source	File Extension
ACCESS	Microsoft Access 2000, 2002, 2003, or 2007 table using the LIBNAME statement.	.mdb .accdb

Data Source Identifier	Output Data Source	File Extension
ACCESSCS	Microsoft Access table connecting remotely through PC Files Server	.mdb .accdb
CSV	Delimited file with comma-separated values	.csv
DBF	dBASE 5.0, IV, III+, and III files	.dbf
DBFMEMO	dBASE 5.0, IV, III+, and III files with memos FoxPro and Visual FoxPro files with memos	.dbf .fpt .dbt
DLM	Delimited file (default delimiter is a blank)	.*
DTA	Stata file	.dta
EXCEL	Microsoft Excel 97, 2000, 2002, 2003, or 2007 workbook using the LIBNAME statement.	.xls .xlsb .xlsm .xlsx
EXCEL4 EXCEL5	Microsoft Excel 4.0, Excel 5.0 or 7.0 (95) workbook.	.xls
EXCELCS	Microsoft Excel workbook connecting remotely through PC Files Server.	xls, .xlsb
JMP	JMP Files	.jmp
PARADOX	Paradox .DB files	.db
PCFS	JMP files, SPSS files, and Stata files connecting remotely through PC Files Server.	.jmp, .sav, .dta
SAV	SPSS file	.sav
TAB	Delimited file (tab-delimited values)	.txt
WK1	Lotus1-2-3 Release 2 spreadsheet	.wk1
WK3	Lotus 1-2-3 Release 3 spreadsheet	.wk3
WK4	Lotus 1-2-3 Release 4 or 5 spreadsheet	.wk4
XLS	Microsoft Excel 5.0, 95, 97, 2000, 2002, or 2003 workbook using file formats	.xls

Data Source Identifier	Output Data Source	File Extension
XLSX	Microsoft Excel 2007 or 2010 workbook using file formats  <i>Note:</i> Transcoding is not supported for DBMS=XLS. Attempted execution of this operation yields unpredictable results. Use DBMS=EXCEL or DBMS=EXCELCS with PC Files Server as an alternative.	.xlsx

*Note:* All DBMS= specifications refer to local access, except for these specifications:

- DBMS=ACCESSCS
- DBMS=EXCELCS
- DBMS=PCFS

These files are accessed remotely by connecting to PC Files Server on Microsoft Windows.

#### Microsoft Excel

When you specify DBMS=XLS or DBMS=XLSX for an Excel file, you can read and write to Excel workbooks under UNIX directly without having to access the PC Files Server. The following example demonstrates the use of DBMS=XLSX specifying a range of cells.

```
proc import datafile="fieldtypes.xlsx"
  out=small dbms=xlsx;
  range=colsb_d;
run;
```

Microsoft Excel 97, 2000, 2002, and 2003 share the same internal file formats. The SAS LIBNAME engine recognizes EXCEL97, EXCEL2000, EXCEL2002, EXCEL2003, EXCEL2007, and EXCEL2010 as aliases for the identifier EXCEL. By specifying DBMS=EXCEL, the IMPORT procedure can read any version of these files that are saved in Microsoft Excel workbooks.

**Table 4.2** Microsoft Excel Workbook Specifications

Identifier	Excel 2007, 2010	Excel 97, 2000, 2002, 2003	Excel 5.0, 95	Excel 4.0
XLS	No	Yes	Yes	No
XLSX	Yes	No	No	No
EXCEL	Yes	Yes	Yes	Yes
EXCEL5	No	No	Yes	Yes
EXCEL4	No	No	Yes	Yes

**Table 4.3** DBMS Specifications for Excel

DBMS	Uses	Requires	Operating Platform
EXCEL	SAS Excel LIBNAME engine	Microsoft ACE or Jet Provider	Microsoft Windows
XLS	File formats technology		Microsoft Windows, Linux, UNIX
XLSX	File formats technology		Microsoft Windows, Linux, UNIX
EXELCS	PC Files LIBNAME engine	PC Files Server Excel Driver on Microsoft Windows	Microsoft Windows, Linux, UNIX

**PCFS**

Specify DBMS=PCFS for JMP, SPSS, and Stata files to use the client/server model. This enables you to access data on Microsoft Windows from Linux, UNIX, or other Microsoft Windows operating environments. These files are accessed remotely by connecting to a PC Files Server on Microsoft Windows.

**Microsoft Access**

Microsoft Access versions 2000, 2002, and 2003 share the same internal file formats. The SAS LIBNAME engine recognizes ACCESS2000, ACCESS2002, ACCESS2003, ACCESS2007, and ACCESS2010 as aliases for the identifier ACCESS. By specifying DBMS=ACCESS, SAS can read any of these versions of files that are saved in Microsoft Access applications.

To import a SAS data from an existing Microsoft Access database, the IMPORT procedure can read existing Access 97, Access 2000, Access 2002, or Access 2003 database files. If you specify DBMS=ACCESS2000 and the database is in Access 97 format, the IMPORT procedure imports the table, and the database remains in Access 97 format.

When the DATABASE= option is specified for an Access database .mdb file that does not exist, a database is created using the format specified in the DBMS= option. If you specify DBMS=ACCESS to create a file, the result is an MDB file that Access 2000, 2002, and 2003 can read. Access 97 cannot read this file.

For more information about the DATABASE= option, see [“Microsoft Access Database Files”](#) on page 52.

Access 2007 can open all formats. Only Access 2007 and later can open Access 2007 file formats.

**Table 4.4** Access Table Specifications

Identifier	Access 2007, 2010	Access 2000, 2002, 2003
ACCESS	Yes	Yes

Identifier	Access 2007, 2010	Access 2000, 2002, 2003
ACCESS2007	Yes	Yes

**Restriction:** The availability of a data source depends on the operating environment and, in some cases, the platform and whether your site has a SAS/ACCESS Interface for PC Files license. If your site does not have a license, only delimited files and JMP files are supported.

**See:** [“Supported Data Sources and Environments”](#) on page 7.

**OUT=libref.SAS data-set**

identifies the output SAS data set with either a one- or two-level SAS name (library and member name). If the specified SAS data set does not exist, The IMPORT procedure creates it. If you specify a one-level name, by default the IMPORT procedure uses either the SASUSER library if assigned or the WORK library if SASUSER not assigned.

**REPLACE**

overwrites an existing SAS data set. If you do not specify REPLACE, the IMPORT procedure does not overwrite an existing file.

*<file-format-specific-statements>*

see [“File Format-Specific Reference for the IMPORT and EXPORT Procedures”](#) on page 29 for the supported syntax for your DBMS.



## Chapter 5

# The EXPORT Procedure

---

<b>Overview: PROC EXPORT</b> .....	<b>21</b>
<b>Syntax: The EXPORT Procedure</b> .....	<b>22</b>
PROC EXPORT Statement .....	22

---

## Overview: PROC EXPORT

The EXPORT procedure reads data from a SAS data set and writes it to an external data source. External data sources can include:

- Microsoft Access database files
- Microsoft Excel workbook files
- Lotus 1–2–3 spreadsheet files
- Paradox files
- SPSS files
- Stata files
- dBase files
- JMP files
- delimited files

Delimited files contain columns of data values that are separated by a delimiter such as a blank or a comma.

The EXPORT procedure reads the input file and writes the data to an external data source. The EXPORT procedure exports the data using one of these methods:

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

Customize the results with statements and options that are specific to the output data source. The EXPORT procedure generates the specified output file and writes information about the export to the SAS log. The DATA step or the SAS/ACCESS code that the EXPORT procedure generates is written to the log. If a translation engine is used, no code is submitted.

To export data, you can use the Export Wizard, which is a windowing tool that guides you through the steps to export to an external data source. You can request that the Export Wizard generate EXPORT procedure statements, which you can save for subsequent use. To open the Export Wizard, select **File** ⇒ **Export Data** from the SAS windowing environment.

---

## Syntax: The EXPORT Procedure

```
PROC EXPORT DATA=<libref.>SAS data set <OUTFILE="filename"
| OUTTABLE="tablename";
```

---

### PROC EXPORT Statement

The EXPORT procedure reads a SAS data set and writes the data to an external data file.

---

#### Syntax

```
PROC EXPORT DATA=<libref.>SAS data set <(SAS data set options)>
OUTFILE="filename" | OUTTABLE="tablename"
<DBMS=identifier> <REPLACE> <LABEL>;
<file-format-specific-statements>
```

#### Required Arguments

**DATA**=<libref.> SAS data set

specifies the input SAS data set with either a one- or two-level SAS name (library and member name). If you specify a one-level name, by default, the EXPORT procedure uses either the SASUSER library (if assigned) or the WORK library (if SAS system option USER is not assigned).

**Default:** If you do not specify a SAS data set, the EXPORT procedure uses the most recently created SAS data set. SAS keeps track of data set order with the system variable `_LAST_`. To ensure that the EXPORT procedure uses the correct data set, identify the SAS data set with a two-level name.

**Restriction:** The EXPORT procedure can export data if the data format is supported and the amount of data is within the limitations of the data source. Some data sources have a maximum number of rows or columns. If the data that you want to export exceeds the limits of the data source, the EXPORT procedure might not be able to export it correctly. When SAS encounters incompatible formats, the procedure formats the data to the best of its ability.

**OUTFILE**= *filename*

specifies the complete path and filename, or a fileref for the output PC file, spreadsheet, or delimited external file. If the name does not include special characters (such as question marks), lowercase characters, or spaces, omit the quotation marks.

**Alias:** FILE

**Restrictions:**

The EXPORT procedure does not support device types or access methods for the FILENAME statement except for DISK. For example, the EXPORT procedure does not support the TEMP device type, which creates a temporary external file.

For client/server applications: Specify the full path and filename of the import file when you are running SAS/ACCESS software on UNIX to access data that is stored on a PC server. Use of a fileref is not supported.

**OUTTABLE=** *table-name*

specifies the DBMS output table. If the name does not include special characters (such as question marks), lowercase characters, or spaces, omit the quotation marks. The DBMS table name might be case sensitive.

**Alias:** TABLE

**Restriction:** Used for only MSACCESS files.

**Note:** For PC files the table name is generally used for Microsoft Access databases.

### Optional Arguments

**<SAS data-set-option(s)>**

specifies SAS data set options. For example, if the data set that you are exporting has an assigned password, you can use the ALTER= option, the PW= option, the READ= option, or the WRITE= option. To export only data that meets a specified condition, you can use the WHERE= data set option. For information about SAS data set options, see “Data Set Options” in *SAS Language Reference: Dictionary*.

**DBMS=** *data-source-identifier*

DBMS= specifies the type of external data source the EXPORT procedure creates. To export to a DBMS table, specify DBMS= using a supported database identifier. For example, DBMS=ACCESS specifies to export a table into a Microsoft Access 2000, 2002, 2003, or 2007 database.

*Note:* Transcoding is not supported for DBMS=XLS. The output yields unpredictable results. Use DBMS=EXCEL as an alternative.

**Table 5.1** Data Source Identifier Summary

Data Source Identifier	Output Data Source	File Extension
ACCESS	Microsoft Access 2000, 2002, 2003, 2007, or 2010 table (using the LIBNAME statement)	.mdb .accdb
ACCESSCS	Microsoft Access table connecting remotely through PC Files Server	.mdb .accdb
CSV	delimited file (comma-separated values)	.csv
DBF	dBASE 5.0, IV, III+, and III files	.dbf
DBFMEMO	dBASE 5.0, IV, III+, and III files with memos FoxPro and VisualPro with memos	.dbf .fpt
DLM	delimited file (default delimiter is a blank)	.
DTA	Stata file	.dta
EXCEL	Excel 97, 2000, 2002, 2003, 2007 or 2010 workbook (using the LIBNAME statement)	.xls .xlsb .xlsx

Data Source Identifier	Output Data Source	File Extension
EXCEL4	Excel 4.0 workbook (using PROC DBLOAD)	.xls
EXCEL5	Excel 5.0 or 7.0 (95) workbook (using PROC DBLOAD)	.xls
EXCELCS	Excel workbook connecting remotely through PC Files Server	.xls .xlsb
JMP	JMP files	.jmp
PARADOX	Paradox DB files	.db
PCFS	JMP files, Stata files, and SPSS files connecting remotely through PC Files Server	.jmp, .dta, .sav
SAV	SPSS files, compressed and uncompressed binary files	.sav
TAB	delimited file (tab-delimited values)	.txt
WK1	Lotus 1-2-3 Release 2 spreadsheet	.wk1
WK3	Lotus 1-2-3 Release 3 spreadsheet	.wk3
WK4	Lotus 1-2-3 releases 4 and 5 spreadsheet	.wk4
XLS	Excel 97, 2000, 2002, or 2003 spreadsheet (using file formats)  <i>Note:</i> Transcoding is not supported for DBMS=XLS. The output yields unpredictable results. Use DBMS=EXCEL as an alternative.	.xls

When you specify a value for DBMS=, consider the following for specific data sources:

- When you specify DBMS=XLS for an Excel file, you can read and write to Excel workbooks under Microsoft Windows and UNIX directly without having to access the PC Files Server.
- Specify DBMS=PCFS for JMP, SPSS, and Stata files to use the client/server model. This enables you to access data on Microsoft Windows from Linux, UNIX, or Microsoft Windows 64-bit operating environments. These files are accessed remotely by connecting to a PC Files Server on Microsoft Windows.
- Microsoft Access versions 2000, 2002, and 2003 share the same internal file formats. The SAS LIBNAME engine recognizes ACCESS 2000, ACCESS 2002, ACCESS 2003, ACCESS 2007, and ACCESS 2010 as aliases for the identifier ACCESS. By specifying DBMS=ACCESS, the SAS export file can be read by any of these versions of files that are saved in Microsoft Access applications.
- Microsoft Excel 97, 2000, 2002, and 2003 share the same internal file formats. The SAS LIBNAME engine recognizes EXCEL97, 2000, 2002, 2003, and EXCEL2007 as aliases for the identifier EXCEL. By specifying DBMS=EXCEL, the SAS export file can read any of these versions of files that are saved in Microsoft Excel workbooks.

- To export a SAS data set to an existing Microsoft Access database, the EXPORT procedure can write to existing Access 97, Access 2000, Access 2002, or Access 2003 database files. If you specify DBMS=ACCESS2000 and the database is in Access 97 format, the EXPORT procedure exports the table, and the database remains in Access 97 format.

When the DATABASE= option is specified for an Access database .mdb file that does not exist, a database is created using the format specified in the DBMS= option. If you specify DBMS=ACCESS to create a file, the result is an MDB file that Access 2000, 2002, and 2003 can read. Access 97 cannot read this file.

For more information about the DATABASE= option, see [“Microsoft Access Database Files”](#) on page 52.

The files created by SAS can be opened and read by various Microsoft Access versions, as indicated in the following table.

**Table 5.2** Exported Data: Microsoft Access Readability

Identifier	Access 2007, 2010	Access 2000, 2002, 2003
ACCESS	Yes	Yes
ACCESS2007	Yes	Yes

#### ACCESS 2007

*Note:* Only Access 2007 and later can open Access 2007 file formats.

- To export a Microsoft Excel spreadsheet, the EXPORT procedure creates an XLS file for the specified version. When exporting to an existing Excel workbook .XLS file a .BAK file is created.

The files created by SAS can be opened and read by various Microsoft Excel versions, as indicated in the following table.

**Table 5.3** Exported Data: Microsoft Excel Workbook Readability

Identifier	Excel 2007, 2010	Excel 97, 2000, 2002, 2003
XLS	Yes	Yes
EXCEL	Yes	Yes
EXCEL5	Yes	Yes
EXCEL4	Yes	Yes

- Missing values are translated to blanks when you export a SAS data set to a dBASE file (DBF).
- Due to dBASE limitations, character variable values longer than 256 characters are truncated in the resulting dBASE file.

#### Restrictions:

The availability of an output data source depends on:

Only Excel 2007 and later can open Excel 2007 formats.

**Note:** All DBMS= specifications refer to local access, except for: These files are accessed remotely by connecting to PC Files Server on Microsoft Windows.

**See:** [“File Format-Specific Reference for the IMPORT and EXPORT Procedures” on page 29](#)

**LABEL**

writes SAS label names as column names to the exported table. If SAS label names do not exist, then the variable names are used as column names in the exported table.

**Alias:** DBLABEL

**REPLACE**

overwrites an existing file. For a Microsoft Access database or an Excel workbook, REPLACE overwrites the target table or spreadsheet. If you do not specify REPLACE, the EXPORT procedure does not overwrite an existing file.

*<file-format-specific-statements>*

see [“File Format-Specific Reference for the IMPORT and EXPORT Procedures” on page 29](#) for the supported syntax for your DBMS.

## Chapter 6

# File Format-Specific Reference for the IMPORT and EXPORT Procedures

---

<b>Delimited Files</b> .....	<b>29</b>
Overview .....	29
CSV Files .....	29
Tab-Delimited Files .....	29
Other Delimiters .....	29
External File Interface (EFI) .....	29
IMPORT and EXPORT Procedure Statements for Delimited Files .....	30
Example 1: Import a Tab-Delimited File into SAS .....	32
Example 2: Import a Space-Delimited File into SAS .....	32
Example 3: Export a SAS Data Set to a CSV File .....	32
Example 4: Import a Subset of a CSV File into SAS .....	32
<b>Microsoft Excel Workbook Files</b> .....	<b>33</b>
Microsoft Excel Files Essentials .....	33
Excel Data Types .....	34
Excel Numeric Data and Time Values .....	34
Excel File Formats .....	35
SAS Import and Export Utilities Support for Excel Files .....	35
Importing and Exporting Microsoft Excel 4 and Excel 5 Files .....	46
Example 1: Import a SAS Data Set to an Excel 5 File .....	48
Example 2: Export a SAS Data Set to an Excel 5 File .....	48
Import and Export Microsoft Excel Files Using XLS and XLSX File Formats . . .	48
Example 1: Export SAS Data Sets to Excel Workbook Files .....	51
Example 2: Import Data Using a Range Name .....	51
Example 3: Import Data Using an Absolute Range Address .....	52
<b>Microsoft Access Database Files</b> .....	<b>52</b>
Microsoft Access File Essentials .....	52
Microsoft Access Data Types .....	53
The Conversion of Date and Time Values between SAS Data Sets and Microsoft Access Database .....	54
IMPORT and EXPORT Procedure Statements for Access Files .....	54
<b>Lotus 1-2-3 WKn Files</b> .....	<b>62</b>
WKn Files Essentials .....	62
WKn Data Types .....	63
Supported Import and Export Methods and Statements for WKn Files .....	63
Example 1: Export a SAS Data Set to a WK4 File .....	65
Example 2: Import Data from a SAS Data Set from a WK4 File .....	65
Example 3: Export Data to a WK1 File from a SAS Data Set .....	65
Example 4: Import Data from a WK1 File into a SAS Data Set .....	66
<b>dBase DBF Files</b> .....	<b>66</b>

dBase DBF Files Essentials . . . . .	66
DBF Data Types . . . . .	66
Setting Environment Variables and System Options . . . . .	68
Supported SAS IMPORT and EXPORT Procedure Statements . . . . .	68
Example 1: Export Data to a DBF File from a SAS Data Set . . . . .	69
Example 2: Import Data from a DBF File into a SAS Data Set . . . . .	69
Example 3: Export Data to a DBF File from a SAS Data Set Using Encoding . . . . .	70
Example 4: Import and Translate Data from a DBF File . . . . .	70
<b>dBase DBF MEMO Files . . . . .</b>	<b>70</b>
Overview . . . . .	70
Import Data from a DBF File with Memo Field into a SAS Data Set . . . . .	71
<b>JMP Files . . . . .</b>	<b>71</b>
JMP File Essentials . . . . .	71
JMP Missing Values . . . . .	71
JMP Data Types . . . . .	71
Importing and Exporting JMP Files Data . . . . .	73
IMPORT Procedure and EXPORT Procedure Supported Syntax . . . . .	73
Example 1: Export a SAS Data Set to a JMP File . . . . .	73
Example 2: Export a SAS Data Set on UNIX to a JMP File . . . . .	73
Example 3: Import a SAS Data Set from a JMP File . . . . .	74
Example 4: Export a SAS Data Set on UNIX to a JMP File on Microsoft Windows . . . . .	74
Example 5: Import Data from a JMP File on Microsoft Windows to a SAS Data Set on UNIX . . . . .	74
<b>Paradox DB File Formats . . . . .</b>	<b>74</b>
Paradox File Essentials . . . . .	74
Export a SAS Data Set to a PARADOX DB File . . . . .	75
Import a SAS Data Set from a Paradox DB File . . . . .	75
<b>SPSS SAV Files . . . . .</b>	<b>75</b>
SAV File Essentials . . . . .	75
SPSS Data Types . . . . .	75
Importing and Exporting Data in SPSS Files . . . . .	77
Import Procedure and the Export Procedure Supported Syntax . . . . .	77
Example 1: Export a SAS Data Set to an SPSS SAV File . . . . .	77
Example 2: Import a SAS Data Set from an SPSS SAV File . . . . .	77
Example 3: Import Data from an SPSS File and Apply FMTLIB= Option . . . . .	77
Example 4: Export a SAS Data Set on UNIX to an SPSS File on Microsoft Windows . . . . .	78
Example 5: Import Data from an SPSS File on Microsoft Windows to a SAS Data Set on UNIX . . . . .	78
<b>Stata DTA Files . . . . .</b>	<b>78</b>
DTA Files Essentials . . . . .	78
DTA Data Types . . . . .	78
Importing and Exporting Stata Data Files . . . . .	80
Import and Export Procedures Supported Syntax . . . . .	80
Example 1: Export a SAS Data Set to a Stata File on a Local System . . . . .	80
Example 2: Import a SAS Data Set from a Stata File on a Local System . . . . .	80
Example 3: EXPORT a SAS Data Set on UNIX to a Stata File on Microsoft Windows . . . . .	80
Example 4: Import Data from a Stata File on Microsoft Windows to a SAS Data Set on UNIX . . . . .	81

---

## Delimited Files

### Overview

In computer programming, a delimited text file is a file in which the individual data values contain embedded delimiters, such as quotation marks, commas, and tabs. A delimiter is a character that separates words or phrases in a text string that defines the beginning or end of a contiguous string of character data.

- The delimiter is not considered part of the character data string.
- The first row of data is usually read as column headings.
- The column headings are then converted to SAS variable names.
- A line character indicates a new row.

A delimited text file is also called a delimiter-separated values file (CSV or DSM).

*Note:* Support of delimited files is included in Base SAS. The SAS/ACCESS to PC Files license is not needed to use this list of features.

### CSV Files

A comma-separated values file is a form of a delimited file. The data values are separated by commas. In a CSV-type file, each line can represent one of these items:

- an entry
- a record
- a row
- an observation in a database management system
- other applications

### Tab-Delimited Files

A tab-delimited file is a form of delimited file. The data values are separated by control characters that represent the TAB key. The data values form columns of a database table. The columns can be exported to a database table.

### Other Delimiters

Files that have other delimiters such as spaces or semicolons are also known as delimited text files or delimited files.

### External File Interface (EFI)

The SAS Import and Export Wizards use the SAS External File Interface methods to read and write data in delimited external files. Be aware of these behaviors when using the wizards and procedures to import or export data in delimited files.

- When data values are enclosed in quotation marks, delimiters within the value are treated as character data.
- Quotation marks are removed from character values.
- Two consecutive delimiters indicate a missing value.
- A delimiter can be specified as one or more characters.
- While exporting data, the EXPORT procedure discards items that exceed the output line length. See the DROPOVER= option in the FILE statement in *SAS Statements: Reference*.
- The delimiter can be in binary form. For example: `delimiter='09'x`
- As the IMPORT procedure reaches the end of the current input data row, variables without any values are set to missing.

### See Also

- [“Using the SAS Import and Export Wizards” on page 11](#)
- [Chapter 4, “The IMPORT Procedure,” on page 13](#)

## IMPORT and EXPORT Procedure Statements for Delimited Files

The supported file types are CSV (comma-separated values), TAB (tab-separated values), and DLM (delimiter-separated values).

See: [“Example 1: Import a Tab-Delimited File into SAS” on page 32](#) and [“Example 3: Export a SAS Data Set to a CSV File” on page 32](#)

**Table 6.1** IMPORT and EXPORT Procedure Statements

Delimited File Type	Statement Options	PROC Import	PROC Export	Valid Value	Default Value
CSV and TAB	DATAROW	Yes	No	1 to 2147483647	Depends on GETNAMES= option value
	GETNAMES	Yes	No	Yes   No	Yes
	GUESSINGROWS	Yes	No	1 to 2147483647	20
	PUTNAMES	No	Yes	Yes   No	Yes
DLM	DATAROW	Yes	No	1 to 2147483647	2
	DELIMITER	Yes	Yes	'char'   'nn'x	''
	GETNAMES	Yes	No	Yes   No	Yes
	GUESSINGROWS	Yes	No	1 to 2147483647	20
	PUTNAMES	No	Yes	Yes   No	Yes

**DATAROW= *n***

specifies the row number where the IMPORT procedure starts reading data.

**Default:** When GETNAMES=NO: 1; when GETNAMES=YES: 2

**Range:** 1 to 2147483647

**Restrictions:**

If GETNAMES = Yes, then DATAROW must be greater than or equal to 2.

If GETNAMES = No, then DATAROW must be greater than or equal to 1.

**DELIMITER= '*char*' | '*nn*'*x***

specifies the delimiter (either a single character or hexadecimal value) that separates the columns of data for the IMPORT and EXPORT procedures.

**Default:** A blank character

**Restriction:** Support only for DLM type files.

**GETNAMES= *YES* | *NO***

specifies whether the IMPORT procedure is to generate SAS variable names from the data values in the first row of the import file. If a data value in the first record contains special characters that are not valid in a SAS name, SAS converts the character to an underscore. For example, The column name *Occupancy Code* becomes the SAS variable name *Occupancy\_Code*.

*YES* specifies that the IMPORT procedure generate SAS variable names from the data values in the first row of the imported Excel file.

*NO* specifies that the IMPORT procedure generate SAS variable names as F1, F2, F3, and so on.

**Default:** Yes

**Restriction:** Valid only for the IMPORT procedure.

**GUESSINGROWS= *n***

specifies the number of rows that the IMPORT procedure is to scan to determine the appropriate data type for the columns. The scan process scans from row 1 to the row number that is specified by GUESSINGROWS = option.

The default row number can be changed in the SAS REGISTRY as follows: From the SAS menu, **Solutions** ⇒ **Accessories** ⇒ **Registry Editor**.

When the Registry Editor opens, select **Products** ⇒ **BASE** ⇒ **EFI** ⇒

**GuessingRows**. This opens the Edit Signed Integer Value window, where you can modify the **Value Data** item.

**Default:** 20

**Range:** 1 to 2147483647

**PUTNAMES= *YES* | *NO***

*YES* specifies that the EXPORT procedure is to do the following tasks:

- Write the SAS variable names to the first row of the exported data file as column headings.
- Write the first row of the SAS data set to the second row of the exported data file.

*NO* specifies that the EXPORT procedure is to write the first row of SAS data set values to the first row of the exported data file.

**Default:** Yes

**Restriction:** Valid only for the EXPORT procedure.

**Note:** If you specify the LABEL= option, the SAS variable labels (not the variable names) are written as column headings.

**Example 1: Import a Tab-Delimited File into SAS**

This code illustrates how the IMPORT procedure uses the first row of the tab delimited file to generate SAS variable names. SAS starts to read data from row 2, and scans 10 rows of data to determine data types for each column. The *invoice.txt* file saves data with the tab character ('09'x) as the delimiter.

```
PROC IMPORT OUT=WORK.TEST
  FILE="&dlmdir.\invoice.txt"
  DBMS=TAB REPLACE;
  GETNAMES=YES;
  DATAROW=2;
  GUESSINGROWS=10;
RUN;
```

**Example 2: Import a Space-Delimited File into SAS**

The IMPORT procedure generates generic variable names such as VAR1 and VAR2. It starts to read data from row 2, and scans the default number of rows (20) to determine the data types for each column. '20'x is the hexadecimal value for a space in ASCII code.

```
PROC IMPORT OUT=WORK.TEST
  DATAFILE="&dlmdir.\invoice.txt"
  DBMS=DLM REPLACE;
  DELIMITER='20'x;
  GETNAMES=NO;
  DATAROW=2;
RUN;
```

**Example 3: Export a SAS Data Set to a CSV File**

The EXPORT procedure exports the SAS data set, SDF.INVOICE, to a CSV file; *invoice.csv*. The SAS variable name is not used. The first row of the SAS data set is written to the first row of the CSV file.

```
LIBNAME SDF "&sasdir";
PROC EXPORT DATA=SDF.INVOICE
  OUTFILE='c:\temp\invoice.csv'
  DBMS=DLM REPLACE;
  DELIMITER=',';
  PUTNAMES=NO;
RUN;
```

**Example 4: Import a Subset of a CSV File into SAS**

The IMPORT procedure starts to read data in row 6. It reads ten observations from the selected columns in the customer CSV file. The global OBS= option limits the number of data rows to import. The OBS=MAX option resets the OBS= option to the default value.

```
OPTIONS OBS=10;
PROC IMPORT OUT= WORK.Test (KEEP=Customer_ID Name Address First-Ordered_Date)
  DATAFILE= "&dlmdir.\customer.csv"
  DBMS=CSV REPLACE;
```



SAS/ACCESS Interface to PC Files Server treats an Excel workbook as a database, and a range (subset of cells in a worksheet) as a table. A range name must be defined in the Excel file before SAS can use it. A worksheet is treated as a special range. A worksheet name appended with a \$ character is treated as a range.

For example, Sheet1 is a sheet name in an Excel file. SAS treats Sheet1\$ as a valid range name and uses it to refer to the whole worksheet. You need to use a SAS name literal when referring to the sheet name. For example, 'Sheet1\$'n. The first row of data in a range is usually treated as a column heading and used to create a SAS variables name.

Remember the following points as you work with Microsoft Excel files.

- Excel 4 files allow only one spreadsheet per file.
- Excel 4, Excel 5, and Excel 95 limits are 256 columns and 16,384 rows.
- Excel 97, 2000, 2002, 2003 limits are 256 columns and 65,536 rows.
- Excel 2007 limits are 16,384 columns and 1,048,576 rows.
- Excel 95 files are treated as the same format as Excel 5 files.
- Excel 2000, 2002, and 2003 files with an .xls file extension are treated as the same format as Excel 97 files.
- Excel 2007 and 2010 have three different file extensions:
  - .xlsb
  - .xlsm
  - .xlsx

### **Excel Data Types**

Microsoft Excel software has two data types: character and numeric.

- Character data can be labels or formula strings. Character data is generally considered text and can include character type dates and numbers. A cell can save up to 32,767 characters.
- Numeric data can be numbers, formulas, or error values. Numeric data can include numbers (0 through 9), formulas, or error values (such as #NULL!, #N/A, #VALUE!).

### **Excel Numeric Data and Time Values**

Numeric data can also include date and time values. 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.

In Microsoft Excel software, a date value is the integer portion of a number that can range from 01 January 1900 (saved as integer value: 1) to 31 December 9999 (saved as integer value: 2,958,465). 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 in a date, time, or date and time format.

In SAS software, SAS dates are valid back to AD 1582 and ahead to AD 9999. A date value is represented by the number of days between January 01, 1960, and that date. A time value is represented by the number of seconds between midnight and that time of

day. A datetime value is represented by the number of seconds between midnight January 01, 1960, and that datetime.

When you export a SAS time value to an Excel file, the value could be displayed as “1/0/1900” in the Excel file. Format the cell with a Time format to see the time value displayed correctly.

## Excel File Formats

Selecting “Save As,” you can also select from the following Excel formats:

- Excel Workbook creates an Excel .xlsx file.
- Excel Macro-Enabled Workbook creates an Excel .xlsm file.
- Excel Binary Workbook creates an Excel .xlsb file.
- Excel 97–2003 Workbook creates an Excel .xls file.

## SAS Import and Export Utilities Support for Excel Files

### Overview

SAS Import and Export Utilities provide three methods to access Microsoft Excel files.

#### LIBNAME statement

generates a LIBNAME statement and SQL commands for the PC files engine. Read data from or write data to an Excel file. This method supports Excel versions 5, 95, 97, 2000, 2002, 2003, and 2007.

See “[Overview: LIBNAME Engines](#)” on page 89 for additional information.

*Note:* DBMS=EXCEL and DBMS=EXCELS use this method to access data in Excel files.

#### ACCESS and DBLOAD procedures

generate ACCESS procedure code to read data from an Excel file. Also generate the DBLOAD procedure code to write data to an Excel file. This method supports only Excel versions 4 and 5/95. This is for SAS 6 compatibility support and is available only on Microsoft Windows.

*Note:* DBMS=EXCEL5 uses this method to access data in Excel files.

#### XLS and XLSX file formats

translates Excel .xls or .xlsx file formats to read data from or write data to an Excel file. This component supports Excel versions 5/95, 97, 2000, 2002, and 2003.

*Note:* This method does not support Excel .xlsb files, and it does not support such DBCS character sets as Chinese, Japanese, or Korean.

*Note:* DBMS=XLS and DBMS=XLSX use this method to access data in Excel files.

See “[SAS/ACCESS Descriptors for PC Files](#)” on page 227.

### Import and Export Microsoft Excel Files Using the LIBNAME Statement

The LIBNAME statement method for importing and exporting Microsoft Excel workbook files generates SAS LIBNAME statement code. The LIBNAME statement uses the Microsoft ACE engine or Microsoft Jet engine to access data in Microsoft Excel workbook files.

**Table 6.2** Statement Options to Import or Export Excel Data

Data Source	Syntax	Valid Value	Default Value	PROC IMPORT	PROC EXPORT
EXCEL	DBDSOPTS	Excel data set options		Yes	Yes
	DBSASLABEL	Compat   None	Compat	Yes	No
	GETNAMES	Yes   No	Yes	Yes	No
	MIXED	Yes   No	No	Yes	No
	NEWFILE	Yes   No	No	No	Yes
	RANGE	range name		Yes	No
	SCANTEXT	Yes   No	Yes	Yes	No
	SCANTIME	Yes   No	Yes	Yes	No
	SHEET	sheet name		Yes	Yes
	TEXTSIZE	1 to 32767	1024	Yes	No
	USEDATE	Yes   No	Yes	Yes	No

The next table lists Statement Options to Import or Export Excel Data files on Linux, Windows, and UNIX. This method requires that the PC Files Server is running on a Microsoft Windows operating system where the PC files reside.

**Table 6.3** Statement Options to Import or Export Excel Data Files on Windows, Linux, and UNIX

Data Source	Syntax	Valid Value	Default Value	PROC IMPORT	PROC EXPORT
EXCELCS	DBDSOPTS	'Excel data set options'		Yes	Yes
	DBSASLABEL	Yes   No	Yes	Yes	No
	PORT	1 to 65535	9621	Yes	Yes
	RANGE	'range name'		Yes	No
	SCANTEXT	Yes   No	Yes	Yes	No
	SCANTIME	Yes   No	Yes	Yes	No
	SERVER	'server name'		Yes	Yes
	SERVERPASS	'server password'		Yes	Yes
	SERVERUSER	'server User ID'		Yes	Yes
	SERVICE	'service name'		Yes	Yes
	SHEET	'sheet name'		Yes	Yes
	SSPI	Yes   No	No	Yes	Yes
	TEXTSIZE	1 to 32767	1024	Yes	No
	USEDATE	Yes   No	Yes	Yes	No
	VERSION	'5'   '95'   '97'   '2000'   '2002'   '2003'   '2007'	'97'	Yes	Yes

**DBDSOPTS= valid data set options for the SAS Excel LIBNAME engine**

enables data set options for the LIBNAME engine such as READBUFF, INSERTBUFF, DBTYPE, DROP, FIRSTOBS, and OBS. These options are for advanced users who are familiar with the PC Files LIBNAME engine.

**Notes:**

To improve performance for reading data, set the READBUFF= option to 25 or higher.

Enclose the options in single or double quotation marks as shown in the examples.

If the option string that you specify contains single quotations marks, use double quotation marks around it in your statement.

**See:** [“Overview of Data Set Options” on page 103](#) and [“Data Set Options for PCFILES on Linux, UNIX, and Microsoft Windows” on page 186](#) for additional information.

**Example:** DBDSOPTS Examples

```
DBDSOPTS= 'FIRSTOBS=10 READBUFF=25';
DBDSOPTS= "DBTYPE=(BilledTo='CHAR(8)')";
```

**DBSASLABEL= COMPAT | NONE | YES | NO**

specifies the data source for column names.

*COMPAT* specifies that the data source column headings are saved as the corresponding SAS label names.

Alias: *YES*

*NONE* specifies that the data source column headings are not saved as SAS label names. The SAS label names are then left as blanks.

Alias: *NO*

**Restrictions:**

Due to Microsoft Jet engine and Microsoft ACE engine limitations, no more than 64 characters of column names are written to SAS variable labels.

Due to Microsoft Jet engine and Microsoft ACE engine limitations, using MIXED=YES could result in improper text variable lengths.

**GETNAMES= YES | NO**

specifies whether the IMPORT procedure is to generate SAS variable names from the data values in the first row of the import file.

If data in the first row of the input file contains special characters for a SAS variable name (such as a blank), SAS converts the character to an underscore.

*YES* specifies that the IMPORT procedure generate SAS variable names from the data values in the first row of the imported Excel file.

*NO* specifies that the IMPORT procedure generate SAS variable names as F1, F2, F3, and so on.

**Default:** YES

**Restrictions:**

Valid only for Windows.

Valid only for the IMPORT procedure.

Supported only when DBMS=EXCEL.

When SAS reads the data value in the first row of the input file, SAS checks for invalid SAS name characters (such as a blank). Invalid characters are converted to an underscore. For example, the data value *Occupancy Code* becomes the SAS variable name **Occupancy\_Code**.

**MIXED= YES | NO**

assigns a SAS character type for the column and converts all numeric data values to character data values when mixed data types are found.

*YES* specifies that the connection is set to import mode and updates are not allowed.

*Note:* Due to a limitation in the Microsoft ACE engine and the Microsoft Jet Excel engine, using MIXED=YES could result in improper text variable lengths.

*NO* assigns numeric or character type for the column, depending on the majority of the type data that is found.

*Note:* Numeric data in a character column and character data in a numeric column are imported as missing values.

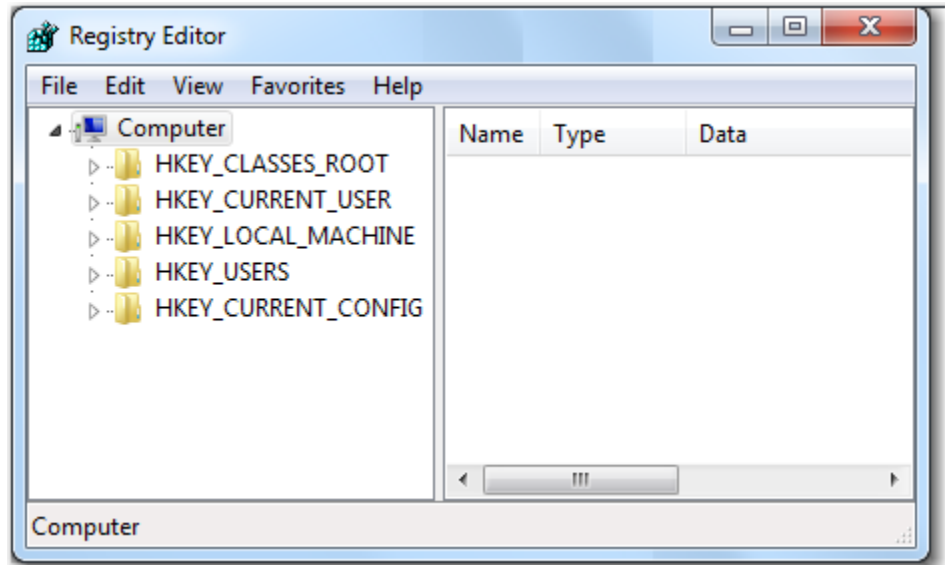
**Default:** NO

**Restriction:** Supported only when DBMS=EXCEL.

**Interaction:** The ‘TypeGuessRows’ entry in your registry settings can affect the behavior of the MIXED= option. The options are located in a key of the Microsoft Windows registry.

To change the value of ‘TypeGuessRows’ in your registry, follow these steps:

1. Access the Registry Editor by either selecting **Start** ⇒ **Run** from your desktop, entering *Regedt* and selecting **OK**, or by following the instructions for your system found in your system or application documentation.



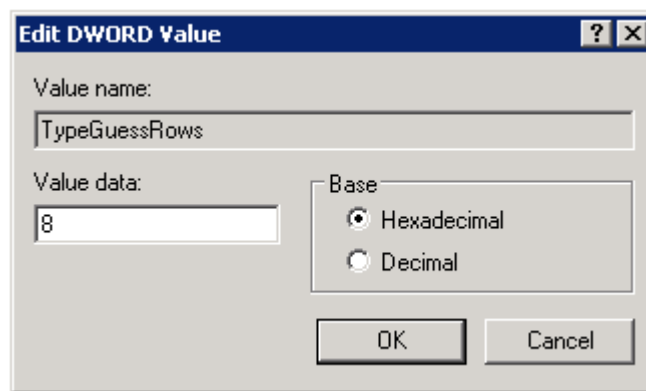
2. Open the appropriate key in the Registry Editor window, as indicated in the table below.

**Table 6.4** Registry Key for TypeGuessRow Based on Office Version or Engine

Environment	Office Version or Engine	Registry Key
Windows	Office 2007	HKEY_LOCAL_MACHINE ⇒ Software ⇒ Microsoft ⇒ Office ⇒ 12.0 ⇒ Access Connectivity Engine ⇒ Engines ⇒ Excel
Windows running 9.2 TS2M0 or later	Office 2010	HKEY_LOCAL_MACHINE ⇒ Software ⇒ Microsoft ⇒ Office ⇒ 14.0 ⇒ Access Connectivity Engine ⇒ Engines ⇒ Excel
Windows 7 or X64 system	Office 2007	HKEY_LOCAL_MACHINE ⇒ Software ⇒ Wow6432Node ⇒ Microsoft ⇒ Office ⇒ 12.0 ⇒ Access Connectivity Engine ⇒ Engines ⇒ Excel
Windows 7 or X64 system	Office 2010 32-bit	HKEY_LOCAL_MACHINE ⇒ Software ⇒ Wow6432Node ⇒ Microsoft ⇒ Office ⇒ 14.0 ⇒ Access Connectivity Engine ⇒ Engines ⇒ Excel
Windows 7 or X64 system	Office 2010 64-bit	HKEY_LOCAL_MACHINE ⇒ Software ⇒ Microsoft ⇒ Office ⇒ 14.0 ⇒ Access Connectivity Engine ⇒ Engines ⇒ Excel

Environment	Office Version or Engine	Registry Key
Windows	ACE Engine	HKEY_LOCAL_MACHINE ⇒ Software ⇒ Microsoft ⇒ Office ⇒ 12.0 ⇒ Access Connectivity Engine ⇒ Engines ⇒ Excel
Windows	Jet Engine (when using MENGINE=JET only)	HKEY_LOCAL_MACHINE ⇒ Software ⇒ Microsoft ⇒ Jet ⇒ 4.0 ⇒ Engines ⇒ Excel

- In the right pane, double-click **TypeGuessRows**.
- Change the **Value data** entry from 8 to 0 and select **OK**.



- Exit from the Registry Editor window.

The following table describes the registry settings for the MIXED= option.

**Table 6.5** Registry Settings for the MIXED= Option

TypeGuessRows	<p>An integer type with a default value of 8. You can use the number of rows in the worksheet range in scans to determine column types. If you set this type to 0, all rows up to 16384 in the range are checked. Microsoft states that the valid range of TypeGuessRows is 0–16. However, you could set as high as 16384, and it would still operate correctly.</p> <p>CAUTION: Changing the TypeGuessRows value causes a scan to fail if you set it higher than 16384. It also affects any software that uses the Microsoft Ace provider to access Excel file data, including accessing Excel data in a Microsoft Access database. The TypeGuessRows value is registered with and controlled by Microsoft. It is recommended that you set the value to 0.</p>
---------------	---

---

ImportMixedTypes	<p>A string type with a default value of Text. If a column contains more than one type of data while scanning TypeGuessRows rows, the column type is determined to be Text if the setting value is Text. If the setting value is Majority Type, the most common column type determines the column type.</p> <p>For the MIXED= YES option to work correctly, you should change TypeGuessRows to 0 in the Microsoft Windows registry so that all rows in the specified range are scanned. As a result, when you use MIXED= YES, the Microsoft Ace engine and the Microsoft Jet engine always assign character type for columns with data of mixed data types.</p> <p>The numeric data is converted to character data.</p>
------------------	---

---

**NEWFILE= YES | NO**

when exporting a SAS data set to an existing Excel file, specifies whether to delete the Excel file, and load the data to a sheet in a new Excel file.

*YES* specifies that the EXPORT procedure deletes the specified Excel file, if it exists. Loads the SAS data set to a sheet in a new Excel file.

*NO* specifies that the EXPORT procedure loads the SAS data set to a sheet and appends it to the existing Excel file. If the specified Excel file does not exist, an Excel file is created, and the SAS data set is loaded.

**Restriction:** Available only when DBMS=EXCEL.

**PORT= port-number**

specifies the number of the port that is listening on the PC Files Server. The valid value is between 1 and 65535. This port or service name displays on the PC Files Server display when the application is started in server mode.

**Alias:** PORT\_NUMBER

**Default:** 9621

**Restrictions:**

Available only for the client/server model.

The PORT= statement option and the SERVICE= statement option should not be used in the same procedure.

**RANGE= range-name | absolute-range**

subsets a spreadsheet by identifying the rectangular set of cells to import. The *range-name* is a user-defined spreadsheet name that represents a range of cells within the spreadsheet in the Excel file. The range-name is not case sensitive and does not allow any special character (except an underscore). The range-name is identified by the top left cell that begins the range and the bottom right cell that ends the range within the Excel 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. You must define range-name with a workbook scope so that the name is visible to SAS.

You can use the DATASETS procedure to list the data set names, that are mapped to the range-names. If the displayed range-name contains single quotes, keep the single quotes as part of the range-name to access the sheet.

An absolute range identifies the top left cell that begins the range and the bottom right cell that ends the range.

The following examples demonstrate the use of RANGE=.

- To retrieve data from the spreadsheet for two separate sheet names, 'My#Test' and 'CustomerOrders':

```
RANGE=" 'My#Test$' ";
RANGE=" ' CustomerOrders$' ";
```

- To represent cells within Column C, Row 2, and Column F, Row 12: **C2:F12** the colon separates the values for upper left (UL) and lower right (LR) of the range. If this statement is not specified, the IMPORT procedure reads the entire spreadsheet as a range.
- When data is imported from an Excel file, a sheet name that is appended with a \$ character is treated as a range name. The range name refers to the whole sheet.  
**RANGE=" 'summary\$a4:b20' ";**
- If the range name is available, it is recommended that you use RANGE = option without the SHEET option for the IMPORT procedure. To use the absolute range address, it is strongly recommended that you use the full range address with quotes. For example, specify '**sheet\_name\$A1:C7'n**

**Restriction:** Supported only for the IMPORT procedure.

### SCANTEXT= YES | NO

specifies whether to scan the column to determine the length of the text data for each data source column. This option applies only to character data type columns.

*YES* scans the length of text data for a data source column and uses the length of the longest string of data that it finds as the SAS variable width. If the maximum length that it finds is greater than the value in the TEXTSIZE = option, the smaller value in TEXTSIZE = option is applied as the SAS variable width.

*NO* does not scan the length of text data for a data source column. The column length returned from the Microsoft Ace or Microsoft Jet engine is used as the SAS variable width. If the returned column width is greater than what is specified in the TEXTSIZE option, the smaller value specified in the TEXTSIZE= option is applied as the SAS variable width.

*Note:* Specify SCANTEXT= NO when you need to update data in a Microsoft Excel workbook.

*Note:* So that the SCANTEXT= YES option works correctly, it is strongly recommended that you change TypeGuessRows to 0 in the Microsoft Windows registry. This ensures that all rows in the specified range are scanned. For registry values, see [Table 6.4 on page 39](#).

*Note:* To correct truncated text data for DBCS users (including Chinese, Japanese, and Korean), you can set the environment variable DBE\_DBCS to YES. This doubles the scanned text data length and therefore corrects the problem of truncated text data. To set the DBE\_DBCS environment variable, submit this statement: **Options set=DBE\_DBCS YES;**

*Note:* The Microsoft ACE or Microsoft Jet engine handles the SCANTEXT= option.

*Note:* These TypeGuessRows and ImportMixedTypes registry settings could affect the behavior of the SCANTEXT= option. Refer to the [Table 6.6 on page 43](#) table for additional information.

**Alias:** SCAN\_TEXT | SCANMEMO

**Table 6.6** Type Guessrows and Mixed Type Data

TypeGuessRows	An integer type with a default value of 8. The number of rows in the worksheet range is used to scan to determine column types. If set to 0, all rows in the range are checked. Microsoft states that the valid range of TypeGuessRows is 0 to 16. However, it could be set as high as 16384 and still operate correctly.
ImportMixedTypes	A string type with a default value of Text. If a column contains more than one data type when scanning of TypeGuessRows rows, the column type is determined to be Text if the setting value is Text. If the setting value is Majority Type, the most common column type determines the column type.

**SCANTIME=YES | NO**

specifies whether to scan the time data while importing data from a time column from the Microsoft Excel workbook.

*YES* scans the time column and assigns the *TIME.* format for a time column.

*NO* specifies to not scan the time column. The *DATETIME* format is assigned if *USEDATE= NO*. The *TIME.* format is assigned if *USEDATE= YES*.

**SERVER= PC-Files- Server-name**

specifies the name of the PC Files Server, where *PC-Files-Server-name* can be either the computer name or the associated IP address. You must bring up the listener on the PC Files Server before you can establish a connection. You can configure the service name, port number, maximum number of concurrent connections allowed, and data encryption on your PC Files Server.

**Alias:** SERVER\_NAME

**Restriction:** Available only for client/server model.

**Note:** You can omit this option if you are running SAS and the PC Files Server on the same machine. Omitting this option under this condition causes the PC Files Server to start automatically in the background.

**See:** “PC Files Server Administration” on page 162.

**SERVERPASS= server-user-password**

specifies the password for the User ID given. If the account has no password, omit this option. Always enclose the value in quotes, this preserves the case of the password.

**Alias:** SERVERPASSWORD | SERVERPW | SERVERPWD

**Notes:**

Passwords are generally case sensitive.

Use the *PASSWORD=* option for database passwords.

**Example:** LIBNAME using explicit user name and password, for PC Files Server:

```
LIBNAME DB PCFILES PATH='C:\myfile.mdb'
        SERVER=fileserv;
        SERVERUSER='mydomain\myusername';
        SERVERPASS='mypassword';
```

**TIP** If you are not on a domain, omit the domain name and the backslash.

**SERVERUSER= server-user-name**

specifies a domain and User ID that is valid for the PC running PC Files Server.

Always enclose the value in quotes. Otherwise, the backslash can be misinterpreted by the SAS parser.

**Alias:** SERVERUID

**Notes:**

If you are not on a domain, omit the domain name and the backslash.

Use the USER= option for database user IDs.

**Example:** Here is an example of the LIBNAME statement using explicit user name and password, used with the PC Files Server.

```
LIBNAME DB PCFILES PATH='C:\myfile.mdb'
        SERVER=fileserv;
        SERVERUSER='mydomain\myusername';
        SERVERPASS='mypassword';
```

**TIP** If you are not on a domain, omit the domain name and the backslash.

**SERVICE= *service-name***

specifies the service name that is defined on your service file for your client and server machines. This port number or service name is displayed on the PC Files Server control panel screen when it is started on the PC. The service name needs to be defined on your UNIX machine and your PC Files Server.

**Alias:** SERVICE\_NAME

**Restrictions:**

Available only for the client/server model.

The SERVICE= statement option and the PORT= statement option should not be used in the same procedure.

**SHEET= *sheet-name***

identifies a particular worksheet in an Excel workbook. Use the SHEET option only when you want to import an entire worksheet. The *sheet-name* can contain up to 31 characters. If the EXPORT procedure *sheet-name* contains special characters (such as space) SAS converts it to an underscore.

The following examples demonstrate how SAS converts non-compliant sheet names.

- The space is converted to an underscore. *Employee Information* becomes

```
Employee_Information
```

- If the sheet name contains single quotes, keep the single quotes as part of the sheet name in order to be able to access the sheet. **SHEET=" 'My#Test' ";**

**Restriction:** Avoid sheet names that look like cell references, which have <1–3 characters> plus 1 or more digits. For example, A1, IV65536, TRY123, XFD1048576.

**Notes:**

It is recommended that you use the RANGE= option without the SHEET= option in the IMPORT procedure.

If both the range name and the sheet name are missing, the IMPORT procedure reads the first worksheet that was physically saved in the Excel file.

**SSPI= YES | NO**

enables the server administrator to allow Integrated Windows Authentication. This is a mechanism for the Windows client and server to exchange credentials.

**Default:** NO

**Restriction:** Valid only on a 64-bit Windows PC.

**Note:** SSPI can also be enabled by specifying the –SSPI option on the SAS command line.

**TEXTSIZE= 1 to 32767**

specifies the SAS maximum variable length that is allowed while importing data from Microsoft Excel text columns. Any text data in Excel whose length exceeds this value is truncated when it is imported into SAS.

**Alias:** DBMAX\_TEXT

**USEDATE= YES | NO**

specifies whether to assign a DATE format while importing a date column from a Microsoft Excel workbook.

*YES* specifies the DATE9. format for the corresponding date column in the Microsoft Excel table.

*NO* does not specify the DATE9. format for the corresponding date column in the Microsoft Excel table.

**See:**

[SCANTIME= statement on page 43](#) to assign the appropriate TIME format.

[“Processing Date and Time Values between SAS and Microsoft Excel” on page 153](#)

**VERSION= file-version**

specifies the version of the file that you want to create. Valid values are 2007, 2003, 2002, 2000, 97, 95, and 5.

**Default:** 97 for .xls files

**Restrictions:**

If the file exists on the PC Files Server, this statement is ignored.

Available only for client/server model.

**Note:** Always enclose the version in single quotes.

**Example 1: Export a SAS Data Set to an Excel File**

Export a SAS data set called SDF.ORDERDS to an Excel 2007 .xlsb file with the Orders sheet name. In this case, SHEET= supports only an .xlsb file.

```
LIBNAME SDF "&sasdir";
PROC EXPORT DATA=SDF.ORDERDS
    FILE='c:\temp\demo.xlsb'
    DBMS=EXCEL REPLACE;
    SHEET='Orders';
RUN;
```

**Example 2: Import a Range from Excel 2007 File to a SAS Data Set**

This example imports a range named INVOICE from an Excel workbook file and performs these tasks:

- uses the first row of data as column names
- scans length for character type columns
- assigns DATE9. or TIME8. format to date and time columns respectively
- leaves SAS labels blank
- limits the size of text fields to be less than or equal to 512 characters

```
PROC IMPORT OUT= WORK.INVOICE
    FILE= "&demodir.demo.xlsb"
    DBMS= EXCEL REPLACE;
    RANGE= 'INVOICE';
    GETNAMES= YES;
```

```

SCANTEXT= YES;
USEDATE= NO;
SCANTIME= YES;
DBSASLABEL= NONE;
TEXTSIZE=512; /* default is 1024 */
RUN;

```

### **Example 3: Import a Range from an Excel File on a PC Files Server to a SAS Data Set**

The code in this example imports a range named Orders from an Excel file on a PC Files Server. It assigns the DATE9. format to date columns. The TIME8. format is assigned to time columns.

```

PROC IMPORT OUT=WORK.ORDERS
    DATAFILE="&pcfdir.demo.xls"
    DBMS=EXCELCS REPLACE;
RANGE='Orders';
SERVER="&server";
USEDATE=NO;
SCANTIME=YES;
RUN;

```

### **Example 4: Export a SAS Data Set to an Excel Workbook File**

This code exports a SAS data set to an Excel workbook file. The Excel file is on a UNIX or Windows 64-bit server with the “Customer” sheet on the PC that is running PC Files Server.

```

LIBNAME SDF "&sasdir";
PROC EXPORT DATA=SDF.CUSTOMER;
    FILE='c:\temp\demo.xls'
    DBMS=EXCELCS REPLACE;
SHEET='Customer';
SERVER="&server";
SERVICE=SASPCFILE;
RUN;

```

### **Example 5: The IMPORT Procedure Using SSPI — Microsoft 64-Bit Windows**

```

PROC IMPORT OUT=work.test
    DATAFILE='C:\myFile.xlsb'
    DBMS=EXCELCS
    REPLACE;
SERVER=FILESRV;
SSPI=YES;
RUN;

```

## **Importing and Exporting Microsoft Excel 4 and Excel 5 Files**

The IMPORT and EXPORT methods use the ACCESS and DBLOAD procedures to access data in Microsoft Excel files. Excel 2007, .xlsx, .xlsb, and .xlsm files are not supported. The ACCESS and DBLOAD procedures are available only on Microsoft Windows.

*Note:* Because the ACCESS and DBLOAD procedures are compatible only with SAS 6 procedures, they ignore SAS system options such as the VALIDVARNAME=

option. The ACCESS procedure and the DBLOAD procedure have other SAS 6 limitations such as a maximum of 8-byte SAS variable names and a maximum of 200-character value.

The following table lists the statements that are available to import data from or export data to an Excel file using the EXPORT and IMPORT procedures.

**Table 6.7** Statements for Importing and Exporting Excel 4 and Excel 5 Files

Data Source	Syntax	Valid Value	Default Value	PROC IMPORT	PROC EXPORT
EXCEL5	GETNAMES	Yes   No	Yes	Yes	No
	RANGE	Range Name or Absolute Range Value, such as 'A1..C4'		Yes	No
	SHEET	Sheet name		Yes	No
EXCEL4	GETNAMES	Yes   No	Yes	Yes	No
	RANGE	Range Name or Absolute Range Value, such as 'A1..C4'		Yes	No

**GETNAMES= YES | NO**

specifies whether the IMPORT procedure is to generate SAS variable names from the first record of the Microsoft Excel import file.

If data in the first record of the input file contains special characters for a SAS variable name (such as a blank), SAS converts the character to an underscore. For example, the data value MY ID becomes the SAS variable name **MY\_ID**.

**YES** specifies that the IMPORT procedure generate SAS variable names from the data values in the first record of the imported Excel file.

**NO** specifies that the IMPORT procedure generate SAS variable names as F1, F2, F3, and so on.

**Default:** YES

**Restrictions:**

Valid only for Windows.

Valid only for the IMPORT procedure.

Supported only when DBMS=EXCEL5.

When SAS reads the data value in the first row of the input file, SAS checks for invalid SAS name characters (such as a blank). Invalid characters are converted to an underscore. For example, the data value *Occupancy Code* becomes the SAS variable name **Occupancy\_Code**.

**RANGE= range-name | absolute-range**

subsets a specified section of an Excel file worksheet. The *range-name* is the name that is assigned to a range address within the worksheet. Range names 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 Excel 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.

**SHEET= *sheet-name***

identifies one worksheet from a group of worksheets while you are reading from an Excel file. The sheet name can be up to 31 characters. The SHEET statement is optional.

**Example 1: Import a SAS Data Set to an Excel 5 File**

This example imports a SAS data set, INVOICE, from an Excel 5 workbook file, INVOICE.

```
PROC IMPORT OUT=WORK.INVOICE
           FILE="&xls5dir.invoice.xls"
           DBMS=EXCEL5 REPLACE;
  GETNAMES=yes;
RUN;
```

**Example 2: Export a SAS Data Set to an Excel 5 File**

This example exports a SAS data set, ORDERS, to an Excel 5 workbook file.

```
LIBNAME SDF "$sasdir";
PROC EXPORT DATA=SDF.ORDERS
  OUTFILE='c:\temp\orders.xls'
```

**Import and Export Microsoft Excel Files Using XLS and XLSX File Formats**

This Import/Export component uses the translation engine method to read and write XLS file formats directly. This component supports Excel versions 5/95, 97, 2000, 2002, and 2003. However, it does not support Excel 2007 .xlsx, .xlsb, or .xlsm files. It is available on Linux, UNIX, and Microsoft Windows operating platforms.

These tables list the statements that are available to import data from or export data to an Excel file.

**Table 6.8** Available Statements for Importing and Exporting Excel Files Using the Translation Engine

Data Source	Syntax	Valid Value	Default Value	PROC IMPORT	PROC EXPORT
XLS	ENDCOL	Last column for data	Last column that contains data	Yes	No
	ENDNAMEROW	Last row for variable names	Same as NAMEROW	Yes	No
	ENDROW	Last row for data	Last row that contains data	Yes	No
	GETNAMES	Yes   No	Yes	Yes	No
	NAMEROW	First row for variable names	First row that contains variable names	Yes	No
	NEWFILE	Yes   No	No	No	Yes
	PUTNAMES	Yes   No	Yes	No	Yes
	RANGE	NAME   SHEET\$UL-LR	First row	Yes	No
	SHEET	Sheet name	First sheet	Yes	Yes
	STARTCOL	First column for data	Last column that contains data	Yes	No
	STARTROW	First row for data	First row that contains data	Yes	No
XLSX	GETNAMES	Yes   No	Yes	Yes	No
	RANGE	NAME   SHEET\$UL-LR	First row	Yes	No
	SHEET	Sheet name	First sheet	Yes	No

**ENDCOL= last-column-for-data**

specifies the last column for data

**Default:** The last row that contains data.**Restriction:** Available only for DBMS=XLS for backward compatibility.**ENDNAMEROW= name-row**

specifies the last row for variable names.

**Default:** The same as NAMEROW**Restriction:** Available only for DBMS=XLS for backward compatibility.**ENDROW= 1 to 65535**

specifies the last row for data.

**Default:** The last row that contains data.**Restriction:** Available only for DBMS=XLS for backward compatibility.

**Note:** Valid Value Range: 1 to 65535

**GETNAMES= YES | NO**

determines whether to generate SAS variable names from the data values in the first record of the imported file. If data in the first record is read and it contains characters that are not valid in a SAS name, SAS converts the character to an underscore.

For example, the data value *Occupancy Code* becomes the SAS variable name **Occupancy\_Code**.

*YES* specifies that PROC IMPORT is to generate SAS variable names from the data values in the first record of the imported Excel file.

*NO* specifies that PROC IMPORT is to generate SAS variable names as VAR1, VAR2, VAR3, and so on.

**Restrictions:**

Available only for DBMS=XLS for backward compatibility.

PROC IMPORT only

**NAMEROW= name-row**

specifies the first row for variable names.

**Default:** The first row that contains variable names.

**Restriction:** Available only for DBMS=XLS for backward compatibility.

**NEWFILE= YES | NO**

when exporting a SAS data set to an existing Excel file, specifies whether to delete the Excel file, and load the data to a sheet in a new Excel file.

*YES* specifies that the EXPORT procedure deletes the specified Excel file, if it exists. Loads the SAS data set to a sheet in a new Excel file.

*NO* specifies that the EXPORT procedure loads the SAS data set to a sheet and appends it to the existing Excel file. If the specified Excel file does not exist, an Excel file is created, and the SAS data set is loaded.

**Restriction:** Available only for DBMS=XLS for backward compatibility.

**PUTNAMES= YES | NO**

determines whether to write SAS variable names to the first record of the exported data file. If you specify the LABEL option, SAS variable labels are written instead of variable names.

*YES* specifies that PROC EXPORT is to write SAS variable names to the first record and write the first observation data to the second record of the exported data file.

*NO* specifies that PROC EXPORT is to write the first observation data to the exported data file.

**Restrictions:**

Available only for DBMS=XLS for backward compatibility.

PROC EXPORT only.

**RANGE= 'range-name' | 'absolute-range';**

subsets a spreadsheet by identifying the rectangular set of cells to import from the specified spreadsheet. The range name is a name that represents a range of cells within the spreadsheet in the Excel file. Absolute range identifies the top left cell that begins the range and the bottom right cell that ends the range. For example, 'C2..F12' represents cells within column C, row 2, and column F, row 12. You need to specify the target sheet name with SHEET= if you use absolute range.

By default, the first row is viewed as containing variable names. If GETNAMES=N, the column is set to the 1- or 2-letter ID, and all rows are read as data.

If you do not specify RANGE=, PROC IMPORT reads the entire worksheet as a range.

Use RANGE= instead of STARTCOL=, STARTROW=, ENDCOL=, ENDROW=, or any combination of these because RANGE= already contains all of these values.

**Restriction:** This statement is not valid for PROC EXPORT.

**SHEET= 'sheet-name'**

identifies a particular worksheet in an Excel workbook. Specify sheet-name to name the sheet name as output. If the sheet already exists, it is replaced. You can also use it to append a new sheet to an existing worksheet.

If you do not specify this statement, PROC IMPORT reads the first worksheet physically saved in the Excel file. To be certain that PROC IMPORT reads the worksheet that you want, specify SHEET= to identify the worksheet.

Even if you can specify the SHEET= statement in PROC IMPORT for EXCEL4, the value is ignored. Excel version 5/95 allows multiple sheets in a file, but this export component supports only exporting a single sheet per file.

For PROC EXPORT, if you specify the SHEET= statement, the name defines the sheet name and range name in the exported Excel file. The underscore character replaces the special character for both the range and sheet names. If you do not specify the SHEET= statement, the SAS data set name defines the sheet name and range name in the exported Excel file.

**STARTCOL= start-column**

specifies the first column for data.

**Default:** The first column that contains data.

**STARTROW= start-row**

specifies the first row for data.

**Default:** The first row that contains data.

**Restriction:** Available only for DBMS=XLS for backward compatibility.

### Example 1: Export SAS Data Sets to Excel Workbook Files

This example exports the SAS data sets, SDF.INVOICE and SDF.ORDERS, to Excel workbook files with Invoice and Orders as sheet names.

```
LIBNAME SDF V9 "&sasdir"
PROC EXPORT DATA=SDF.INVOICE
    FILE="&tmpdir.text.xlsx"
    DBMS=XLS REPLACE;
    SHEET='Invoice';
RUN;

PROC EXPORT DATA=SDF.ORDERS
    FILE="&tmpdir.text.xls"
    DBMS=XLS REPLACE;
    SHEET='Orders';
RUN;
```

### Example 2: Import Data Using a Range Name

This example imports SAS data from a demo XLS file using a range name.

```

PROC IMPORT OUT=WORK.INVOICE
    FILE="&demodir.demo.xls"
    DBMS=XLS REPLACE;
    RANGE=' INVOICE' ;
    GETNAMES=YES;
RUN;

```

### Example 3: Import Data Using an Absolute Range Address

This example imports SAS data from a demo XLS file using an absolute range address.

```

PROC IMPORT OUT=WORK.INVOICE
    FILE="&demodir.demo.xls"
    DBMS=XLS REPLACE;
    RANGE="Invoice$B4:D10";
    GETNAMES=NO;
RUN;

```

---

## Microsoft Access Database Files

### Microsoft Access File Essentials

SAS/ACCESS Interface to PC Files works with Microsoft Access database 97, 2000, 2002, 2003, and 2007 files.

Microsoft Access is a desktop relational database system that uses the Microsoft Ace engine or the Microsoft Jet engine to store and retrieve data.

A database is a collection of information that is related to a particular subject or purpose, such as tracking customer orders or maintaining a music collection. If the size is greater than, or equal to MS Access database 2007, all objects in an MS Access database are stored in the Jet .mdb or Ace .accdb format.

The following table lists the maximum size limits for the methods of .mdb and .accdb files.

**Table 6.9** Microsoft Access Database (.mdb and .accdb) Maximum Size Limits per Method

File size	2GB /32,768 objects
Number of fields per table	255 units
	255 characters
Memo field size	65535 characters
Table name size	64 characters
Field name size	64 characters

---

Record size	For .mdb: 2000 characters (excluding memo and OLE object fields)
	For .accdb: 4000 characters (excluding memo and OLE object fields)

---

While importing data from a table, SAS converts special characters in a table name to underscores in the corresponding data set name. If a field contains special characters, SAS converts special to underscores in the corresponding variable name.

## Microsoft Access Data Types

Summary of field data types that are available in Microsoft Access; their uses; and their storage sizes.

### ATTACHMENTS

specify attachment to images, spreadsheet files, documents, charts, and other types of supported files to rows in your database.

### AUTONUMBER

use for unique sequential (incrementing by 1) or random numbers that are automatically inserted when a row is added. Stores 4 bytes; stores 16 bytes for Replication ID (GUID).

### CURRENCY

use for currency values and to prevent rounding during calculations. Stores as 8-byte numbers with precision to four decimal places.

### DATE/TIME

use for dates and times. Stores as 8-byte numbers.

### HYPER LINK

use for hyperlinks. A hyperlink can be a UNC path (link) or a URL (link). Stores up to 64,000 characters.

### MEMO

use for lengthy text and numbers, such as notes or descriptions. Stores up to 32767 characters. ACCESS 2007 stores up to 65535 characters.

### NUMBER

use for data to be included in mathematical calculations, except for calculations involving money (use Currency type). Stores 1, 2, 4, or 8 bytes; stores 16- bytes for Replication ID (GUID). The `FIELD SIZE` property defines the specific Number type.

### OLE OBJECT

use for OLE objects (such as Microsoft Word documents, Microsoft Excel spreadsheets, pictures, sounds, or other binary data) created in other programs using the OLE protocol (link). Stores up to 1 gigabyte (limited by disk space).

### TEXT

use for text or combinations of text and numbers, such as addresses, numbers that do not require calculations, telephone numbers, part numbers, or postal codes. Stores up to 255 characters. The `FIELD SIZE=` option, controls the maximum number of characters that can be entered.

### YES | NO

use for data that can be only one of two possible values, such as YES | NO, TRUE | FALSE, ON | OFF. Stores 1 bit.

For YES value use -1.

For NO value use 0.

*Note:* NULL values are not allowed.

### **The Conversion of Date and Time Values between SAS Data Sets and Microsoft Access Database**

In Microsoft Access database software, the following date and time rules apply:

- Date values are valid back to 30 December 1899 and are saved as the integer value: 0.
- Date values are valid ahead to 31 December 9999, and are saved as the integer value: 2,958,465.
- Years 4000 and 8000 are considered leap years.
- Time value is the decimal portion of a number that represents time as a proportion of a day.
- Numbers can have both a date and a time portion, the format is in date/time format.
- Number display formats support date, time, or date/time formats.

In SAS software, the following date and time rules apply:

- Dates are valid back to A.D.1582.
- Date values before year 1582 are represented as missing values.
- Dates are valid forward to A.D. 9,999.
- Date values are represented by the number of days between January 01, 1960, and that date.
- Years 4000 and 8000 are not considered leap years.
- Time values are represented by the number of seconds between midnight and that time of day.
- Date and time values are represented by the number of seconds between midnight January 01, 1960, and that date and time.
- Time values can be imported as a date value 30 Dec 1899. Ensure that you assign the correct format.

### **IMPORT and EXPORT Procedure Statements for Access Files**

#### **Overview**

Before you use the IMPORT and the EXPORT procedures for Access files, it is helpful to be familiar with Access file formats. For an existing .mdb file, you can specify DBMS=ACCESS on Windows platforms, and SAS identifies the version.

- Access 2000, 2002, and 2003 share the same .mdb file formats. ACCESS2000, ACCESS2002, and ACCESS2003 are treated as aliases for Access in SAS.
- Access 2007 supports .acdb file formats.

See [“Example 1: Import a SAS Data Set from an Access 2007 Database Table”](#) on page 61 and [“Example 2: Export a SAS Data Set to Create an Access Database File”](#) on page 61 for additional information.

**Table 6.10** IMPORT and EXPORT Procedure Statements for Access Files on Windows

Data Source	Syntax	Valid Value	Default Value	PROC IMPORT	PROC EXPORT
ACCESS ACCESS97	DATABASE	Complete path and filename for the Microsoft Access database file.		Yes	Yes
	DBDSOPTS	Any valid data set options for Microsoft Access database engine.		Yes	Yes
	DBPASSWORD	Database password.		Yes	Yes
	DBSASLABEL	Yes   No	Yes	Yes	No
	MEMOSIZE	1 to 32767	1024	Yes	No
	PASSWORD	User password.		Yes	Yes
	SCANMEMO	Yes   No	Yes	Yes	No
	SCANTIME	Yes   No	Yes	Yes	No
	USEDATE	Yes   No	No	Yes	No
	USER	User ID.		Yes	Yes
DBSYSFILE	Complete path and filename for the Workgroup Administration file.		Yes	Yes	

On Linux, UNIX, and Microsoft Windows operating platforms, you can use the client/server model to access data in .mdb files and .accdb files. For more information, see [“PC Files Server Administration”](#) on page 162.

For an existing .mdb you can specify DBMS=ACCESSCS when using the client/server model and SAS identifies the version of Access for you. You can specify VERSION = '97' when using the client/server model only when you want to export and create a new .mdb file with Access version 97 formats.

*Note:* If the client/server model is used, the SAS client cannot access an Access database file with both database password protection and user level security protection. In this case, you must choose only one security protection to be able to access your Access database file from a SAS client.

The following table lists the statements to import or export data to or from an Access database file using the client/server model. The statements are valid on Linux, UNIX, and Microsoft Windows operating platforms.

**Table 6.11** IMPORT and EXPORT Procedure Statements for Access Files When Using PC Files Server

Data Source	Syntax	Valid Value	Default Value	PROC IMPORT	PROC EXPORT
ACCESSCS	DATABASE	Complete path and filename for the Microsoft Access database file.		Yes	Yes
	DBDSOPTS	Any valid data set options for Microsoft Access database engine.		Yes	Yes
	DBPASSWORD	Database password.		Yes	Yes
	DBSASLABEL	Yes   No	Yes	Yes	No
	DBSYSFILE	Complete path and filename for the Workgroup Administration file.		Yes	Yes
	MEMOSIZE	1 to 32767	1024	Yes	No
	PASSWORD	User password.		Yes	Yes
	PORT	1 to 65535	9621	Yes	Yes
	SCANMEMO	Yes   No	Yes	Yes	No
	SCANTIME	Yes   No	Yes	Yes	No
	SERVER	Server name.		Yes	Yes
	SERVERPASS	Server password.		Yes	Yes
	SERVERUSER	Server user ID.		Yes	Yes
	SERVICE	Service name.		Yes	Yes
	SSPI	Yes   No		Yes	Yes
	USEDATE	Yes   No	No	Yes	No
	USER	User ID		Yes	Yes
VERSION	97   2000   2002   2003   2007   2010	2000	Yes	Yes	

**DATABASE= *database***

specifies the complete path and filename of the Access database file that contains the specified DBMS table. If the database name does not contain lowercase characters, special characters, or national characters (\$, #, or @), you can omit the quotation marks.

**Note:** SAS does not generate a default value, but you can configure one in the DBMS client software.

**DBDSOPTS= *data set options that are valid for the Access database LIBNAME engine***

Enables you to take advantage of useful data set options for the LIBNAME engine such as READBUFF=, INSERTBUFF=, DBTYPE=, DROP=, FIRSTOBS=, and OBS=. This option is for advanced users who are familiar with the PC files LIBNAME engine.

**Requirements:**

You must surround the options in single quotation marks.

```
/* Example of correct use */
DBDSOPTS='FIRSTOBS=10 READBUFF=25';
```

If the option string that you are specifying contains single quotation marks, you must use double quotation marks around it in your statement.

```
/*Example of correct use */
DBDSOPTS="DBTYPE=(BilledTo=' CHAR (8) ' )";
```

**Notes:**

For users who use the client/server model to access data in an Access database, the default value for READBUFF= is 1. To improve performance for reading data, you should set the READBUFF= option to 25 or higher.

For users who use the client/server model to access data in an Access database, the default value for INSERTBUFF= is 1. To improve performance for reading data, you should set the INSERTBUFF= option to 25 or higher.

**See:** [“Data Set Options for PCFILES on Linux, UNIX, and Microsoft Windows”](#) on page 186

**DBPASSWORD= *database-file-password***

enables you to access a file if database-level security is set in the .mdb file. This option enables you to access .mdb and .accdb files with passwords, but it does not allow you to create .mdb and .accdb files with passwords included. A database password is case sensitive. You can define a database password instead of user-level security.

**Alias:** DBPWD | DBPW

**DBSASLABEL= *COMPAT | NONE | YES | NO***

specifies the data source for column names.

*COMPAT* specifies that the data source column headings are saved as the corresponding SAS label names.

*NONE* specifies that the data source column headings are not saved as SAS label names. The SAS label names are then left as blanks.

**Alias:**

YES (for COMPAT)

NO (for NONE)

**Restrictions:**

Due to a Microsoft Jet engine limitation, no more than 64 characters of column names are written to SAS variable labels.

Due to a limitation in the Microsoft Ace engine and the Microsoft Jet Excel engine, using MIXED=YES could result in improper text variable lengths.

**DBSYSFILE= *complete path and filename for the Workgroup Administration file*** specifies the location of the Workgroup Administration file. You might have defined this file, which contains information about the users in a Workgroup, for your Microsoft Access database.

**Alias:** WGDB

**Note:** When you install Microsoft Access, the Setup program automatically creates a Microsoft Access Workgroup information file that is identified by the name and organization information that you specify. Because this information is often easy to determine, it is possible for unauthorized users to create another version of this Workgroup Information file and assume the irrevocable permissions of an administrator account (a member of the Admins group) in the Workgroup defined by that Workgroup Information file. To prevent this, create a new Workgroup Information file and specify a Workgroup ID (WID). Only someone who knows the WID can create a copy of the Workgroup Information file. Any user and group accounts or passwords that you create are saved in the new Workgroup Information file.

**MEMOSIZE= 1 to 32767**

Specifies the maximum variable length in SAS that is allowed while importing data from memo columns of an Access database table. Any memo data in an Access database tables whose length exceeds 32767 is truncated when it is imported into SAS.

**Alias:** DBMAX\_TEXT

**Restriction:** If the maximum length that SCANMEMO = option is greater than the value of the MEMOSIZE= option, the smaller value in the MEMOSIZE = option is applied as the SAS variable width.

**PORT= *port-number***

specifies the number of the port that is listening on the PC Files Server. The valid value is between 1 and 65535. This port or service name displays on the PC Files Server display when the application is started in server mode.

**Alias:** PORT\_NUMBER

**Default:** 9621

**Restrictions:**

Available only for the client/server model.

The PORT= statement option and the SERVICE= statement option should not be used in the same procedure.

**SCANMEMO= YES | NO**

specifies whether to scan the memo data to determine the column length for each memo-type source column.

*YES* scans the length of memo data for a data source column. Uses the length of the longest memo text of data that it finds, as the SAS variable width.

*NO* does not scan the length of memo data for a data source column. The column length returned from the Microsoft Ace engine or the Microsoft Jet engine is used as the SAS variable length.

**Restrictions:**

When SCANMEMO=YES, if the maximum length that SCANMEMO = option is greater than the value of the MEMOSIZE= option, the smaller value in the MEMOSIZE = option is applied as the SAS variable width.

SCANMEMO = does not apply to text type columns. This option applies only to memo data type columns.

**SCANTIME= YES | NO**

specifies whether to scan the date/time data while importing data from a date/time column from an Access database.

*YES* scans the date/time column and assigns the TIME. format for a date/time column only if time values are found in the column.

*NO* specifies not to scan the date/time column.

**Interactions:**

The DATE9. format is assigned for a date/time column if USEDATE= YES.

The DATETIME. format is assigned for a date/time column if USEDATE= NO.

**SERVER= PC-Files-Server-name**

specifies the name of the PC Files Server, where *PC-Files-Server-name* can be either the computer name or the associated IP address. You must bring up the listener on the PC Files Server before you can establish a connection to it. You can also configure these items:

- the service name
- the port number
- the maximum number of concurrent connections
- specifications to indicate whether data encryption will be used

**Alias:** SERVER\_NAME

**Restriction:** Available only for the client/server model.

**Note:** You can omit this option if you are running SAS and the PC Files Server on the same machine. Omitting this option under this condition causes the PC Files Server to start automatically in the background.

**SERVERPASS= server-user-password**

specifies the password for the User ID given. If the account has no password, omit this option. Always enclose the value in quotes. This preserves the case of the password.

**Alias:** SERVERPASSWORD | SERVERPW | SERVERPWD

**Notes:**

Passwords are generally case sensitive.

Use the PASSWORD= option for database passwords.

**Example:** LIBNAME example using explicit user name and password for PC Files Server:

```
LIBNAME DB PCFILES PATH='C:\myfile.mdb'
         SERVER=fileserv;
         SERVERUSER='mydomain\myusername';
         SERVERPASS='mypassword';
RUN;
```

**SERVERUSER= server-user-name**

specifies a domain and user ID that is valid for the PC running PC Files Server. Always enclose the value in quotes. Otherwise, the backslash can be misinterpreted by the SAS parser.

**Alias:** SERVERUID

**Notes:**

If you are not on a domain, omit the domain name and the backslash.

Use the USER= option for database User IDs.

**Example:** LIBNAME using explicit user name and password, for PC Files Server:

```
LIBNAME DB PCFILES PATH='C:\myfile.mdb'
        SERVER=fileserv;
        SERVERUSER='mydomain\myusername';
        SERVERPASS='mypassword';

RUN;
```

**SERVICE= *service-name***

specifies the service name that is defined on your service file for your client and server machines. This port number or service name is displayed on the PC Files Server control panel screen when it is started on the PC in server mode.

**Alias:** SERVICE\_NAME

**Restrictions:**

Available only for client/server model.

Do not use this statement and the PORT= statement option in the same procedure.

**Note:** This service name needs to be defined on both your UNIX machine and your PC Files Server.

**SSPI= *YES* | *NO***

enables PC Files Server to allow Integrated Windows Authentication. This is a mechanism for Windows client and server to exchange credentials.

**Default:** NO

**Restriction:** Valid only on Windows 64-Bit.

**Note:** SSPI can also be enabled by specifying the -SSPI option on the SAS command line.

**Example:** LIBNAME using SSPI:

```
LIBNAME DB PCFILES PATH='C:\myfile.mdb'
        SERVER=localhost;
        SSPI = 'yes';

RUN;
```

**USEDATE= *YES* | *NO***

specifies whether to assign a DATE. or a DATETIME. format while importing a date/time column from a Microsoft ACCESS workbook.

*YES* assigns the DATE. format for the corresponding date/time column in the Microsoft ACCESS table. See the SCANTIME = option to assign the appropriate TIME format.

*NO* assigns the DATETIME. format for the corresponding date/time column in the Microsoft ACCESS table. assigns the DATETIME. format for the corresponding date/time column in the Microsoft ACCESS table.

**See:**

For processing of date and time values between SAS and Microsoft Access, see [“Processing Date and Time Values between SAS and Microsoft Access” on page 157.](#)

The [SCANTIME= statement option on page 43](#) in order to assign the appropriate TIME format.

**VERSION= *file-version***

specifies the version of the file that you want to create. Valid values are 2007, 2003, 2002, 2000, and 97. The default value depends on the extension of the file. Always surround the version value with single quotation marks.

**Restriction:** Available only for client/server model.

**Interaction:** If the file exists on the PC Files Server, then the statement is ignored.

### **Example 1: Import a SAS Data Set from an Access 2007 Database Table**

This code imports a data set named CUSTOMER from the Customers table in a Microsoft Access database: demo.accdb. The Microsoft Access table was saved in version 2007 format.

```
PROC IMPORT OUT=WORK.CUSTOMER
            DATATABLE='Customers'
            DBMS=ACCESS REPLACE;
DATABASE="&demodir.demo.accdb";
USEDATE=YES;
SCANTIME=NO;
DBSASLABEL=NONE;
RUN;
```

### **Example 2: Export a SAS Data Set to Create an Access Database File**

This code exports a SAS data set named EMPLOYEE and creates a new Microsoft Access database file named test2000.mdb. Note that test2000.mdb does not exist before the EXPORT procedure is submitted. SAS loads and names the table Employees. It then creates and saves it in the new file, test2000.mdb.

```
X 'DEL c:\temp\test2000.mdb';
PROC EXPORT DATA=SDF.EMPLOYEE
            OUTTABLE='Employees'
            DBMS=ACCESS REPLACE;
DATABASE='c:\temp\test2000.mdb';
RUN;
```

### **Example 3: Import a Data Set from an Access Database File Using a Read Buffer**

This code imports a data set named INVOICE from the Invoice table in a Microsoft Access database named demo.mdb. The read buffer is set to 10 rows.

```
PROC IMPORT OUT=SDF.INVOICE
            TABLE='Invoice'
            DBMS=ACCESSCS REPLACE;
DATABASE="&pcfdir.demo.mdb";
SERVER="&server";
DBDSOPTS='READBUFF=10';
RUN;
```

### **Example 4: Export a SAS Data Set to an Access Database File on a PC Files Server**

This code exports a SAS data set named ORDERS to a new Microsoft Access database file named testpcfs.mdb, located on the PC Files Server. The column, SPECINST, is dropped. The write buffer is set to 25 rows.

The code performs these tasks:

```
LIBNAME SDF "&sasdir";
PROC EXPORT DATA=SDF.ORDERS (DROP=SPECINST)
            OUTTABLE='Orders'
            DBMS=ACCESSCS REPLACE LABEL;
DATABASE='c:\temp\testpcfs.mdb';
SERVER="&server";
```

```

VERSION='97';
DBDSOPTS='INSERTBUFF=25';
RUN;

```

## Lotus 1-2-3 WK $n$ Files

### WK $n$ Files Essentials

This section introduces Lotus 1-2-3 WK $n$  files. It focuses on the terms and concepts that help you use SAS/ACCESS Interface to PC Files.

SAS/ACCESS Interface to 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. They are referred to collectively in this document as WK $n$  files, where  $n$  represents releases 1, 3, or 4. SAS/ACCESS Interface to PC Files does not support the .123 format for files from Lotus SmartSuite 97 software.

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 spreadsheet 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. This display illustrates a portion of a standard 1-2-3 spreadsheet.

**Display 6.1** Columns and Rows of Data in a WK $n$  File

The screenshot shows a Lotus 1-2-3 spreadsheet window titled "Lotus 1-2-3 Release 5 - [CUSTDATA.WK4]". The active cell is B2..E8, containing the text "^CUSTOMER". The spreadsheet grid shows columns A through G and rows 1 through 20. The data is organized as follows:

	A	B	C	D	E	F	G
1							
2		CUSTOMER	CITY	STATE	COUNTRY		
3		14324724	San Jose	CA	USA		
4		14569877	Memphis	TN	USA		
5		14898029	Rockville	MD	USA		
6		26422096	LaRochelle		France		
7		38763919	Buenos Aires		Argentina		
8		46783280	Singapore		Singapore		
9							
10							
11							
12							
13							
14							
15							
16							
17							
18							
19							
20							

The status bar at the bottom shows "Automatic", "Arial", "12", and "Ru".

Column letters for each column appear above the spreadsheet. Columns are lettered A–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 spreadsheet. Rows are numbered 1 to 8,192. For WK1 files, only one spreadsheet (spreadsheet A) is allowed per file. For WK3 and WK4 files, up to 256 spreadsheets (spreadsheets A-IV) are allowed. However, the SAS/ACCESS interface to WK $n$  files uses only one spreadsheet and defaults to spreadsheet A.

A range is a subset of cells in a spreadsheet. A range is identified by its address. The address begins with the name of the top left cell. The range ends with the name of the bottom right cell. The names are 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 contain no spaces. For example, if the range B3..D6 is named GRADE\_TABLE, the formula @AVG(GRADE\_TABLE) has the same value as @AVG(B3..D6).

For more information about ranges and their naming conventions, see the Lotus 1-2-3 software documentation.

## 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 quotation mark 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 these symbols: +, \$, @, -, or #.

Numeric data can also 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.

## Supported Import and Export Methods and Statements for WK $n$ Files

The IMPORT | EXPORT method for WK $n$  files uses ACCESS and DBLOAD procedures behind the scenes to access data in WK $n$  files. This method is available only in Microsoft Windows.

Because the ACCESS and DBLOAD procedures are compatible only with SAS 6 procedures, SAS system options such as the VALIDVARNAME= option are ignored. This method has other SAS 6 limitations such as:

- not being case sensitive
- a maximum length of 8 bytes of SAS variable name

- maximum length of 200 characters of data values

The following table lists the statements that are available to import and export data from Lotus 1-2-3 files using the IMPORT and EXPORT procedures on Microsoft Windows.

**Table 6.12** Import or Export Data from Lotus 1–2–3 Files

Data Source	Syntax	Valid Value	Default Value	PROC IMPORT	PROC EXPORT
WK4	GETNAMES	Yes   No	Yes	Yes	No
WK3	RANGE	Range Name   Absolute Range Value		Yes	No
	SHEET	Sheet name		Yes	Yes
WK1	GETNAMES	Yes   No	Yes	Yes	No
	RANGE	Range Name   Absolute Range Value		Yes	No

**GETNAMES= YES | NO**

specifies whether the IMPORT procedure is to generate SAS variable names from the data values in the first row of the import file.

If data in the first record of the input file contains special characters for a SAS variable name (such as a blank), SAS converts the character to an underscore. For example, the data value MY ID becomes the SAS variable name **MY\_ID**.

**YES** specifies that the IMPORT procedure generate SAS variable names from the data values in the first record of the imported Excel file.

**NO** specifies that the IMPORT procedure generate SAS variable names as F1, F2, F3, and so on.

**Default:** YES

**Restrictions:**

Valid only for DBMS=EXCEL.

Valid only for the IMPORT procedure.

When SAS reads the data value in the first row of the input file, SAS checks for invalid SAS name characters (such as a blank). Invalid characters are converted to an underscore. For example, the data value *Occupancy Code* becomes the SAS variable name **Occupancy\_Code**.

**RANGE= “range-name” | “absolute-range”**

subsets a spreadsheet by identifying the rectangular set of cells to import from the specified spreadsheet. 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. Absolute range identifies the top left cell that begins the range and the bottom right cell that ends the range. For example, 'C2..F12' represents cells within column C, row 2 and column F, row 12. If this statement is not specified, PROC IMPORT reads the entire spreadsheet as a range.

**Restriction:** This statement is valid for PROC IMPORT only.

**SHEET=** *worksheet-letter* | '*worksheet-name*'

identifies a particular spreadsheet in a WK $n$  file. Sheet names can be up to 15 characters long and are not case sensitive. A spreadsheet letter is a one- or two-letter alpha character. For WK1 files, there is only one spreadsheet letter: spreadsheet A. For WK3 and WK4 files, there can be up to 256 different spreadsheet letters: spreadsheet A - spreadsheet Z and spreadsheet AA - spreadsheet IV. The default value is A. For example, specifying **SHEET=B**; identifies spreadsheet B from a group of spreadsheets.

If this statement is not specified, the IMPORT procedure reads the first spreadsheet physically saved in the WK $n$  file. To be certain that IMPORT procedure reads the desired spreadsheet, you should identify the spreadsheet by specifying SHEET= option.

**Restriction:** Valid for only the IMPORT procedure.

**Example 1: Export a SAS Data Set to a WK4 File**

This example exports data from SAS data set SDF.EMPLOYEE to a WK4 file Employee.wk4 without variables FRSTNAME and MIDNAME.

```
LIBNAME SDF "&sasdir";
PROC EXPORT DATA=SDF.EMPLOYEE (DROP=FRSTNAME MIDNAME)
           OUTFILE="&tmpdir.Employee.wk4"
           DBMS=WK4 REPLACE;
RUN;
```

**Example 2: Import Data from a SAS Data Set from a WK4 File**

This example imports data from a WK4 file named invoice.wk4 into SAS data set named Invoice. It retrieves data from Sheet A, within range from left top cell, A1, to right bottom cell, D12. It then reads the first row of data in the range as SAS variable names.

```
PROC IMPORT OUT=WORK.Invoice
           DATAFILE="&wkndir.Invoice.wk4"
           DBMS=WK4 REPLACE;
           SHEET='A';
           RANGE='A1..D12';
           GETNAMES=YES;
RUN;
```

**Example 3: Export Data to a WK1 File from a SAS Data Set**

The next example exports data to a WK1 file named Orders.wk1 from a SAS data set named SDF.ORDER, without the variable SPECINST.

```
LIBNAME SDF "&sasdir";
PROC EXPORT DATA=SDF.ORDER (DROP=SPECINST)
           OUTFILE="&tmpdir.Orders.wk1"
           DBMS=WK1 REPLACE;
RUN;
```

**Example 4: Import Data from a WK1 File into a SAS Data Set**

This example imports data from a WK1 file named Orders.wk1 into a SAS data set named Test. It retrieves data from left top cell, B5, to right bottom cell, D12. SAS assigns name for each variable as VAR0, VAR1, and VAR2.

```
PROC IMPORT OUT=WORK.Test
            DATAFILE="&wkndir.orders.wk1"
            DBMS=WK1 REPLACE;
RANGE='B5..D12';
GETNAMES=NO;
RUN;
```

---

**dBase DBF Files****dBase DBF Files Essentials**

This section introduces dBase DBF files. It focuses on the terms and concepts that help you use SAS/ACCESS Interface to PC Files. For information about Visual FoxPro, see [“dBase DBF MEMO Files” on page 70](#).

DBF files are in a file format that dBASE creates. dBASE is a relational database management system for PC systems. DBF files can be created using a variety of PC software programs, such as Microsoft Excel.

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. This picture illustrates four database fields from Customer.DBF and highlights a database field and a record.

**Display 6.2 Database Field and Record**

<b>database field</b>			
CUSTOMER	CITY	STATE	COUNTRY
14324742	San Jose	CA	USA
14569877	Memphis	TN	USA
14898029	Rockville	MD	USA
26422096	La Rochelle		France
38763919	Buenos Aires		Argentina
46783280	Singapore		Singapore

**record****DBF 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. This list 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.

ALIAS: CHAR

#### NUMERIC (*N*, *n*)

specifies a decimal number. The *N* value is the total number of digits that are used to express the value (precision). The *n* value is the number of digits following the decimal point (scale). The maximum values allowed depend on which software product you are using.

**Table 6.13** dBase Maximum Numeric Values

dBASE Version	Maximum Numeric (N, n) Values
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, SAS interprets the value of the field as missing. 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 a 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 that you might be using to create DBF files to determine whether 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 row in a file. This type is 1 byte long and accepts these character values: Y, y, N, n, T, t, F, and f.

*Note:* dBASE also has data types called Memo, General, binary, and OLE. These data types are stored in an associated memo text file (a DBT file). These data types are not supported in the SAS/ACCESS Interface to PC Files.

## Setting Environment Variables and System Options

### MISSING VALUES

Missing numeric values are filled in with blanks 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. If you try to write a SAS file with a missing numeric variable to a DBF file, the corresponding DBF field is filled with the DBFMISCH character. Conversely, any numeric or float field in a DBF file that contains the DBFMISCH character is treated as a missing value when SAS read it.

You set the DBFMISCH environment variable in the SAS configuration file by using this syntax: **-set DBFMISCH value**

Valid values:

#### *any single character*

Type in any single character. For example, to fill missing numeric values with the character '9', enter **-set DBFMISCH 9**.

### 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**.

### DECIMAL SEPARATOR

Although the United States uses a decimal separator, other countries use different symbol characters. For example, some European countries use a comma. You must set the CTRYDECIMALSEPARATOR= system option to enable users to import or export data that is saved with a different decimal.

CTRYDECIMALSEPARATOR= system option syntax: **OPTIONS CTRYDECIMALSEPARATOR= value;**

Any character is valid. For example, to set a comma as the decimal separator submit this statement in SAS. **OPTIONS CTRYDECIMALSEPARATOR=',';**

This code uses the period character instead of the comma character. To save the numeric values in an exported DBF file while running SAS in a German environment.

```
OPTIONS CTRYDECIMALSEPARATOR='.';
PROC EXPORT DATA = sashelp.class
FILE= 'c:\temp\class.dbf'
DBMS=DBF REPLACE;
RUN;
```

## Supported SAS IMPORT and EXPORT Procedure Statements

The IMPORT | EXPORT method uses DBF file formats to access data in DBF Files on Linux, UNIX, and Microsoft Windows operating environments.

The method imports data from DBF files in versions 3, 4, and 5 formats. It exports data to DBF files with version 5 formats.

See [“Example 1: Export Data to a DBF File from a SAS Data Set”](#) on page 69 for additional information.

**Table 6.14** *IMPORT and EXPORT Procedure Statements for DBF Files*

Data Source	Syntax	Valid Values	Default Value	PROC IMPORT	PROC EXPORT
DBF	DBENCODING	Encoding-value	Current SAS session encoding	Yes	Yes
	GETDELETED	Yes   No	Yes	Yes	No

**DBENCODING = 12-byte SAS encoding-value**

indicates the encoding used to save data in DBF files. Encoding maps each character in a character set to a unique numeric representation, which results in a table of code points. A single character can have different numeric representations in different encodings.

For example, some DBF files were saved with pcoem850 encoding. When you are importing these DBF files in Microsoft Windows, specify:

**DBENCODING=pcoem850 .**

**Interaction:** The IMPORT procedure reads and transcodes data from pcoem850 to Microsoft Windows default WLATIN1.

**Note:** Refer to the SAS NLS User's Guide for information about transcoding and valid encoding values.

**GETDELETED= YES | NO**

indicates whether to write rows to the SAS data sets that are marked for deletion but have not been purged.

*YES* writes rows to the SAS data sets that are marked for deletion and have not been purged.

*NO* does not write rows to the SAS data sets that are marked for deletion and have not been purged.

**Alias:** GETDEL

**Example 1: Export Data to a DBF File from a SAS Data Set**

This example exports data to a DBF file, named test.dbf, from a SAS data set named SDF.EMPLOYEE, with a WHERE condition in the data set option.

```
LIBNAME SDF "&sasdir";
PROC EXPORT DATA=SDF.EMPLOYEE (WHERE=(HIREDATE is not missing))
  OUTFILE="&tmpdir.test.dbf"
  DBMS=DBF REPLACE;
RUN;
```

**Example 2: Import Data from a DBF File into a SAS Data Set**

This example imports data from a DBF file named invoice.dbf into SAS data set named TEST5. The data is imported without the DELETE flag field.

```
PROC IMPORT OUT=WORK.TEST5
  DATAFILE="&dbfdir.Invoice.dbf"
  DBMS=DBF REPLACE;
```

```

    GETDEL=NO;
  RUN;

```

### Example 3: Export Data to a DBF File from a SAS Data Set Using Encoding

This example exports data from a SAS data set named SDF.ORDERS to a DBF file named Oem850.dbf. The procedure translates SAS data from its current session encoding, to pcoem850 encoding and writes it to the DBF file.

```

PROC EXPORT DATA=SDF.ORDERS
    OUTFILE="&dbfdir.Oem850.dbf"
    DBMS=DBF REPLACE;
    DBENCODING=pcoem850;
  RUN;

```

### Example 4: Import and Translate Data from a DBF File

This example imports data from a DBF file named Oem850.dbf, which was saved with pcoem850 encoding. The procedure reads in the data and translates it from pcoem850 encoding to current SAS session encoding.

```

PROC IMPORT OUT=WORK.ORDERS
    DATAFILE="&dbfdir.Oem850.dbf"
    DBMS=DBF REPLACE;
    DBENCODING=pcoem850;
  RUN;

```

---

## dBase DBF MEMO Files

### Overview

When you use the DBFMEMO engine to import dBase Memo fields into the SAS System, the fields can be imported into multiple variables with numeric suffixes appended. When a **Memo** field is imported, each line of the field is imported as a separate variable. Each variable is given a numeric suffix to distinguish the particular line of the **Memo** field that was read. For example, a dBase Memo field of AE1 is imported as AE11, AE12, and so on.

All versions of dBase under Linux, UNIX, and Microsoft Windows are supported. Memo files have a .dbt (dBase) or .fpt (FoxPro and Visual FoxPro) file extension.

*Note:* Memo support is read only.

If a memo file exists with the same filename but with a .dbt or .fpt extension, the driver also reads the memo text for that file. It scans the memo file to determine how many lines comprise the largest individual memo and the lengths of the longest lines. It then splits memos into one variable per memo line. For example, the first three lines of a memo file called xyz would be named xyz01, xyz02, and xyz03.

## Import Data from a DBF File with Memo Field into a SAS Data Set

This example imports data from a DBF file named orders.dbf into a SAS data set named TEST.

```
PROC IMPORT OUT=WORK.TEST
  DATAFILE='orders.dbf'
  DBMS=DBFMEMO REPLACE;
RUN;
```

## JMP Files

### JMP File Essentials

A JMP file is a file format that the JMP software program creates. JMP is an interactive statistics package that is available for Microsoft Windows and Macintosh. For more information about a JMP concept or term, see the JMP documentation that is packaged with your system.

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.

Variable names can be up to 31 characters in length. When reading a JMP file, any embedded blank or special characters in a variable name are replaced with an underscore. This is noted in the log.

Base SAS supports access to JMP files. This enables you to access JMP files with the IMPORT and EXPORT procedures and the Import and Export Wizard without a license for SAS/ACCESS Interface to PC Files.

### JMP Missing Values

JMP supports a single missing value in all variable types other than character. When reading a JMP file, JMP missing values map to a single SAS missing value. When writing a JMP file, all SAS missing values map to a single JMP missing value.

### 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.

#### CHARACTER

specifies a field for character string data. Characters can be letters, digits, spaces, or special characters.

#### META

specifies how metadata contained in the specified data set is processed.

**meta= libref.member**

These record types are processed for the metadata. Table properties:

**code="text", subcode="prop", variable="",  
label=property\_text. Variable notes : code="note", subcode="",  
variable=varname, label=note\_text**

You can create any number of properties or notes.

When you add META= to the IMPORT procedure, an extra data set that contains the metadata is created when reading the JMP file.

When you add META= with the IMPORT procedure, value labels from the JMP file can be read and added to the format library.

When you add META= to the EXPORT procedure, the metadata contained in the specified data set is added to the built JMP file.

When you add META= with the EXPORT procedure, value lists are read from the format libraries when the JMP table is being built

The metadata data set contains the following fields:

- code
- index
- label
- name
- number

The names must match when the SAS metadata data set is being read.

*Note:* To do this easily, write the metadata from an existing JMP file to a SAS data set, view the variable information, and record the contents of the file.

#### 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 output format for the numeric variable is checked to determine whether 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 output format for the numeric variable is checked to determine whether 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 output format for the numeric variable is checked to determine whether it is a time format. If so, the SAS numeric value is scaled to a JMP time value with the appropriate time display format.

## Importing and Exporting JMP Files Data

SAS IMPORT | EXPORT utilities provide two methods for accessing JMP files.

### JMP File Formats (DBMS= JMP)

This IMPORT | EXPORT method uses JMP file formats to access data in JMP files on Linux, UNIX, and Microsoft Windows operating platforms. It imports data from JMP files saved with any version of JMP formats. It exports data to JMP files with V5 formats.

### PC Files Server (DBMS=PCFS)

This IMPORT | EXPORT method uses the client/server model to access data in JMP files on Microsoft Windows from Linux, UNIX, or Microsoft Windows operating environments. This method requires running the PC Files Server on Microsoft Windows.

*Note:* A filename with a .jmp extension is required.

## IMPORT Procedure and EXPORT Procedure Supported Syntax

### FMTLIB= *libref.format-catalog*

When importing a JMP file, this saves value labels to the specified SAS format catalog. When exporting a SAS data set to a JMP file, this writes the specified SAS format catalog to the JMP file.

### META= *libref.member-data-set*;

When importing a JMP file, this saves JMP metadata information to the specified SAS metadata set. When exporting a SAS data set to a JMP file, this writes the specified SAS metadata information to the JMP file.

**Alias:** METADATA

### Example 1: Export a SAS Data Set to a JMP File

This example exports a SAS data set named SDF.CUSTOMER to a JMP file named customer.jmp on a local system.

```
LIBNAME SDF "&sasdir";
PROC EXPORT DATA=SDF.CUSTOMER
           FILE="&tmpdir.customer.jmp"
           DBMS=JMP REPLACE;
RUN;
```

### Example 2: Export a SAS Data Set on UNIX to a JMP File

This example runs SAS on UNIX and requires access through PC Files Server. The example exports a SAS data set named SASHELP.CLASS to a JMP file named class.jmp.

```
LIBNAME SDF "&sasdir";
PROC EXPORT DBMS=PCFS DATA=SDF.CUSTOMER
           OUTFILE="&tmpdir.customer.jmp"
           REPLACE;
           SERVER=fileserv;
RUN;
```

**Example 3: Import a SAS Data Set from a JMP File**

This example imports to a SAS data set named CUSTOMER from a JMP file named customer.jmp on a local system.

```
PROC IMPORT OUT=WORK.CUSTOMER
           FILE="%jmdir.customer.jmp"
           DBMS=JMP REPLACE;
RUN;
```

**Example 4: Export a SAS Data Set on UNIX to a JMP File on Microsoft Windows**

This example exports a SAS data set named SDF.CUSTOMER to a JMP file named customer.jmp. Note that SAS is running on the UNIX operating platform and the JMP file is loaded on Microsoft Windows where PC Files Server is running.

```
LIBNAME SDF "%sasdir";
PROC EXPORT DATA=SDF.CUSTOMER
           FILE="%cstmpdir.customer.jmp"
           DBMS=PCFS REPLACE;
           SERVER="%server";
RUN;
```

**Example 5: Import Data from a JMP File on Microsoft Windows to a SAS Data Set on UNIX**

This example imports data from a JMP file named customer.jmp to a SAS data set named WORK.CUSTOMER. Note that SAS is running on a UNIX platform and the JMP file is located on Microsoft Windows where PC Files Server is running.

```
PROC IMPORT OUT= WORK.CUSTOMER
           FILE="%csjmdir.customer.jmp"
           DBMS=PCFS REPLACE;
           SERVER="%server";
RUN;
```

---

## Paradox DB File Formats

**Paradox File Essentials**

All versions of Paradox under Linux, UNIX, and Microsoft Windows are supported. Paradox files have a .db file extension. Paradox supports missing values. It does not have variables or value labels.

If a memo file with the same filename but with an .db extension exists, the memo text on that file is also read. The memo file is scanned to determine how many lines comprise the largest individual memo and the lengths of the longest lines. The driver then splits the memos into one variable per memo line. Memo support is read-only.

### Export a SAS Data Set to a PARADOX DB File

This example exports the SAS data set, SDF.CUSTOMER, to the Paradox DB file, customer.db, on a local system.

```
LIBNAME SDF "&sasdir";
PROC EXPORT DATA=SDF.CUSTOMER
            FILE="&tmpdir.customer.db"
            DBMS=DB REPLACE;
RUN;
```

### Import a SAS Data Set from a Paradox DB File

This example imports the SAS data set, WORK.CUSTOMER, from the Paradox DB file, customer.db, on a local system.

```
PROC IMPORT OUT=WORK.CUSTOMER
            FILE="&tmpdir.customer.db"
            DBMS=DB REPLACE;
RUN;
```

---

## SPSS SAV Files

### SAV File Essentials

All versions of SPSS under Microsoft Windows are supported. SPSS files have a .sav file extension. SPSS files that have short variable names are exported. See [“Example 1: Export a SAS Data Set to an SPSS SAV File” on page 77](#) for additional information.

### SPSS Data Types

#### MISSING VALUES

SPSS supports missing values. SAS missing values are written as SPSS missing values.

#### VARIABLE NAMES

SPSS variable names can be up to 32 characters in length. All alphabetic characters must be uppercase. The first character in a variable name can be an uppercase letter, a dollar sign (\$), or the “at” (@) symbol. Subsequent characters can be any of these characters, plus numerals, periods, number signs, or underscores.

SPSS reserves 13 words that are not allowed to stand alone as variable names: ALL, AND, BY, EQ, GE, GT, LE, LT, NE, NOT, OR, TO, and WITH. If the program encounters any of these as a variable name, it appends an underscore to the variable name to distinguish it from the reserved word. For example, **ALL** becomes **ALL\_**.

Invalid characters are converted to underscores unless they are encountered as the first character in a variable name. In that event, the “at” symbol (@) is used instead. For example, **%ALL** becomes **@ALL**.

When you are exporting to SPSS, SAS variable names that are longer than eight characters are truncated to eight characters. If the new name is truncated and results

in an existing name, the last character changes to a single digit (1,2, 3...) until the variable name becomes unique.

#### VALUE LABELS

SPSS stores value labels within the data file. The values are turned into format library entries as they are read with the IMPORT procedure. The name of the format includes its associated variable name, modified to meet the requirements of format names. The name of the format is also associated with a variable in the data set. You can use the FMTLIB = libref.format-catalog statement to save the formats catalog in a specified SAS library.

The EXPORT procedure saves the value labels that are associated with the variables when writing to an SPSS file. The procedure uses the formats that are associated with the variables to retrieve the value entries. You can use the FMTLIB = libref.format-catalog statement to tell SAS the location of the format catalog.

#### VARIABLE LABELS

SPSS supports variable labels. the EXPORT procedure writes the variable name to an SPSS file as the label if the variable name is not a valid SPSS name and no label exists.

#### DATA TYPES

SPSS supports numeric and character field types that map directly to SAS numeric and character fields. This list shows other SPSS data types and how the IMPORT procedure converts them to SAS formats.

Date, Jdate, Wkday, Qyr, Wkyr: Date, Jdate, Wkday, Qyr, Wkyr

Datetime, Dtime: Converts to a SAS datetime value and SAS datetime format.

Time: Converts to a SAS datetime value and SAS datetime format.

Adate: Converts to a SAS date value in the mmddy format.

Moyr: Converts to a SAS date value in the mmyy format.

When writing SAS data to an SPSS file, the EXPORT procedure converts data into SPSS variable types.

When exporting data, character fields have a maximum length of 256.

Numeric fields are 8 byte floating-point numbers, with these format conversions:

#### COMMA

Converts to SPSS format type comma.

#### DOLLAR

Converts to SPSS format type dollar.

#### DATE

Converts to SPSS format type date.

#### MMDDYY

Converts to SPSS format Adate.

#### MMYY

Converts to SPSS format Moyr.

#### DATETIME

Converts to SPSS format Dtime.

#### TIME

Converts to SPSS format Time.

## Importing and Exporting Data in SPSS Files

### SPSS Files (DBMS=SPSS)

This IMPORT | EXPORT method uses SPSS file formats to access data in SPSS files on Linux, UNIX, and Microsoft Windows operating platforms.

### PC Files Server (DBMS=PCFS)

This IMPORT | EXPORT method uses the client/server model to access data in SPSS files on Microsoft Windows from Linux, UNIX, or Microsoft Windows 64-bit operating environments. This method requires running the PC Files Server on Microsoft Windows.

*Note:* A filename with a .sav extension is required.

## Import Procedure and the Export Procedure Supported Syntax

### FMTLIB = libref.format-catalog

When importing an SPSS file, SAS saves value labels to a specified SAS format catalog. When exporting a SAS data set to an SPSS file, SAS writes the specified SAS format catalog to the SPSS file.

### Example 1: Export a SAS Data Set to an SPSS SAV File

This example exports the SAS data set SDF.CUSTOMER, to the SPSS file, CUSTOMER.SAV, on a local system.

```
LIBNAME SDF "&sasdir";
PROC EXPORT DATA=SDF.CUSTOMER
           FILE="&tmpdir.customer.sav"
           DBMS=SPSS REPLACE;
RUN;
```

### Example 2: Import a SAS Data Set from an SPSS SAV File

This example imports data from customer.sav, on a local system, to the SAS data set WORK.CUSTOMER.

```
PROC IMPORT OUT=WORK.CUSTOMER
           FILE="&tmpdir.customer.sav"
           DBMS=SPSS REPLACE;
RUN;
```

### Example 3: Import Data from an SPSS File and Apply FMTLIB= Option

This example imports the BANK.SAV data file to the “small” SAS data set and saves the value list from the SPSS file into the “FORMATS\_SPSS” format library.

```
LIBNAME A '.';
PROC IMPORT DATAFILE="BANK.SAV" OUT=SMALL DBMS=SAV;
           FMTLIB=A.FORMATS_SPSS;
RUN;
```

**Example 4: Export a SAS Data Set on UNIX to an SPSS File on Microsoft Windows**

This example exports a SAS data set named SDF.CUSTOMER to an SPSS file named CUSTOMER.SAV. Note that SAS is running on the UNIX operating platform. The SPSS file is loaded on Microsoft Windows where PC Files Server is running.

```
LIBNAME SDF "&sasdir";
PROC EXPORT DATA=SDF.CUSTOMER
            FILE="&tmpdir.customer.sav"
            DBMS=PCFS REPLACE;
            SERVER="&server";
RUN;
```

**Example 5: Import Data from an SPSS File on Microsoft Windows to a SAS Data Set on UNIX**

This example imports data from an SPSS file named CUSTOMER.SAV to a SAS data set named WORK.CUSTOMER. Note that SAS is running on a UNIX platform. The SPSS file is located on Microsoft Windows where PC Files Server is running.

```
PROC IMPORT OUT= WORK.CUSTOMER
            FILE="&tmpdir.customer.sav"
            DBMS=PCFS REPLACE;
            SERVER="&server";
RUN;
```

---

## Stata DTA Files

**DTA Files Essentials**

All versions of Stata under Microsoft Windows are supported. Stata files have a .dta file extension.

See [“Example 1: Export a SAS Data Set to a Stata File on a Local System”](#) on page 80 for additional information.

**DTA Data Types****FILES**

Import of all Stata versions under Microsoft Windows and UNIX are supported. Export of Stata version 8 and later is supported.

**MISSING VALUES**

Stata supports missing values. SAS missing values are written as Stata missing values.

**VARIABLE NAMES**

When using importing, Stata variable names can be up to 32 characters in length. The first character in a variable name can be any lowercase letter or uppercase letter or an underscore. Subsequent characters can be any of these characters, plus

numerals. No other characters are permitted. Stata reserves the 19 words shown in the table below, which are not allowed to stand alone as variable names:

**Table 6.15** *Stata Reserved Words*

<code>_all</code>	<code>_n</code>
<code>in</code>	<code>using</code>
<code>_pred</code>	<code>double</code>
<code>_b</code>	<code>_N</code>
<code>int</code>	<code>_weight</code>
<code>_rc</code>	<code>float</code>
<code>_coef</code>	<code>pi</code>
<code>long</code>	<code>with</code>
<code>_skip</code>	<code>if</code>
<code>_cons</code>	

If the program encounters any of these reserved words as variable names, it appends an underscore to the variable name to distinguish it from the reserved word. For example, `_N` becomes `_N_`.

When exporting, variable names greater than 32 characters are truncated. The first character in a variable name can be any lowercase letter or uppercase letter or an underscore. Subsequent characters can be any of these characters plus numerals. No other characters are permitted. Invalid characters are converted to underscores.

#### VARIABLE LABELS

Stata supports variable labels when using the `IMPORT` procedure. When exporting, if the variable name is not a valid Stata name and there is no label, the `EXPORT` procedure writes the variable name as the label.

#### VALUE LABELS

Stata stores value labels within the data file. The value labels are converted to format library entries as they are read with the `IMPORT` procedure. The name of the format includes its associated variable name modified to meet the requirements of format names. The name of the format is also associated with a variable in the SAS data set. You can use `FMTLIB= libref.format-catalog` statement to save the formats catalog under a specified SAS library.

When writing SAS data to a Stata file, the `EXPORT` procedure saves the value labels that are associated with the variables. The procedure uses the formats that are associated with the variables to retrieve the value entries. You can use the `FMTLIB= libref.format-catalog` statement to tell SAS where to locate the formats catalog.

*Note:* Numeric formats only.

#### DATA TYPES

Stata supports numeric field types that map directly to SAS numeric fields.

Stata date variables become numerics with a date format.

When writing SAS data to a Stata file, the EXPORT procedure converts data into variable type double. A SAS date format becomes a Stata date variable.

### Importing and Exporting Stata Data Files

#### Stata DTA Files (DBMS=STATA)

This IMPORT | EXPORT method uses Stata DTA file formats to access data in Stata DTA files on Linux, UNIX, and Microsoft Windows operating environments.

#### PC Files Server (DBMS=PCFS)

This IMPORT | EXPORT method uses the client/server model to access data in Stata files on Microsoft Windows from Linux, UNIX, or Microsoft Windows 64-bit operating environments. This method requires running the PC Files Server on Microsoft Windows.

*Note:* A filename with a .dta extension is required.

### Import and Export Procedures Supported Syntax

FMTLIB= libref.format-catalog. When importing a Stata file, SAS saves value labels to the specified SAS format catalog. When exporting a SAS data set to a Stata file, SAS uses formats that are associated with the variables to retrieve the value entries.

#### Example 1: Export a SAS Data Set to a Stata File on a Local System

This example exports the SAS data set SDF.CUSTOMER, to the Stata file, CUSTOMER.DTA, on a local system.

```
LIBNAME SDF "&sasdir";
PROC EXPORT DATA=SDF.CUSTOMER
            FILE="&tmpdir.customer.dta"
            DBMS=STATA REPLACE;
RUN;
```

#### Example 2: Import a SAS Data Set from a Stata File on a Local System

This example imports the SAS data set, WORK.CUSTOMER, from the Stata file, CUSTOMER.DTA, on a local system.

```
PROC IMPORT OUT=WORK.CUSTOMER
            FILE="&tmpdir.customer.dta"
            DBMS=STATA REPLACE;
RUN;
```

#### Example 3: EXPORT a SAS Data Set on UNIX to a Stata File on Microsoft Windows

This example exports a SAS data set named SDF.CUSTOMER to a Stata file named CUSTOMER.DTA. Note that SAS is running on the UNIX operating platform. The Stata file is loaded on Microsoft Windows where PC Files Server is running.

```
LIBNAME SDF "&sasdir";
PROC EXPORT DATA=SDF.CUSTOMER
            FILE="&tmpdir.customer.dta"
```

```
DBMS=PCFS REPLACE;  
SERVER="&server";  
RUN;
```

**Example 4: Import Data from a Stata File on Microsoft Windows to a SAS Data Set on UNIX**

This example imports data from a Stata file named CUSTOMER.DTA to a SAS data set named WORK.CUSTOMER. Note that SAS is running on a UNIX platform. The Stata file is located on Microsoft Windows where PC Files Server is running.

```
PROC IMPORT OUT= WORK.CUSTOMER  
FILE="&tmpdir.customer.dta"  
DBMS=PCFS REPLACE;  
SERVER="&server";  
RUN;
```



## Part 3

---

# LIBNAME Access and Excel Engines on Microsoft Windows

<i>Chapter 7</i>	
<b>Interaction and Functionality</b> .....	85
<i>Chapter 8</i>	
<b>The LIBNAME Engines</b> .....	89
<i>Chapter 9</i>	
<b>The LIBNAME Statement for Access and Excel on Microsoft Windows</b> .....	97
<i>Chapter 10</i>	
<b>Data Set Options</b> .....	103
<i>Chapter 11</i>	
<b>LIBNAME Options</b> .....	121
<i>Chapter 12</i>	
<b>Pass-Through Facility for Access and Excel on Microsoft Windows</b> .	
129	
<i>Chapter 13</i>	
<b>File-Specific Reference for Access and Excel on Microsoft Windows</b>	
.....	147



## Chapter 7

# Interaction and Functionality

---

<b>Overview of LIBNAME Statement for Access and Excel on Microsoft Windows</b> .....	<b>85</b>
LIBNAME Statement Advantages .....	85
<b>Sorting PC Files Data</b> .....	<b>85</b>
<b>Using SAS Functions with PC Files Data</b> .....	<b>86</b>
<b>Assigning a Libref Interactively</b> .....	<b>86</b>

---

## Overview of LIBNAME Statement for Access and Excel on Microsoft Windows

The SAS/ACCESS LIBNAME statement extends the SAS global LIBNAME statement to support assigning a libref to Microsoft Excel files and Microsoft Access files.

### *LIBNAME Statement Advantages*

Use of the SAS/ACCESS LIBNAME statement enables you to reference spreadsheets and databases directly in a DATA step or SAS procedure. You can also read from and write to a Microsoft Access object or Microsoft Excel object. See “[LIBNAME Statement Syntax](#)” on page 97 for additional information.

---

## 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. When you sort PC files data, the results might vary. Depending on whether the external spreadsheet or database places data has NULL

values. If the sort encounters NULL values, are they listed at the beginning or end of the result set. NULL values are translated in SAS to missing values.

---

## Using SAS Functions with PC Files Data

Librefs that refer to PC files with SAS functions might return a different value than the value returned when you use the functions with normal SAS data sets. 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.

Other function options can also vary. The LIBNAME function can accept an optional *SAS data-library* argument. When you use the LIBNAME function to assign or de-assign a libref that refers to PC files data, you omit this argument. For full details about how to use SAS functions, see *SAS DS2 Language Reference*.

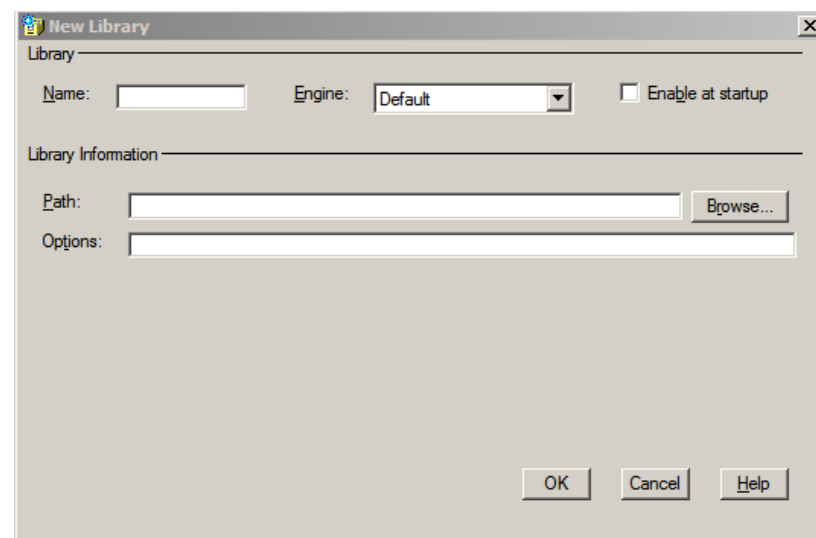
---

## Assigning a Libref Interactively

An easy way to associate a libref with PC files data is to use the New Library user interface. To access: Issue the DMLIBASSIGN command from the SAS session command line. You can also access the New Library window by right-clicking the libraries icon in the Explorer window and selecting **New**.

*Note:* Use of LIBNAME locks the file, and you must use a LIBNAME clear to unlock the file.

**Figure 7.1** New Library Window



- **Name:** up to eight alphanumeric characters. The library reference (libref) that you want to assign. Use the libref to point SAS to a SAS library or an external data source.
- **Engine:** select the engine that you want to use. The default engine enables SAS to choose which engine to use based on the existing data sets in a library. If no data sets exists in the same location as your new library, the Base SAS engine is assigned.

- **Enable at startup:** select to assign the specified libref to automatically when you open a SAS session. If you select **Enable at Startup**, you must go to **Solutions** ⇨ **Accessories** ⇨ **Registry Editor** ⇨ **CORE** ⇨ **OPTIONS** ⇨ **LIBNAMES** to remove the libref.
- **Path:** specifies the path of the libref.
- **Library Information:** represents 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.



## Chapter 8

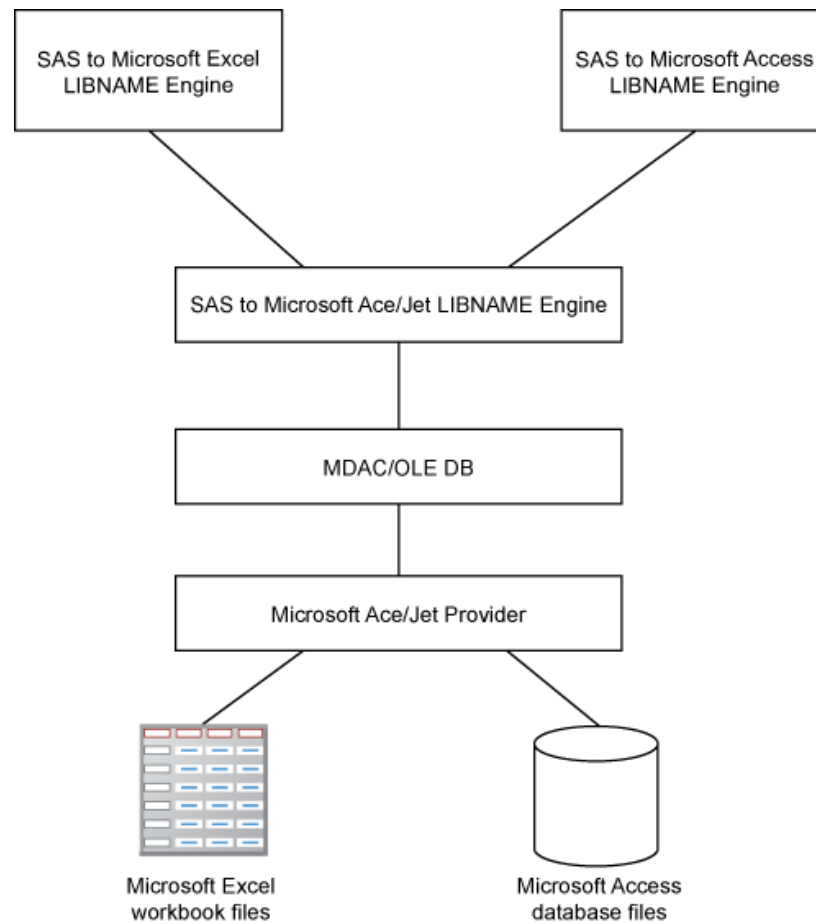
# The LIBNAME Engines

---

<b>Overview: LIBNAME Engines</b> .....	<b>89</b>
<b>Software Requirements</b> .....	<b>90</b>
<b>Macro Variables</b> .....	<b>90</b>
<b>System Options</b> .....	<b>91</b>
<b>Dictionary</b> .....	<b>91</b>
SASTRACE .....	91
SASTRACELOC .....	94

---

## Overview: LIBNAME Engines




---

## Software Requirements

On Windows, SAS/ACCESS Interface to PC Files supports a LIBNAME engine for Microsoft Access database files and a LIBNAME engine for Excel workbook files. These two engines share most of code that calls OLE DB APIs internally. They require that you have installed Microsoft Data Access Components (MDAC) and Microsoft Jet (Joint Engine Technology) or Microsoft Ace (for 2007 and above) provider software. By default, SAS software installs and checks the required Microsoft software.

---

## Macro Variables

The automatic macro variables `SYSDBMSG`, `SYSDBRC`, `SQLXMSG`, and `SQLXRC` are portable, but the SAS/ACCESS engine and the Microsoft Jet Engine determine their values. Initially, the macro variables `SYSDBMSG` and `SQLXMSG` are blank, whereas `SYSDBRC` and `SQLXRC` are set to 0.

SAS/ACCESS generates several return codes and error messages while it processes your programs. This information is available to you through these SAS macro variables:

**SYSDBMSG**

contains Jet provider-specific error messages that are generated when you use SAS/ACCESS software to access Microsoft Access or Excel data.

**SYSDBRC**

contains Jet provider-specific error codes that are generated when you use SAS/ACCESS software to Microsoft Access or Excel data. Error codes that are returned are text, not numbers.

In this statement, %SUPERQ masks special characters such as &, %, and any unbalanced parentheses or quotation marks that might exist in the text stored in the SYSDBMSG macro.

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

These special characters can cause unpredictable results if you use this statement, so it is more advantageous to use %SUPERQ.

You can also use SYMGET to retrieve error messages:

```
MSG=SYMGET ("SYSDBMSG" );
```

**SYMGET example:**

```
DATA_NULL_ ;
    MSG=SYMGET ("SYSDBMSG" );
    PUT MSG;
RUN;
```

The pass-through facility generates return codes and error messages that are available to you through these SAS macro variables:

**SQLXMSG**

contains Jet provider-specific error messages.

**SQLXRC**

contains Jet provider-specific error codes.

SQLXMSG and SQLXRC can be used only with the pass-through facility. See [“Return Codes” on page 130](#) for additional information.

The contents of the SQLXMSG and SQLXRC macro variables can be printed in the SAS log by using the %PUT macro. SQLXMSG is reset to a blank string and SQLXRC is reset to 0 when any pass-through facility statement is executed.

## System Options

SASTRACE = and SASTRACELOC = are SAS system options that have specific SAS/ACCESS applications.

## Dictionary

### SASTRACE

Generates trace information from a DBMS engine.

**Valid in:** OPTIONS statement, configuration file, SAS invocation

**Default:** None

## Syntax

SASTRACE=','d'|','d'|','d'|','s'

### Syntax Description

**'','d'**

specifies that all of these SQL statements sent to the Microsoft Jet engine are sent to the SAS log:

```
SELECT    INSERT
UPDATE    DROP
CREATE    DELETE
```

**'','d',''**

specifies that all routine calls are sent to the log. When this option is selected, all function enters and exits, as well as pertinent parameters and return codes, are traced. However, the information varies from engine to engine.

This option is most useful if you are having a problem and need to send a SAS log to technical support for troubleshooting.

**'d',''**

specifies that all OLE DB API calls for connection information, column bindings, column error information, and row processing are sent to the log. This option is most useful if you are having a problem and need to send a SAS log to technical support for troubleshooting.

**'','s'**

specifies that a summary of timing information for calls made to the DBMS is sent to the SAS log.

## Details

The SASTRACE= option has behavior that is specific to SAS/ACCESS software. SASTRACE= is a very powerful tool to use when you want to see the commands that the SAS/ACCESS engine sends. SASTRACE= output is DBMS-specific. However, most SAS/ACCESS engines show statements like SELECT or COMMIT as the DBMS processes them for the SAS application. These details can help you manage SASTRACE= output for your files:

- When using SASTRACE= on PC platforms, you must also specify SASTRACELOC= options, as described in [“SASTRACELOC” on page 94](#).
- To turn SAS tracing off, specify this option:

```
options sastrace=off;
```

- Log output is much easier to read if you specify NOSTSUFFIX. Here is an example:

```
OPTIONS SASTRACE=','d',' SASTRACELOC=SASLOG NOSTSUFFIX;
```

*Note:* By default, Microsoft Access and Microsoft Excel LIBNAME engines use ROWSET\_INSERT instead of executing the SQL INSERT command. You do not see the INSERT statement in the trace log when inserting rows into a table. You can use the LIBNAME statement or the SQL\_INSERT=YES option to see the INSERT statement in the trace log.

## Example: Review SQL Statements

This example specifies SASTRACE =',,,d' so that SQL statements are sent to the SAS log.

```
DATA work.winter_birthdays;
  INPUT empid birthdat DATE9. lastname $18.;
  FORMAT birthdat DATE9.;
DATALINES;
678999 28DEC1966 PAVEO          JULIANA          3451
456788 12JAN1977 SHIPTON        TIFFANY          3468
890123 20FEB1973 THORSTAD        EDVARD           3329
;
RUN;
OPTIONS SASTRACE=',,,d' SASTRACELOC=saslog nostsuffix;
LIBNAME mydblib 'c:\sasdemo\demo.mdb' ;
PROC DELETE DATA=mydblib.snow_birthdays; RUN;
DATA mydblib.snow_birthdays;
  SET work.winter_birthdays;
RUN;
PROC PRINT DATA=mydblib.snow_birthdays;
RUN;
LIBNAME mydblib CLEAR;
```

Here is the SQL statements in the SAS log output.

```

1  DATA WORK.winter_birthdays;
2  INPUT empid birthdat date9. lastname $18.;
3  FORMAT birthdat DATE9.;
4  DATALINES;
NOTE: The data set WORK.WINTER_BIRTHDAYS has 3 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time          0.06 seconds
      CPU time           0.04 seconds
8  ;
9  RUN;
10
11  OPTIONS SASTRACE=',,,d' SASTRACELOC=SASLOG NOSTSUFFIX;
12  LIBNAME mydblib 'c:\sasdemo\demo.mdb' ;
NOTE: Libref MYDBLIB was successfully assigned as follows:
      Engine:           ACCESS
      Physical Name:   c:\sasdemo\demo.mdb
13
14  proc delete data=mydblib.snow_birthdays; RUN;
Jet_0: Executed:
DROP TABLE `snow_birthdays`
NOTE: Deleting MYDBLIB.snow_birthdays (memtype=DATA) .
NOTE: PROCEDURE DELETE used (Total process time):
      real time          0.01 seconds
      CPU time           0.00 seconds
15
16  DATA mydblib.snow_birthdays;
17  SET work.winter_birthdays;
18  RUN;
NOTE: SAS variable labels, formats, and lengths are not written to DBMS tables.
Jet_1: Executed:
CREATE TABLE `snow_birthdays` (`empid` Double,`birthdat` DateTime,`lastname`
      VarChar(18))
NOTE: There were 3 observations read from the data set WORK.WINTER_BIRTHDAYS.
NOTE: The data set MYDBLIB.snow_birthdays has 3 observations and 3 variables.
NOTE: Successfully Inserted 3 row(s)
NOTE: DATA statement used (Total process time):
      real time          0.03 seconds
      CPU time           0.01 seconds
19
20  LIBNAME mydblib clear;
NOTE: Libref MYDBLIB has been deassigned.

```

---

## SASTRACELOC

Writes SASTRACE information to a specified location.

**Valid in:** OPTIONS statement, configuration file, SAS invocation

**Default:** stdout

---

### Syntax

SASTRACELOC= stdout | SASLOG | FILE 'path-and-filename'

### Details

specify where to put the trace messages that are generated by SASTRACE=. By default, the output goes to the default output location for your operating environment. You can send the output to the SAS log by specifying SASTRACELOC=SASLOG.

**Example: Write Information to the SAS Log**

This example runs on a PC platform and writes trace information to the SASTRACELOC file in the work directory on the C drive.

```
options sastrace='d,,d' sastraceloc=file 'c:\work\trace.log';
```



## Chapter 9

# The LIBNAME Statement for Access and Excel on Microsoft Windows

---

Dictionary .....	97
LIBNAME Statement Syntax .....	97

---

## Dictionary

---

### LIBNAME Statement Syntax

Associates a SAS libref with a workbook or database.

**Valid in:** Anywhere

---

### Syntax

```
LIBNAME < libref> engine-name <physical-path and filename>
<SAS/ACCESS engine-connection-options>
<SAS/ACCESS LIBNAME-options> ;
```

```
LIBNAME libref CLEAR|_ALL_;
```

```
LIBNAME libref LIST|_ALL_;
```

### Optional Arguments

#### *libref*

is any SAS name that associates SAS with the SAS library where the spreadsheet or database is stored.

#### *engine-name*

is the SAS/ACCESS engine name for your PC file format. The SAS/ACCESS LIBNAME statement associates a libref with a SAS/ACCESS engine that supports connections to a particular PC file type. The supported engine-names are as follows:

EXCEL for Microsoft Excel 5, 95, 97, 2000, 2002, 2003, 2007, and 2010.

ACCESS for Microsoft Access 97, 2000, 2002, 2003, 2007, and 2010.

**Note:** The engine name is optional if the *physical-path-filename.ext* is specified. The file extension provides enough information for SAS.

#### **Examples:**

This example illustrates two options for using LIBNAME statement with physical filename, including the file's extension. The *accdb* file extension provides SAS information about the data type. The PATH= option provides SAS with the location of the data.

```
LIBNAME libref ACCESS PATH='C:\PCFData\Demo.accdb';
```

or

```
LIBNAME libref 'C:\PCFData\Demo.accdb';
```

This LIBNAME statement specifies *xdb* as a reference to a SAS library. The EXCEL engine specifies the engine that supports the connection to the file type .XLSX.

```
LIBNAME xdb EXCEL PATH='C:\PCFData\Demo.xlsx';
```

### ***physical-path-filename.ext***

is the *physical-path and filename.ext* of the data source.

- Microsoft Excel data source extensions include: .XLS, .XLSB, .XLSM, .XLSX.
- Microsoft Access extensions include: .MDB and .ACMDB.

**Note:** Providing the *physical-path-filename.ext* sets the NOPROMPT ENGINE option.

**See:** ["Connection Details" on page 101](#) [Details on page 101](#)

**Example:** Physical Path and Filename Omitting Engine Name:

```
LIBNAME xdb 'C:\PCFData\Demo.xlsx';
```

```
LIBNAME adb 'C:\PCFData\Demo.accdb';
```

### **CLEAR**

clears one libref.

Specify *libref* to disassociate a single libref.

### **ALL**

specifies that the CLEAR or LIST argument applies to all librefs.

### **LIST**

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

Specify *libref* to list the attributes of a single SAS/ACCESS library or SAS library. Specify ALL to list the attributes of all librefs in your current session.

#### **Examples:**

List the attributes of a single library:

```
LIBNAME LIBREF LIST;
```

List the attributes of all the libraries.

```
LIBNAME ALL LIST;
```

### ***SAS/ACCESS engine-connection-options***

provide connection options 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 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 Details" on page 101](#)

### ***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 that you do not need to specify any of these advanced options.

See: “LIBNAME Statement Syntax” on page 101

### Connection Options

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

#### INIT= *connection-string*

used for OLEDB, specifies an initialization string that SAS uses when connecting to a data source.

**Alias:** INIT\_STRING

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

**Note:** This is used rarely, specifically in SAS 9.2 and later, with the Microsoft ACE engine being dominant.

#### MSENGINE= *ACE* | *JET*

determines the database engine used for accessing the Microsoft Excel file or Microsoft Access database. The Microsoft Jet engine is older and supports formats up to 2003. The Microsoft Ace engine supports 2007 and older formats.

**Default:** ACE

#### PATH= *data-source full-path and file-name with extension*

specifies the full path and filename for the data source file. The PATH= option can be for the Microsoft Access database file or Microsoft Excel workbook file. The PATH= value is treated the same as the physical filename.

**Alias:** DATASRC | DS

**Restriction:** Use PATH= only when the physical filename is not specified in the LIBNAME statement

**Requirement:** Use of this option requires the engine name to be specified. The file extensions .mdb for Microsoft Access and .xls for Excel are also required.

#### PROMPT= *YES* | *NO* | *REQUIRED* | *NOPROMPT* | *PROMPT* | *UDL*

determines whether you are prompted for connection information that supplies the data source information

*YES* enables prompting with a Data Link Properties dialog box. To write the initialization string to the SAS log, submit this code immediately after connecting to the data source:

```
%PUT %SUPERQ (SYSDBMSG) ;
```

*NO* prompting is not available. You must specify the data source as a physical filename or complete path.

*REQUIRED* enables connection without prompting for more information. If a valid connection is not specified, you are prompted for the connection options. The prompt enables you to change the data source file and other properties.

*Note:* You must specify a valid physical filename for a successful connection.

*NOPROMPT* prompting is not available.

*PROMPT* enables prompting for connection information for the data source.

*UDL* enables browsing so you can 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) as in this example.

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

**Alias:** UDL\_FILE

**Restrictions:**

This option should not be used with a physical filename or other connection options, such as PATH and INIT.

This option does not support SAS filerefs. The SYSDBMSG macro variable is set on successful completion. For more information, refer to Microsoft documentation about using data link.

### **Options for Access LIBNAME Statements Only**

The following options are for Access LIBNAME statements only.

**DBPASSWORD= *database-file-password***

enables you to access database files with database-level security. This security level can be defined instead of user-level security.

**Alias:** DBPWD | DBPW | PASS | PASSWORD

**Restriction:** Microsoft Access Database only.

**Note:** Database password is case sensitive.

**DBSYSFILE= *workgroup-information-file***

specifies the workgroup information file. This file contains a collection of information defined for the Microsoft Access database. User, group accounts, and passwords that you create, are saved in the workgroup information file.

**Alias:** SYSTEMDB

**Restriction:** Microsoft Access database files only.

**PASSWORD= *user-password***

specifies a password required by the data source for the user account.

**Alias:** PWD | PW | PASS | PASSWORD

**Note:** Passwords are case sensitive.

**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 on your .mdb file, you need to use the USER= and PASSWORD= options to access your file.

**Alias:** UID

**Restriction:** Microsoft Access database files only.

**Note:** Use the SERVERUSER= option to connect to a server.

### **Option for Excel LIBNAME Statements Only**

**VERSION= 2010 | 2007 | 2003 | 2002 | 2000 | 97 | 95 | 5**

sets the version for a new Excel workbook.

**Alias:** VER

**Default:** 97 for .xls files; 2007 for .xlsb and .xlsx files.

**Restriction:** Excel workbooks only.

**Notes:**

Excel 2010, 2007, 2003, 2000, and 97 share the same .xls file format. Excel 5 and 95 share a different file format.

You do not need to specify this option for an existing Excel file.

## Details

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 if 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,

```
MyPCLib.Employees_Q2
```

- MyPCLib is a SAS libref that points to a particular group of external data objects.
- 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.

Clearing Libref from a SAS Library. To disassociate or clear a libref, use a LIBNAME statement. Specify the libref and the CLEAR option. SAS/ACCESS disconnects from the data source and closes any free threads or resources that are associated with that libref's connection.

To clear a single libref:

```
LIBNAME mypclub CLEAR;
```

To clear all User-defined librefs:

```
LIBNAME CLEAR;
```

Writing SAS 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 libraries to the SAS log.

To list attributes of a single library:

```
LIBNAME mypclub LIST;
```

To list attributes of all libraries:

```
LIBNAME _ALL_ LIST;
```

Assigning a Libref with a SAS/ACCESS LIBNAME Statement. This statement assigns the libref, mymdb to 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. You can also use it as a data source in any DATA step or SAS procedure.

In this 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 use the libref and data set with any other SAS procedure. This statement prints the Staff table:

```
PROC PRINT DATA=mymdb.staff;  
RUN;
```

This statement lists the database objects in the MyMdb library:

```
PROC DATASETS LIBRARY=mymdb;  
QUIT;
```

This statement associates the SAS libref MYXLS with an Excel workbook:

```
LIBNAME myxls 'c:\demo.xls';
```

## See Also

[“LIBNAME Statement” on page 147](#)

## Chapter 10

# Data Set Options

---

<b>Overview of Data Set Options</b> . . . . .	<b>103</b>
<b>Dictionary</b> . . . . .	<b>104</b>
AUTOCOMMIT . . . . .	104
COMMAND_TIMEOUT . . . . .	104
CURSOR_TYPE . . . . .	105
DBCOMMIT . . . . .	105
DBCONDITION . . . . .	106
DBCREATE_TABLE_OPTS . . . . .	107
DBFORCE . . . . .	107
DBGEN_NAME . . . . .	108
DBKEY . . . . .	109
DBLABEL . . . . .	109
DBMAX_TEXT . . . . .	110
DBNULL . . . . .	111
DBNULLKEYS . . . . .	112
DBSASLABEL . . . . .	112
DBSASTYPE . . . . .	113
DBTYPE . . . . .	114
DBTYPE . . . . .	114
ERRLIMIT . . . . .	115
INSERT_SQL . . . . .	116
INSERTBUFF . . . . .	116
NULLCHAR . . . . .	117
NULLCHARVAL . . . . .	118
READBUFF . . . . .	118
SASDATEFMT . . . . .	119

---

## Overview of Data Set Options

Specify SAS/ACCESS data set options on a SAS data set when accessing PC files data with the LIBNAME statement, see “[LIBNAME Statement Syntax](#)” on page 97. A data set option applies only to the data set on which it is specified. The option remains in effect for the duration of the DATA step or procedure.

This example illustrates the format of data set options:

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

The CNTLLEV=, DROP=, FIRSTOBS=, IN=, KEEP=, OBS=, RENAME=, and WHERE SAS data set options can be used when you access PC files data. SAS/ACCESS interfaces do not support the REPLACE= SAS data set option.

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

---

## Dictionary

---

### AUTOCOMMIT

Determines whether the ACCESS engine commits (saves) updates as soon as you submit them.

**Valid in:** DATA and PROC steps

**Default:** LIBNAME option setting

---

#### Syntax

AUTOCOMMIT=YES | NO

#### Syntax Description

##### 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.

---

### 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:** LIBNAME option setting

**See:** To assign this option to a group of tables, use the COMMAND\_TIMEOUT option specified in [“LIBNAME Options” on page 121](#).

---

#### Syntax

COMMAND\_TIMEOUT= *number-of-seconds*

#### Syntax Description

##### number-of-seconds

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

---

## CURSOR\_TYPE

Specifies the cursor type for read-only cursors and for cursors to be updated.

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

**Table 10.1** OLE DB Properties Applied to an Open Row Set

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” on page 121](#).

---

## DBCMMIT

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:** LIBNAME option default: 1000

---

## 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 the DBCOMMIT= option and the ERRLIMIT= option, and these options collide during processing, then the DBCOMMIT= option is issued first and the rollback is issued second. Because the DBCOMMIT= option is issued before the ERRLIMIT= option, the DBCOMMIT= option overrides the ERRLIMIT= option in this situation.

## Example: Issue Automatic Commit Statement

```
/* a commit is issued after every 10 rows are inserted */
DATA myxls.dept (DBCOMMIT=10);
  SET mysas.staff;
RUN;
```

## See Also

- [“ERRLIMIT” on page 115](#)
- [“LIBNAME Options” on page 121](#)

---

## DBCONDITION

Specifies criteria for subsetting and ordering data.

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

**Default:** 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 option is ignored when you use DBCONDITION.

## See Also

[“DBKEY” on page 109](#)

---

## DBCREATE\_TABLE\_OPTS

Specifies data source-specific syntax to add to the CREATE TABLE statement.

**Valid in:** DATA and PROC steps

**Default:** 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” on page 121](#).

---

## DBFORCE

Specifies whether to force the truncation of data during insert processing.

**Valid in:** DATA and PROC steps

**Default:** 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 width.

#### **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 disallowed characters.

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

**Default:** 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. The English characters converted from kanji are often 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” on page 121](#).

---

## DBKEY

Improves performance for a join with a large source table and a small SAS data set (specifies a column to use as an index).

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

**Default:** 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

**Default:** 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 a Microsoft Access or a Microsoft Excel file.

**Example: Specify Label Use**

In this 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. When DBLABEL=YES, you can use the label 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

**Default:** 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 typically used with a very long character data type.

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

## See Also

To assign this option to a group of tables, use the DBMAX\_TEXT= option specified in [“LIBNAME Options” on page 121](#).

---

## DBNULL

Specifies 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:** YES

---

### Syntax

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

### Syntax Description

**YES**

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

**NO**

specifies that a 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. 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:** only the Access engine supports this option. The Excel engine does not support this option.

### Example: Specify NULL Value Disposition

In this example, using the DBNULL option prevents the EmpId and Jobcode columns in the new MyDBLib.MyDept2 table from accepting null values. If the Employees table contains any 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 this 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 any null values in any column other than the Jobcode column, the DATA step fails.

```
DATA mydblib.mydept3 (DBNULLL1=( _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:** 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, use DBNULLKEYS=YES. When you specify DBNULLKEYS=YES, and specify a column that is not defined as DBKEY=NOT NULL, SAS generates a WHERE clause that finds NULL values.

If you specify DBKEY=COLUMN and COLUMN is not defined as NOT NULL, SAS generates a WHERE clause.

Example:

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

SAS generates the WHERE clause once and uses it for any value, NULL, or NOT NULL in the column. This syntax can be much less efficient than the shorter form of the WHERE clause. When you specify DBNULLKEYS=NO, or specify a column that is NOT NULL in the DBKEY= option, SAS generates a simple WHERE clause.

If there are no NULL values in the transaction or master table for the columns, use DBNULLKEYS=NO. If you specify DBNULLKEYS=NO and DBKEY=COLUMN, SAS generates a shorter form of the WHERE clause. SAS generates the WHERE clause even if the column DBKEY specifies is defined as NOT NULL.

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

### See Also

- [“DBKEY” on page 109](#)
- [“LIBNAME Options” on page 121](#)

---

## 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:** 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 writing data into SAS from a 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:** 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 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.

*Note:* 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

**Default:** none

---

### Syntax

```
DBTYPE=(<column-name-1> =<data-source-type> ...
<column-name-1> =<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 it writes 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.

### Example: Specify the Data Type to Use

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;
```

---

## 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

**Default:** none

---

### Syntax

```
DBTYPE=(<column-name-1> =<data-source-type> ...
<column-name-1> =<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 it writes 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.

### Example: Specify the Data Type to Use

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;
```

---

## ERRLIMIT

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

**Valid in:** DATA and PROC steps

**Default:** 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 nonzero value for the DBCOMMIT= option, rollbacks affected by the ERRLIMIT= option might not be complete. Records already committed by DBCOMMIT= option are not processed again.

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

## Example: Specify Error Limit

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;
```

## See Also

[“DBCMMIT” on page 105](#)

## INSERT\_SQL

Determines the method to use to insert rows into a data source.

**Valid in:** DATA and PROC steps

**Default:** 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” on page 121](#).

## INSERTBUFF

Specifies the number of rows in a single insert.

**Valid in:** DATA and PROC steps

**Default:** 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 greater than INSERTBUFF=1, the SAS notes indicating success or failure of the insert operation might be incorrect. These notes are generated for a single insert. This is also true, when multiple inserts are performed.

*Note:* PC Files Server does not support INSERTBUFF= option with a value higher than 1 for writing data to Excel. It ignores this option when writing data to Excel.

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

*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

- [“DBCOMMIT” on page 105](#)
- [“Overview of Data Set Options” on page 103](#)

---

## NULLCHAR

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

**Valid in:** DATA and PROC steps

**Default:** SAS

---

## Syntax

NULLCHAR= YES | NO

## Syntax Description

### 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.

in conjunction with the NULLCHARVAL= data set option, NULLCHARVAL= 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.

## See Also

[“DBKEY” on page 109](#)

## NULLCHARVAL

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

**Valid in:** DATA and PROC steps

**Default:** a blank character

## Syntax

NULLCHARVAL=<'character-string'>

## Details

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

This option works with the NULLCHAR= option. NULLCHAR= determines whether a SAS character NULL value is treated as a NULL value.

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

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

## See Also

["DBKEY=" on page 109](#)

## READBUFF

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

**Valid in:** DATA and PROC steps

**Default:** 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. Because SAS stores the rows in memory, higher values for READBUFF= use more memory. If too many rows are selected at once, then the rows that are returned to the SAS application can 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 as specified in “LIBNAME Options” on page 121.

---

## SASDATEFMT

Changes the SAS date format of a data source column.

**Valid in:** DATA and PROC steps

**Default:** 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 informat. For example, DATETIME21.2 is both a SAS format and informat, so it is valid 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 column, convert the SAS date values to appropriate values. The SAS DATEFMT= option enables you to convert date values from a SAS date format to different SAS date format.

Use SAS DATEFMT= to prevent date type mismatches under these 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 SAS DATEFMT= option does not currently handle these date types. You can use a PROC SQL view to convert the source data to a SAS date format, as you retrieve the data. You can also use a format statement in other contexts.

## Chapter 11

# LIBNAME Options

---

<b>Dictionary</b> .....	<b>121</b>
LIBNAME Options .....	121

---

## Dictionary

### LIBNAME Options

provides additional control over the way that SAS processes PC files data.

**Interaction:** For many tasks that you perform, you do not need to specify any of these advanced options. Many of these options are also available as data set options.

---

### Syntax

#### **Optional Arguments**

##### **ACCESS= *READONLY***

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

##### **AUTOCOMMIT= YES | NO**

specifies whether the ACCESS engine commits updates when submitted.

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

NO specifies that updates are committed when SAS reaches the end of the file.

##### **COMMAND\_TIMEOUT= *number-of-seconds***

specifies the number of seconds that pass before a data source command times out.

**Alias:** TIMEOUT

**Default:** 0 (no time-out)

##### **CONNECTION= SHAREDREAD | UNIQUE | GLOBALREAD**

specifies whether operations against a single libref share a connection to the data source. Also specifies 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 a SAS application.

*GLOBALREAD* specifies that all READ operations that access data source tables with multiple librefs, share a single connection if the librefs are created by LIBNAME statements that specify:

- identical values for the CONNECTION= option
- identical values for the CONNECTION\_GROUP= option
- identical values for all data source connection options

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

**Default:** SHAREDREAD

**See:** [CONNECTION\\_GROUP](#) on page 122

### CONNECTION\_GROUP

specifies that operations against multiple LIBREFS share a single connection to the data source. Also specifies that operations against multiple pass-through facility CONNECT statements share a single connection to the data source.

### CURSOR\_TYPE= KEYSSET\_DRIVEN | STATIC

specifies the cursor type for read-only cursors and for cursors to be updated. If you do not set CURSOR\_TYPE=, the Jet provider that you are using determines the default.

*KEYSET\_DRIVEN* specifies that the cursor determines which rows belong to the result set when you open the cursor. Changes that are made to these rows are reflected as you move the cursor. The OLE DB property DBPROP\_OTHERUPDATEDELETE= TRUE for key set driven cursors.

*STATIC* specifies that the complete result set is built when you open the cursor. No changes that are made to the result set are reflected in the cursor. Static cursors are read-only. The OLE DB property DBPROP\_OTHERUPDATEDELETE= FALSE for static cursors.

**Alias:** CURSOR

### DBCMMIT= 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 DBCMMIT= 0, a commit is issued only once (after the procedure or DATA step completes). If the DBCMMIT= option is explicitly set, SAS/ACCESS fails any update that has a WHERE clause.

**Default:** 1,000 [inserting]

0 [updating; commit occurs when data set or procedure completes]

**Note:** If you specify both DBCMMIT= and ERRLIMIT= options, and these options collide during processing, DBCMMIT= is issued first and ERRLIMIT= is issued second. Because the DBCMMIT= option is issued before the ERRLIMIT= option, the DBCMMIT= option overrides the ERRLIMIT= option in this situation.

**DBGEN\_NAME= DBMS | SAS**

specifies that the data source columns are renamed and the format used for the names.

*DBMS* specifies that:

- the data source columns are renamed to valid SAS variable names.
- invalid characters are converted to underscores.
- if a *column-name* is converted to an existing name, then a sequence number is appended to the new name.

*SAS* specifies that data source columns are renamed to the format `_COLn`, where *n* is the column number. Zero-based, starts at zero.

**Default:** DBMS

**DBMAX\_TEXT= integer between 1 and 32,767**

specifies the maximum length for a character string. Character strings longer than 32,767 are truncated. This option only applies when you are reading, appending, and updating character data in a Microsoft Access database or Microsoft Excel workbook from SAS.

**Default:** 1,024

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

**DBNULLKEYS= YES | NO**

specifies whether there might be NULL values in the columns.

*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 this syntax:

```
WHERE ((COLUMN = ?) or ((COLUMN IS NULL) AND (? IS NULL)));
```

This syntax enables SAS to prepare the statement once and use it for any (NULL or NOT NULL) in the column.

*Note:* This syntax has the potential to be much less efficient than the shorter form the WHERE clause presented below.

In the `DBKEY=` option, and there might be NULL values in a column, specify `DBNULLKEYS= YES`. SAS generates a WHERE clause to find NULL values. If you do either of the following steps:

- specify `DBNULLKEYS= YES` and a column not defined NOT `DBNULLKEYS=` data set option, SAS generates a WHERE clause to find NULL values.
- specify `DBKEY=` and `COLUMN` is not defined as NOT NULL, SAS generates a WHERE clause similar to the following:

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

*NO* If you:

- specify `DBNULLKEYS=NO` or specify a column that is defined as NOT NULL in the `DBKEY=` option, SAS generates a simple WHERE clause.

- know that there are no NULL values in the transaction or the master table for the columns specified in the DBKEY= option, use DBNULLKEYS=NO.
- specify DBNULLKEYS=NO and specify DBKEY=COLUMN, SAS generates a shorter form of a WHERE clause. The WHERE clause is generated whether the column that DBKEY= is defined as NOT NULL.

**Default:** YES

**Note:** This syntax enables SAS to prepare the statement once and use it for any value, NULL, or NOT NULL in the column.

**Example:**

Example WHERE (COLUMN = ?)

### **DBSASLABEL= COMPAT | NONE**

specifies whether SAS/ACCESS saves the data source column names as SAS label names. This option is valid only when 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* specifies that the data source column names are not saved as SAS label names. SAS label names are left as null values.

**Default:** COMPAT

### **DEFER= NO | YES**

specifies when a 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**

specifies 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. Specify any of these 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.
- *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

**Restriction:** The NOMULTOUTJOINS option does not affect outer joins of two tables.

**Note:** The NOMULTOUTJOINS option is always set for the Microsoft Jet engine.

#### **FILELOCK= YES | NO**

specifies the access level to a Microsoft Excel file. By default, FILELOCK is not set. Specifying FILELOCK= YES indicates that the LIBNAME engine checks and denies a connection if the file was opened by Excel or another application.

#### **CAUTION:**

**SAS does not check whether the Excel file was opened by Excel or another application when you assign a LIBNAME statement for the file. A potential access violation can occur if a user attempts to update the Excel file using Microsoft Excel.**

*YES* allows only one LIBNAME assignment with READ and WRITE permission to connect to the file. The LIBNAME assignment is denied if the file has been opened by Excel or any other application.

*NO* specifies that the Microsoft Excel LIBNAME engine behaves the same as in SAS 9.1.

By setting FILELOCK= YES with ACCESS= READONLY, this enables other applications to open the Excel file in READONLY mode with connection information set to browse data.

**Restriction:** Applies only to the Microsoft Excel LIBNAME engine; has no effect on the Microsoft Access LIBNAME engine.

#### **INSERT\_SQL= YES | NO**

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

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

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

**Default:** NO

#### **INSERTBUFF= *number-of-rows***

specifies the number of rows for a multi-row insert. If the INSERTBUFF value is greater than the DBCOMMIT value, the DBCOMMIT value overrides the INSERTBUFF value. The value for INSERTBUFF= must be a positive number.

**Default:** 1

**Note:** If you assign a value that is greater than INSERTBUFF= 1, the information written to the log indicating the success or failure might be incorrect. SAS only writes information for a single insert, even when multiple inserts are performed.

#### **MSENGINE= ACE | JET**

specifies the database engine used for accessing the Microsoft Excel file or the Microsoft Access database. The Microsoft Jet engine is older and supports formats up to 2003. The Microsoft Ace engine supports 2007 and later formats.

**Default:** ACE

**Restriction:** It is recommended that you do not use this file option unless you are trying to create a 95 format file.

**PREPARE= YES | NO**

*NO* forces the engine to execute the SQL statement before the Describe action. If you have a linked table to a text file this ensures that the table is found.

**Default:** YES

**READBUFF= *number-of-rows***

specifies the number of rows of data to read. Setting a higher value for this option reduces I/O and increases performance, and memory usage. If too many rows are read at once, values returned to SAS might be out of date.

**Alias:** ROWSET | ROWSET\_SIZE

**Default:** 1

**SCANTEXT= YES | NO**

specifies whether to scan the length of text data for a data source column and use the length of the longest data string as the SAS column width. For Microsoft Excel, this option applies to all character data type columns. For Microsoft Access, this option only applies to the MEMO data type field, it does not apply to the TEXT (less than 256 characters long) field.

*YES* scans the length of text data for a data source column. Sets the length of the longest data string as the SAS variable length. If the maximum SCAN\_TEXT= length is greater than the maximum DBMAX\_TEXT= length, the DBMAX\_TEXT= value is set as the SAS variable length. Default for Microsoft Excel workbook.

*NO* specifies that the column length that is returned from the Microsoft Jet provider is set as the SAS variable length. If the length that is returned from the Microsoft Jet provider is greater than the DBMAX\_TEXT= value, the smaller value is set as the SAS variable length. Specify SCANTEXT= NO when you need to update data in a Microsoft Access database or a Microsoft Excel workbook.

**Alias:** SCAN\_TEXT | SCANMEMO | SCAN\_TEXTSIZE

**SCANTIME= YES | NO | ANY**

specifies whether to scan all row values for a DATETIME data type field to determine the TIME data type based on the setting.

*YES* specifies to scan all row values for a DATETIME data type field to determine the TIME data type based on the setting.

*NO* turns off the scan function.

*ANY* specifies to scan all row values for a DATETIME data type field to determine the TIME data type based on the setting.

**Alias:** SCAN\_TIME | SCAN\_TIMETYPE

**Default:** NO

**Restriction:** available only for Microsoft Windows.

**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. The row set might not be 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.

**Alias:** STRDATES

**Default:** NO

**UNICODE= YES | NO**

determines whether the LIBNAME engine binds the character data type columns with narrow or wide character mode.

This option supersedes the DBENCODING option. For DBCS (Chinese, Korean, and Japanese) use, it is recommended to use this option instead of the DBENCODING= option.

*YES* specifies that SAS binds the character type columns with wide character mode. This allows some character data such as DBCS to be returned correctly. This setting works the same as setting DBENCODING='UTF-16'.

*NO* specifies that SAS binds the character type columns with narrow character mode.

**Default:** NO

**USEDATE= YES | NO**

specifies whether to assign the DATE. format or the DATETIME. format for datetime columns in the data source table while importing data from a Microsoft Access database or a Microsoft Excel workbook.

*YES* specifies that the DATE. format is assigned to datetime columns in the data source table.

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

**Alias:** USE\_DATE | USE\_DATETYPE

**Default:** YES for Microsoft Excel workbooks.

NO for Microsoft Access databases.



## Chapter 12

# Pass-Through Facility for Access and Excel on Microsoft Windows

<b>Pass-Through Facility on Microsoft Windows</b> . . . . .	<b>129</b>
<b>Dictionary</b> . . . . .	<b>130</b>
SQL Procedure . . . . .	130
CONNECT Statement . . . . .	131
DISCONNECT Statement . . . . .	137
EXECUTE Statement . . . . .	137
CONNECTION TO Component . . . . .	139
Microsoft Jet and Microsoft Ace Provider Supported Data Types . . . . .	142
Ace and Jet Special Queries . . . . .	144
Special Jet Commands . . . . .	145

## Pass-Through Facility on Microsoft Windows

The SQL procedure implements the Structured Query Language (SQL) for SAS. See *SAS SQL Procedure User's Guide* for PROC SQL information. 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 on page 97](#). 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.

You can use the pass-through facility for the following tasks:

- Establish and terminate connections with a data source using the pass-through facility CONNECT and DISCONNECT statements.  
See [“CONNECT Statement” on page 131](#) and the [“DISCONNECT Statement” on page 137](#).
- Send dynamic, non-query, data source-specific SQL statements to a data source using the [“CONNECT Statement” on page 131](#) or the [“EXECUTE Statement” on page 137](#).
- Retrieve data directly from a data source using the [“CONNECTION TO Component” on page 139](#) component of the pass-through facility 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 the queries in a PROC SQL view. When you create an 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.

The following sections present the syntax for the pass-through facility statements and the CONNECTION TO component. You can use this component with the PROC SQL SELECT statement to query data from a data source.

---

## Dictionary

---

### SQL Procedure

Implements the Structured Query Language for SAS.

**See:** *Base SAS Procedures Guide*  
*SAS SQL Procedure User's Guide*  
*SAS SQL Query Window User's Guide*

---

### Syntax

**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*

**AS**

<alias> <database-connection-arguments;>

### Without Arguments

Use the CONNECT TO statement with the SQL procedure to query data from a data source.

### Details

#### 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 these 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 can be written to the SAS log using the %PUT macro. The contents are reset after each pass-through facility statement is executed.

### Example: Connect to an Excel 2007 .xlsx File and Query the INVOICE Table (range) within the Excel Workbook

```
PROC SQL DQUOTE=ANSI;
CONNECT TO EXCEL (PATH='c:\sasdemo\sasdemo.xlsx');
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 statement

---

### Syntax

**CONNECT TO** *data-source-name*

**CONNECT TO** *option(s)*

### Optional Arguments

#### **data-source-name**

Specifies the data source to which you want to connect. Because this method requires connecting through a 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 connections, unique connections, and so on) to the database.

#### **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. The default behavior opens a dialog box with prompts to specify connection information.

## Details

### Overview

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. See “[DISCONNECT Statement](#)” on page 137.

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 CONNECT statement with the PC files engine name and then assign an alias if you want.
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 you do not specify it, default values for all 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 130 for more information about these macro variables.

### Database Statement Arguments

The arguments that are listed below are available with the pass-through facility for PC files. These arguments provide information to the pass-through facility to connect to the PC files or to the database. These options are used when connecting to PC Files Server.

#### **DSN=** *data-source-name*

specifies the ODBC data source name that is used to access the PC data through 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 semicolons. This is an advanced connection method that you should use only when you know the exact syntax of all connection options that the ODBC driver requires for a successful connection.

#### **PATH=** *path-for-file*

specifies the data source file location for the Microsoft Access database file or Microsoft Excel workbook file.

#### **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.

**Alias:** SERVICE | SERVICE\_NAME

**Default:** 9621

#### **SERVER=** *pc-server-host-name*

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.

You can specify this host name as a simple computer name (for example, wxp320), a fully qualified network name (for example, **wxp320.domain.com**), or an IP address.

**Note:** Omitting the SERVER= option on Microsoft Windows clients invokes Autostart.

**SERVERUSER= *'domain\server-user-name'***

specifies the domain name and User ID for the PC running PC Files Server. Always enclose the value in quotation marks. Otherwise, the backslash can be misinterpreted by the SAS parser.

**Alias:** SERVERUID

**Notes:**

If you are not on a domain, omit the domain name and the backslash.

Use the USER= option for database user IDs.

**SERVERPASS= *'server-user-password'***

specifies the password for the PC Files Server for the user ID given. If the account has no password, omit this option. Always enclose the value in quotes in order to preserve the case of the password.

**Alias:** SERVERPASSWORD | SERVERPW | SERVERPWD

**Notes:**

Passwords are generally case sensitive.

Use the PASSWORD= option for database passwords.

**SSPI= *YES* | *NO***

enables the PC Files Server to allow Integrated Windows Authentication. This is a mechanism for the Windows client and server to exchange credentials.

**Default:** NO

**Restriction:** Microsoft Windows 64-Bit only.

**Note:** SSPI can also be enabled by specifying the `-SSPI` option on the SAS command line.

**DBPASSWORD= *database-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 you can define it instead of user-level security.

**Restriction:** Microsoft Access only.

**DBSYSFILE= *workgroup-information-file***

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

**PASSWORD= *user-password***

specifies a password for the user account, if required by the data source. Passwords are case sensitive.

**MSENGINE= *ACE* | *JET***

determines the database engine used for accessing the Microsoft Excel file or Microsoft Access database. The Microsoft Jet engine is older and supports formats up to 2003. The Microsoft ACE engine supports Microsoft Excel 2007 and Microsoft Access 2007 and older formats.

**Default:** ACE

**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 access your file.

**VERSION= 2007 | 2003 | 2002 | 2000 | 97 | 95 | 5**

sets the version of Microsoft Excel workbook. The default value is 97.

**Alias:** VER

**Restriction:** Microsoft Excel only.

**Note:** You do not need to specify this option for an existing Microsoft Excel file. If you want to create a new Microsoft Excel workbook file, you can use this option to specify the version that you want to create. Note that versions 97, 2000, and 2003 of Excel share the same file format. Versions 95 and 5 share a separate file format.

**CONNECT Statement Arguments**

Connect Statement arguments are supported by 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 they are submitted.

**YES**

specifies that updates are committed (saved) to the table as soon as they are submitted. 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 before a data source command times out.

**Alias:** TIMEOUT

**Default:** 0 (no time-out)

**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.

- The CONNECT statements must use identical values for the CONNECTION= option.

- The CONNECT statement must use identical values for the CONNECTION\_GROUP= option.
- Database connection arguments must be identical.

**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=** *DYNAMIC* | *FORWARD\_ONLY* | *KEYSET\_DRIVEN* | *STATIC*

specifies the cursor type for read-only cursors and for cursors to be updated.

**DYNAMIC**

specifies that the cursor reflects all 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 supports only 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.

**Alias:** CURSOR

**Default:** None

**DBGEN\_NAME=** *DBMS* | *SAS*

specifies that the data source columns are renamed, and specifies the format that the new names 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 applies only when you are reading, appending, and updating Microsoft Access or Excel character data from SAS.

**Default:** 1,024

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

**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

**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. In addition, if too many rows are read at once, values returned to SAS might be out of date.

**Alias:**

ROWSET=

ROWSET\_SIZE=

**Default:** 1

**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.

**Alias:** STRDATES

**Default:** NO

**USEDATE= YES | NO**

specifies whether to assign the DATE. format or the DATETIME. format for datetime columns in the data source table while importing data from a Microsoft Access database or a Microsoft Excel workbook.

YES

specifies that the DATE. format is assigned to datetime columns in the data source table.

NO

specifies that the DATETIME. format is assigned for datetime columns in the data source table.

**Alias:** USE\_DATE | USE\_DATETYPE

**Default:** NO

**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

Ends the connection to the data source.

**Valid in:** SQL procedure.

---

### Syntax

```
DISCONNECT FROM <data-source-name> <alias>
```

### Syntax Description

#### *Data-source-name*

specifies the *data-source-name* from which you want to disconnect. The DISCONNECT statement's *data-source-name* must match the *data-source-name* that you specified in the CONNECT statement.

#### *Alias*

specifies the data source *alias* from which you want to disconnect. The DISCONNECT statement's *alias* must match the *alias* that you specified 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 the procedure ends. The SQL procedure continues to execute until you submit a QUIT statement, a SAS procedure, or a DATA step.

The contents of the SQLXRC and SQLXMSG macro variables can be written to the SAS log using the macro. The contents are reset after each pass-through facility statement is executed.

See [“Return Codes” on page 130](#) for additional information.

### Example: Disconnect and Quit

SQL processing uses the DISCONNECT statement to end the connection with the database. Use the QUIT statement to quit the SQL procedure after the connection ends:

```
DISCONNECT FROM db;  
QUIT;
```

---

## EXECUTE Statement

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

**Valid in:** SQL procedure steps.

---

## Syntax

```
EXECUTE (data-source-specific-SQL-statement(s))
  BY
  <data-source-name> <alias>
```

### Syntax Description

#### *Data-source-specific-SQL-statements*

specifies a dynamic non-query, data source-specific SQL statement. Depending on your data source, the SQL statement can be case sensitive. The statement is passed to the data source exactly as you type it.

#### 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

drops 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

inserts rows to a data source table.

#### REVOKE

revokes the access or modification privileges that were given to users by the GRANT statement.

#### UPDATE

updates the data in the specified columns of a row in a data source table.

#### Requirements:

At least one statement is required.

The statement must be enclosed in parentheses.

#### *Data-source-name*

specifies the *data-source-name* to which you direct the data source-specific SQL statements. The EXECUTE statement's *data-source-name* must match the *data-source-name* specified in the CONNECT statement.

#### *Alias*

specifies the data source *alias* that was defined in the CONNECT statement. The EXECUTE statement's *alias* must match the *alias* that you specified in the CONNECT statement.

## Details

The EXECUTE statement sends dynamic non-query, 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 an SQL view.

The contents of the SQLXRC and SQLXMSG macro variables can be written to the SAS log using the macros. The contents are reset after each pass-through facility statement is executed.

## Example: Drop and Create a Table and Insert a Data Row

Use the EXECUTE statement to drop a table, create a table, and insert a row of data after the connection:

```
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:** SQL procedure STEP statements.

---

### Syntax

**CONNECT TO** a *data-source* **AS** *alias* (*connect statement arguments*)  
(*database connection-options*)

### Summary of Optional Arguments

[ALIAS](#)

[DATABASE CONNECTION ARGUMENTS](#)

[DATA SOURCE NAME](#)

[CONNECTION COMPONENT](#)

### Optional Arguments

#### ALIAS

specifies the data source alias for the connection. If you specify an alias, the keyword AS must appear before the alias.

**Restriction:** ALIAS is not supported if the CONNECT statement is omitted.

**Requirement:** The range of the ALIAS is between 1 and 32 characters.

**Note:** The data source name is used as the name of the pass-through connection if an alias is not specified.

#### CONNECTION COMPONENT

specifies arguments that indicate whether you can make multiple connections, shared connections, or unique connections, to the database.

#### DATA SOURCE NAME

specifies the data source name to which you want to connect and direct the data source-specific SQL statements.

**Requirement:** You must use back quotation marks ( ` ), not single (forward) quotation marks, to enclose any data source name that contains a space.

**Note:** The data source name becomes the name of the pass-through connection if an alias is not specified.

### DATABASE CONNECTION ARGUMENTS

specifies the data source-specific arguments to the pass-through facility that are needed by the SQL procedure to connect to a data source.

#### **Database Connection Arguments**

Connection arguments provide database connection information to the pass-through facility to connect to a Microsoft Access database or a Microsoft Excel workbook file.

#### **INIT= "initialization-string"**

specifies the initialization string when connecting to a data source.

**Note:** This statement option applies to the INIT= option and the UDL= option.

#### **MSENGINE= ACE | JET**

determines the database engine to use for accessing Microsoft Excel files or Microsoft Access databases. The Microsoft Jet engine supports Microsoft formats up to 2003. The Microsoft Ace engine supports 2007 formats and formats in subsequent releases of Windows.

**Default:** ACE

**Restriction:** It is recommended that this option is used to create only a Windows 95 format file.

#### **PATH= data-source-path**

specifies the path of the Microsoft Access database or the Microsoft Excel workbook file.

#### **PROMPT= YES | NO | REQUIRED | NOPROMPT | PROMPT | UDL**

specifies whether user is prompted for data source connection information.

YES enables prompting with a Data Link Properties dialog box. To write the initialization string to the SAS log, submit this code immediately after connecting to the data source:

```
%PUT %SUPERQ (SYSDBMSG) ;
```

NO prompting is not available. You must specify the data source as a physical filename or complete path.

REQUIRED connect with a valid data-source-name. If a valid connection is not specified, you are prompted for the connection options. The prompt enables you to change the data source file and other properties.

NOPROMPT disables the display of the Data Link Properties window. Prompting is not available.

PROMPT enables the display of the Data Link Properties window. Prompting is available.

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

*Note:* This statement option applies to the INIT= argument and the UDL= argument.

#### **UDL= "path and filename"**

specifies the path and filename for a UDL (a Microsoft data link file). This option does not support SAS filerefs. The macro variable SYSDBMSG is set upon successful completion.

```
UDL_FILE= 'C:\WinNT\profiles\me\desktop\MyDBLink.udl' ;
%PUT %SUPERQ (SYSDBMSG) ;
```

**Alias:** UDL\_FILE

**See:** Microsoft Data Link API documentation.

### ***Additional Options for Microsoft Access Database Only***

**DBPASSWORD=** *database-file-password*

enables you to access database files with database-level security. This security level can be defined instead of user-level security.

**Alias:** DBPWD | DBPW | PASS | PASSWORD

**Restriction:** Microsoft Access Database only.

**Note:** Database password is case sensitive.

**DBSYSFILE=** *workgroup-information-file*

specifies the workgroup information file. This file contains a collection of information defined for the Microsoft Access database. User, group accounts, and passwords that you create are saved in the workgroup information file.

**Alias:** SYSTEMDB

**Restriction:** Microsoft Access database files only.

**PASSWORD=** *user-password*

specifies a password required by the data source for the user account.

**Alias:** PWD | PW | PASS | PASSWORD

**Note:** Passwords are case sensitive.

**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 on your .mdb file, you need to use the USER= and PASSWORD= options to access your file.

**Alias:** UID

**Restriction:** Microsoft Access database files only.

**Note:** Use the SERVERUSER= option to connect to a server.

## **Details**

The CONNECTION component specifies the data source connection to use or to create. CONNECTION enables you to retrieve data source data directly through an SQL procedure query.

- The CONNECTION component can be used in any FROM clause, including those in nested queries (subqueries).
- You can store a pass-through facility query in an SQL view and then use that view in SAS programs.
- When you create an SQL view, any options that you specify in the corresponding CONNECTION statement are stored too. Thus, when the 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 component.

## **Example: Connect and Query a Table**

Use the CONNECTION component to query a table or a subtable after the connection:

```
SELECT * FROM CONNECTION TO db(SELECT * FROM `my invoice`);
SELECT * FROM CONNECTION TO db
(SELECT `Invoice Number`, Amount from `my invoice`);
```

## Microsoft Jet and Microsoft Ace Provider Supported Data Types

Valid data types that are supported by Jet provider.

**Valid in:** CREATE statements.

### Details

You can use these data types when you use the CREATE statement using the SQL pass-through facility to create a table. You can also use the data types when you use data set option DBTYPE to change the data type for a loaded column.

**Table 12.1** Microsoft Jet Provider Supported Data Types

Data Type	Column Size	Create Parameters	Prefix and Suffix	Comments
BIT	2			
BYTE	3			
SHORT	5			
LONG	10			
SINGLE	7			
DOUBLE	15			
DECIMAL	28	precision,scale		*
COUNTER	10			**
GUID	16			
CURRENCY	19			
DATETIME	8		#	***
VARCHAR	255	max length		†
LONGTEXT	536,870,910			††
VARBINARY	255			
BIGBINARY	4000			

Data Type	Column Size	Create Parameters	Prefix and Suffix	Comments
LONGBINARY	1,073,741,823			††

- \* When you use the data type DECIMAL, you can specify precision and scale.
- \*\* When you use the DBTYPE option, the COUNTER data type is valid only when you set INSERT SQL=YES. The COUNTER data type is supported only in the pass-through facility when you create a table.
- \*\*\* When you use the pass-through facility to set a date/time value, you must add the prefix and suffix, #. For example, #01/01/2001# and #03/12/1999 12:12:12#.
- † When you use the data types VARCHAR or VARBINARY, you must specify the maximum length.
- †† All Excel columns can have a null value. Do not specify NOTNULL for a column when you create an Excel table. You can specify NULL or NULL attributes for a field when you create a Microsoft Excel table.

## Examples

### **Example 1: Connect to Excel and Drop and Create a Table**

Connects to Microsoft Excel, drops the DEMO table, and creates a new table.

```
PROC SQL;
CONNECT TO EXCEL AS db (PATH='c:\temp\demo.xls'filelock=yes);
EXECUTE(DROP table demo) BY db;
EXECUTE(CREATE TABLE demo(EmpID long, FirstName char(10),
Salary decimal(10,2), hiredate datetime)) BY db;
EXECUTE(INSERT INTO demo values(12345678, 'Michael',
123456.78, #07/01/2001#)) BY db;
EXECUTE(INSERT INTO demo values(23456789, 'Howard', 234567.89,
#04/01/1983#)) BY db;
EXECUTE(INSERT INTO demo values(34567890, 'Nancy', null,
#02/01/1982#)) BY db;
EXECUTE(INSERT INTO demo values(34567890, 'Andy', 456789.01,
null)) BY DB;
SELECT * FROM CONNECTION TO DB(SELECT * FROM demo);
DISCONNECT FROM DB;
QUIT;
```

### **Example 2: Connect to Access and Drop and Create a Table**

Connects to Microsoft Access, drops the DEMO table, and creates a new table.

```
PROC SQL;
CONNECT TO Access AS db (PATH='c:\temp\demo.mdb');
EXECUTE(DROP table demo) BY db;
EXECUTE(CREATE table demo(EmpID long not null,
FirstName char(10) not null,
Salary decimal(10,2), hiredate datetime)) BY db;
EXECUTE(INSERT INTO demo values(12345678, 'Michael',
123456.78, #07/01/2001#)) BY db;
EXECUTE(INSERT INTO demo values(23456789, 'Howard', 234567.89,
#04/01/1983#)) BY db;
EXECUTE(INSERT INTO demo values(34567890, 'Nancy', null,
#02/01/1982#)) BY db;
EXECUTE(INSERT INTO demo values(34567890, 'Andy', 456789.01,
null)) BY db;
SELECT * FROM CONNECTION TO db(SELECT * FROM demo);
```

```
DISCONNECT FROM db;
QUIT;
```

---

## Ace and Jet Special Queries

Queries that return information.

---

### Syntax

**ACE | JET :** *schema-rowset*'*parameter-1*, ..., *parameter-n*

### Required Arguments

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

**ACE | JET :**

required to distinguish special queries from regular queries.

*schema-rowset*

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

**"*parameter-1*, ..., *parameter-n*"**

- Separate parameters from one another by commas.
- All parameters are optional.
- Quotation marks are required.
- If you specify some, but not all, parameters within an argument, use commas to indicate the omitted parameters.

### Details

**ACE|JET::CHECK\_CONSTRAINTS**

returns the check constraints that are defined in the database file.

**ACE|JET::COLUMNS**<"*table-name*"> , <"*column-name*">

returns the columns of the tables that are defined in the database file.

**ACE|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.

**ACE|JET::FOREIGN\_KEYS**<"*primary-key-table-name*"> , <"*foreign-key-table-name*">

returns the foreign key columns that are defined in the database file.

**ACE|JET::INDEXES**<"*index-name*"> , <"*table-name*">

returns the indexes that are defined in the database file.

**ACE|JET::KEY\_COLUMN\_USAGE**<"*constraint-name*"> , <"*table-name*"> , <"*column-name*">

returns the key columns that are defined in the database file.

**ACE|JET::PRIMARY\_KEYS**<"*table-name*">

returns the primary key columns that are defined in the database file.

**ACE|JET::PROCEDURES** <"*procedure-name*">

returns the procedures that are defined in the database file.

**ACE|JET::PROVIDER\_TYPES**

returns information about the base data types that are supported by the Jet data provider.

**ACE|JET::REFERENTIAL\_CONSTRAINTS** <"constraint-name">

returns the referential constraints that are defined in the database file.

**ACE|JET::STATISTICS** <"table-name">

returns the statistics that are defined in the database file.

**ACE|JET::TABLE\_CONSTRAINTS** <"constraint-name"> , <"table-name"> , <"constraint-type">

returns the table constraints that are defined in the database file.

**ACE|JET::TABLES** <"table-name"> , <"table-type">

returns the tables that are defined in the database file.

**ACE|JET::VIEWS** <"table-name">

returns the viewed tables that are defined in the database file.

## Examples

### Example 1: Retrieve a Specific Rowset

Retrieve a rowset that displays all tables in the NorthWind database.

```
PROC SQL;
  * CONNECT TO access database;
  CONNECT TO Access AS db (PATH='c:\NorthWind.mdb');
  * list all tables including system tables and pass-through;
  SELECT * FROM CONNECTION TO db(jet::tables);
  * list table name and type where table type is TABLE only;
  SELECT table_name, table_type from CONNECTION TO db(jet::tables , "TABLE");
  DISCONNECT FROM db;`
QUIT;
```

### Example 2: Retrieve Specific Data Types

Retrieve all data types that the Ace or Jet provider for Microsoft Access supports.

```
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. Here is the general format of special commands.

---

### Possible Values

JET::COMMAND	general format
JET::	required to distinguish special queries from regular queries.
JET::COMMIT	to commit a transaction

JET::ROLLBACK	to cause a rollback in the transaction
JET::AUTOCOMMIT	to set the COMMIT mode to AUTO and commit the transaction immediately
JET::NOAUTOCOMMIT	to set 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.

## Example: Syntax Examples

Although these examples are for Microsoft Access, the syntax is the same for Microsoft Excel.

### **Example Code 12.1** *AUTOCOMMIT with the NO Connection Option.*

```

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 13

# File-Specific Reference for Access and Excel on Microsoft Windows

---

<b>Microsoft Excel Workbook Files</b> .....	<b>147</b>
LIBNAME Statement .....	147
Connection Options .....	148
Data Types Conversion .....	150
Processing Date and Time Values between SAS and Microsoft Excel .....	153
<b>Microsoft Access Files</b> .....	<b>153</b>
Statements .....	153
Connection Options .....	153
Data Types Conversion .....	154
Processing Date and Time Values between SAS and Microsoft Access .....	157

---

## Microsoft Excel Workbook Files

### *LIBNAME Statement*

By default, the SAS LIBNAME statement connects to a Microsoft Excel file in limited READ/WRITE mode. Although you can read data, delete a table, or create a new table, you cannot update data or append a new data row. To allow data update and append, set the LIBNAME option SCANTEXT=NO.

Because the SAS PROC SQL pass-through connects to a Microsoft Excel file in READ/WRITE mode. You can read, write, and update data by passing SQL command statements.

#### **CAUTION:**

**Due to the use of the Microsoft Jet Excel engine and the Microsoft Ace Excel engine, the SAS engines for Excel have limited update and delete capability. There might be other unsolved issues. You should therefore avoid using the update and delete features. Back up your Excel files before you try using any update functions.**

#### **CAUTION:**

**Although you can connect to an Excel file while the application is opening it, any updates to the Excel file can cause the SAS Excel engine to malfunction. It is recommended that you close the application that is using the Excel file, disconnect, and then reconnect the file in SAS. Set the LIBNAME option, FILELOCK=YES, to ensure that Excel or other applications have not opened the connected Excel file.**

## Connection Options

You can use this connection option in the LIBNAME statement or in the PROC SQL CONNECT statement to connect to a Microsoft Excel file.

### **HEADER= YES | NO**

determines whether the first row of data in a Microsoft Excel range (or spreadsheet) are column names.

**YES**

specifies to use the first row of data in an Excel range (or spreadsheet) as column names.

**NO**

specifies not to use the first row of data as column names in an Excel range (or spreadsheet). SAS generates and uses the variable names F1, F2, F3, and so on.

**Alias:** HDR | GETNAMES

**Default:** YES

**Note:** This connection option is only for reading Microsoft Excel spreadsheets. This option is ignored when you are writing data to an Excel spreadsheet.

### **MIXED= YES | NO**

specifies whether to convert numeric data values into character data values for a column of mixed data types. This option is valid only when importing (reading) data from Excel. The Microsoft Ace and Jet Excel engines handle this option.

**YES**

assigns a SAS character type for the column and converts all numeric data values to character data values when mixed data types are found. When you specify MIXED=YES, the connection is set in import mode and no updates are allowed.

**CAUTION:**

**Due to a limitation in the Microsoft Ace and Microsoft Jet engines, MIXED=YES could result in improper text variable lengths.**

**NO**

assigns numeric or character type for the column, depending on the majority of the type data that is found. Both numeric data in a character column and character data in a numeric column are imported as missing values.

**Default:** NO

**Restrictions:**

DBMS= only

This option is available only for Windows reading Excel data into SAS. You cannot use this option for delimited files.

Registry settings might affect the behavior of the MIXED= option. Refer to the following tables for Registry Key location and Registry Settings for the MIXED= option for additional information.

**Table 13.1** Registry Key for TypeGuessRow Based on Office Version or Engine

Environment	Office Version or Engine	Registry Key
Windows	Office 2007	<b>HKEY_LOCAL_MACHINE</b> ⇒ <b>Software</b> ⇒ <b>Microsoft</b> ⇒ <b>Office</b> ⇒ <b>12.0</b> ⇒ <b>Access Connectivity Engine</b> ⇒ <b>Engines</b> ⇒ <b>Excel</b>
Windows running 9.2 TS2M0 or later	Office 2010	<b>HKEY_LOCAL_MACHINE</b> ⇒ <b>Software</b> ⇒ <b>Microsoft</b> ⇒ <b>Office</b> ⇒ <b>14.0</b> ⇒ <b>Access Connectivity Engine</b> ⇒ <b>Engines</b> ⇒ <b>Excel</b>
Windows 7 or X64 system	Office 2007	<b>HKEY_LOCAL_MACHINE</b> ⇒ <b>Software</b> ⇒ <b>Wow6432Node</b> ⇒ <b>Microsoft</b> ⇒ <b>Office</b> ⇒ <b>12.0</b> ⇒ <b>Access Connectivity Engine</b> ⇒ <b>Engines</b> ⇒ <b>Excel</b>
Windows 7 or X64 system	Office 2010 32-bit	<b>HKEY_LOCAL_MACHINE</b> ⇒ <b>Software</b> ⇒ <b>Wow6432Node</b> ⇒ <b>Microsoft</b> ⇒ <b>Office</b> ⇒ <b>14.0</b> ⇒ <b>Access Connectivity Engine</b> ⇒ <b>Engines</b> ⇒ <b>Excel</b>
Windows 7 or X64 system	Office 2010 64-bit	<b>HKEY_LOCAL_MACHINE</b> ⇒ <b>Software</b> ⇒ <b>Microsoft</b> ⇒ <b>Office</b> ⇒ <b>14.0</b> ⇒ <b>Access Connectivity Engine</b> ⇒ <b>Engines</b> ⇒ <b>Excel</b>
Windows	ACE Engine	<b>HKEY_LOCAL_MACHINE</b> ⇒ <b>Software</b> ⇒ <b>Microsoft</b> ⇒ <b>Office</b> ⇒ <b>12.0</b> ⇒ <b>Access Connectivity Engine</b> ⇒ <b>Engines</b> ⇒ <b>Excel</b>
Windows	Jet Engine (when using MENGINE=JET only)	<b>HKEY_LOCAL_MACHINE</b> ⇒ <b>Software</b> ⇒ <b>Microsoft</b> ⇒ <b>Jet</b> ⇒ <b>4.0</b> ⇒ <b>Engines</b> ⇒ <b>Excel</b>

**Table 13.2** Registry Settings for the MIXED Option

TypeGuessRows	<p>An integer type with a default value of 8. You can use the number of rows in the worksheet range in scans to determine column types. If you set this type to 0, all rows in the range are checked. Microsoft states that the valid range of TypeGuessRows is 0–16. However, you could set as high as 16384 and it would still operate correctly.</p> <p><b>CAUTION</b></p> <p>Changing the TypeGuessRows value cause a scan to fail if you set it the value higher than 16384. Changes also affect any software that uses the Microsoft Jet provider to access Excel file data, including accessing Excel data in a Microsoft Access database. The TypeGuessRows value is registered with and controlled by Microsoft. It is therefore recommended that you set the value to 0</p>
---------------	---

---

ImportMixedTypes	<p>A string type with a default value of Text. While scanning TypeGuessRows rows, if a column has multiple data types, the column type is Text. This is only if the value of the setting is Text. If the value of the setting is Majority, the most common data type in the column is set as the column type.</p> <p>For the MIXED=YES option to work correctly, you should you change TypeGuessingRows to 0 in the Microsoft Windows registry so that all rows in the specified range are scanned. As a result, when you use MIXED=YES, the Jet provider always assigns character type for columns with data of mixed data types and converts numeric data to character data.</p>
------------------	--

---

**TypeGuessRows**

an integer type with a default value of 8. The number of rows in the worksheet range is used to scan and determine column types. If set to 0, then all rows in the range are checked. Microsoft states that the valid range of TypeGuessRows is 0 to 16. However, it can be set as high as 16,384 and still operate correctly.

**ImportMixedTypes**

a string type with a default value of Text. If a column contains more than one type of data (during the scan of TypeGuessRows rows), the column type is Text if the value is Text. If the value is Majority, the most common type determines the column type.

**CAUTION:**

**These settings are registered by the Microsoft Jet engine. Changing settings such as TypeGuessRows for Microsoft Jet engine in the Windows registry affects all software that uses Microsoft Jet engine to import data. This includes Microsoft Office products, as well as other database products and application software.**

*Note:* For the MIXED=YES option to work correctly, change TypeGuessingRows to 0 in the Windows registry. All rows in the specified range are scanned. Now when you use MIXED=YES, the Jet provider always assigns character type for columns with mixed data types. It also converts numeric data to character data.

**VERSION= '2007'|'2003'|'2002'|'2000'|'97'|'95'|'5'**

specifies the version of the file that you want to create if the file does not exist. Valid values are 2003, 2002, 2000, 97, 95, and 5. The default value is 97.

**Alias:** VER

**Notes:**

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 that you create. There is no need to specify the VERSION= value if you want to create a 2007 Excel .xlsb or .xlsx file. The file extension tells SAS the VERSION= value.

Versions 2003, 2002, 2000, and 97 are treated as the same format. Versions 95 and 5 share the same format.

**Data Types Conversion**

This table shows the default SAS variable formats that SAS/ACCESS assigns to Excel data types. The data formats are assigned when SAS reads data from Microsoft Excel spreadsheets using the LIBNAME engine.

**Table 13.3** Default SAS Formats Assigned for Excel Formats

Excel Column Format	SAS Variable Format	SAS Variable Type
Text	\$w.	Character
General		Numeric
Number		Numeric
Scientific		Numeric
Percentage	See Note ***	Numeric
Fraction	See Note ***	Numeric
Currency	DOLLAR21.2	Numeric
Accounting	DOLLAR21.2	Numeric
Date	DATE9. See Note ***	Numeric
Time	TIME8.***	Numeric

\* The default format is DATE9. You can use USEDATE=NO to change format from DATE9. to DATETIME. You can also use the SASDATEFMT= option to change the format of other SAS date and time formats.

\*\* Note that the SAS date time value uses 01Jan1960 as a cutoff line, while the Jet provider date value uses 30Dec1899 as a cutoff line for internal values.

\*\*\* 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.

The following table shows the default Excel data types that SAS/ACCESS assigns to SAS variable formats. These data types are assigned when SAS writes data to an Excel file using the LIBNAME engine. You can override these default conversions by using “DBTYPE” on page 114 during output processing.

**Table 13.4** Default Excel Formats Assigned for SAS Variable Formats

SAS Variable Format	XLS Column Data Type
\$BINARYw.	Text
\$CHARw.	Text
\$HEX w.	Text
\$w.	Text
w.d	Number
BESTw.	Number

<b>SAS Variable Format</b>	<b>XLS Column Data Type</b>
BINARYw.	Number
COMMA w.d	Number
COMMAXw.d	Number
Ew.	Number
FRACTw.	Number
HEXw.	Number
NEGPARENw.d	Number
PERCENTw.d	Number
DOLLARw.d	Currency
DOLLARXw.d	Currency
DATEw.	Date
DATETIMEw.d	Date
DDMMYYw.	Date
HHMMw.d	Time
JULDAYw.	Date
JULIANw.	Date
MMDDYYw.	Date
MMYYw.d	Date
MONTHw.	Date
MOYYw.	Date
WEEKDATEw.	Date
WEEKDATXw.	Date
WEEKDAYw.	Date
WORDDATEw.	Date
WORDDATXw.	Date

## Processing Date and Time Values between SAS and Microsoft Excel

To import date or time values from a Microsoft Excel file, the SAS LIBNAME engine reads date values using DATE9. by default. Time values are assigned the TIME8. format. However, you can set the LIBNAME option, USEDATE=NO, or the LIBNAME statement, USEDATE=NO. Using the IMPORT procedure, you can have date and time values read in using the DATETIME. format.

To export SAS values with DATE, TIME, or DATETIME formats to a Microsoft Excel file, values are written using DATE9. format. When you see values with the date 1/0/1900 in Microsoft Excel, format them using the TIME format to display the correct time values.

TIME\_VAL: By default, the SAS/ACCESS LIBNAME engine loads the SAS time value in the ACCESS database. The time value is set to less than or equal to 24 hours. Set the time environment variables follows:

```
OPTIONS SET=TIME_VAL SAS;
```

To load SAS time values with the SAS date and time base of 01Jan1960:00:00:00, enable the time value to be reserved for longer than one day. When you import data with SCANTIME=YES, SAS scans date and time values in the column and assigns the TIME8. format if all the values in the column are in the year 1960. If this is not the case, SAS assigns the DATETIME format for the column. You can reset the TIME\_VAL back to the default value with the following code:

```
OPTIONS SET=TIME_VAL_ONEDAY;
```

## Microsoft Access Files

### Statements

The SAS/ACCESS LIBNAME statement connects to a Microsoft Access database MDB file in READ/WRITE mode by default. The SAS CONNECT statement in PROC SQL also connects in READ/WRITE mode by default. Set the LIBNAME option ACCESS=READONLY to connect to the file in READONLY mode.

The SAS engine for the Microsoft Access database uses the Microsoft Jet Provider to connect to Microsoft Access data in the MDB file. A connection requires an existing MDB file. The engine knows the version of the MDB file that the Microsoft Access database saves.

The engine supports some special queries. The queries return information such as available tables, primary keys, foreign keys, and indexes. For more information, see [“Pass-Through Facility on Microsoft Windows” on page 129](#).

### Connection Options

These connection options can be used in the LIBNAME statement or in the PROC SQL statement.

*Note:* You can use USER, PASSWORD, DBPASSWORD, and DBSYSFILE to access your .mdb files, but it does not change your current security settings for those 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 can be defined in addition to user-level security. Do not include ampersands, quotation marks, or invisible characters in your password.

**Alias:** DBPWD | DBPW

**DBSYSFILE= 'workgroup-information-file'**

contains information about the users in a workgroup based on information that defines your Microsoft Access database. Any user and group accounts or passwords that you create are saved in the workgroup information file.

**Alias:** DBSYS | WGDB

**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. Do not include ampersands, quotation marks, or invisible characters in your password.

**Alias:** PWD | PW

**Note:** If you have user-level security set in your .mdb file, you need to use this option and the USER option to be able to access your file.

**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.

**Alias:** UID | USERID

**Note:** 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.

## Data Types Conversion

The following table shows the default SAS variable formats that SAS/ACCESS assigns to .mdb data types. These formats are assigned when SAS reads data from Microsoft Access files using the LIBNAME engine.

**Table 13.5** Default SAS Formats Assigned for MDB Data Types

MDB Field Data Type	SAS Variable Format	SAS Variable Type
Yes No	2.	Numeric
Number (FieldSize=Byte)	4.	Numeric
Number (FieldSize=Decimal)	w.d <sup>††</sup>	Numeric
Number (FieldSize=Integer)	6.	Numeric
Number (FieldSize=Long Integer)	11.	Numeric

MDB Field Data Type	SAS Variable Format	SAS Variable Type
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	DATETIME. ***	Numeric
Text	\$w. ***	Character
Memo	\$w. †	Character
OLE Object	\$w. †	Character
Hyperlink	\$w. †	Character

\* The default format is DATETIME. You can use USEDATE=YES to change format from DATETIME. to DATE. You can also use the SASDATEFMT option to change the format to other date or date and time formats.

\*\* The SAS date/time value uses 01Jan1960 as the cutoff date. The Jet provider date/ time value uses 30Dec1899 as the cutoff date.

\*\*\* The width of \$w. is equal to the field size of the column defined in your Microsoft Access table.

† When SCANMEMO=YES (default is NO), the width value of \$w. is determined by the longest string of data that is scanned in the field. It can also be determined 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 w width value is equal to the precision value plus 1. The d decimal value is equal to the scale value, where precision and scale are defined for the column in the table.

The following table shows the default .mdb data types that SAS/ACCESS assigns to SAS variable formats. These data types are assigned when you write SAS data to an .mdb file using the LIBNAME engine. You can override these default conversions by using the DBTYPE data set option during processing.

**Table 13.6** Default DB Data Types Assigned for SAS Variable Formats

SAS Variable Format	MDB Data Type
\$BINARYw.	Text (VarChar) or Memo (LongText) *
\$CHARw.	Text (VarChar) or Memo (LongText)*
HEX w.	Text (VarChar) or Memo (LongText) *

SAS Variable Format	MDB Data Type
\$w.	Text (VarChar) or Memo (LongText)*
w.d	Number **
BESTw.	Number **
BINARYw.	Number **
COMMA w.d	Number **
COMMAXw.d	Number **
Ew.	Number **
FRACTw.	Number **
HEXw.	Number **
NEGPARENw.d	Number **
PERCENTw.d	Number **
DOLLARw.d	Currency
DOLLARXw.d	Currency
DATEw.	Date/Time
DATETIMEw.d	Date/Time
DDMMYYw.	Date/Time
HHMMw.d	Date/Time
JULDAYw.	Date/Time
JULIANw.	Date/Time
MMDDYYw.	Date/Time
MMYYw.d	Date/Time
MONTHw.	Date/Time
MOYYw.	Date/Time
WEEKDATEw.	Date/Time
WEEKDATXw.	Date/Time
WEEKDAYw.	Date/Time

SAS Variable Format	MDB Data Type
WORDDATEw.	Date/Time
WORDDATXw.	Date/Time

\* If the character format length is greater than 255 characters, the loaded format is Memo. Otherwise, the loaded format is Text.

\*\* For Microsoft Access 2000, 2002, and 2003, 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 is a number with a long integer field size. If the format is specified as w.d in SAS, the loaded data type is a number data type with a decimal field size. For Microsoft Access 97, if the format is specified as w. in SAS, the loaded data type is a number with a long integer field size. Otherwise, the SAS numeric data type is converted to a number data type with a double field size. However, you can set the SAS environment variable, LOAD\_DBL=YES, to force a SAS numeric data type to be loaded into a numeric data type with a Double field size.

#### Example:

```
DATA test;
  FORMAT j 5. k 6.2;
  i=123.45; j=12345; k=123.45;
RUN;
```

```
/* The following PROC loads the Test1 table, which contains
  Column i with a Double field size,
  Column j with a Long Integer field size,
  and Column k with a Decimal field size. */
```

```
PROC EXPORT DATA=test
  OUTTABLE= 'Test1'
  DBMS=ACCESS REPLACE;
  DATABASE='c:\temp\test.mdb';
RUN;
```

```
/* The following PROC loads the Test2 table, which contains
  Columns i, j, and k, all of which have a Double field size. */
```

```
OPTIONS SET=load_dbl yes;
PROC EXPORT DATA=test
  OUTTABLE= 'Test2'
  DBMS=ACCESS REPLACE;
  DATABASE='c:\temp\test.mdb';
RUN;
```

## Processing Date and Time Values between SAS and Microsoft Access

### Date and Time Value Import and Export

To import date and time values from a Microsoft Access database, the SAS/ACCESS LIBNAME engine reads in the date using the DATE9. format. It reads in the time values using TIME8. However, you can set the LIBNAME option, USEDATE=YES, or the LIBNAME statement, USEDATE=YES. Using the IMPORT procedure, you can have date and time values read in using the DATE format.

To export SAS data values with DATE, TIME, or DATE TIME format to a Microsoft Access database, SAS values are written using the date and time data type. However, Microsoft Access can identify and display the values in the correct DATE, TIME, or DATETIME format.

**Setting Environment Variables: BOOL\_VAL**

By default, the SAS/ACCESS LIBNAME engine imports YES (TRUE) into SAS as the numeric value 1. However, you can set BOOL\_VAL option value to ASIS, which tells SAS to import YES (TRUE) value into SAS as the numeric value -1 instead.

*Note:* Microsoft saves YES (TRUE) with the numeric value -1 internally, and SAS saves TRUE value with the numeric value 1 internally.

Set the environment variable with the following statement:

```
/* To have the YES value imported into SAS as numeric value-1 */
OPTIONS SET=BOOL_VAL ASIS;

/* Reset to the default value */
OPTIONS SET=BOOL_VAL SAS;
```

**Setting Environment Variables: TIME\_VAL**

By default, the SAS/ACCESS LIBNAME engine loads SAS time values into Microsoft Access databases with a time value less than or equal to 24 hours. Set the environment variable with this statement:

```
/* To have SAS time values exported to Microsoft Access
database with SAS date/time base, 01Jan1960. */
OPTIONS SET=TIME_VAL SAS;
```

When importing data with the SCANTIME=YES option, SAS scans date/time values in the columns and assigns the TIME. format if all the values are in the year 1960. Otherwise SAS assigns the DATETIME. format to the columns.

```
/* Reset to default value */
OPTIONS SET=TIME_VAL ONEDAY;
```

By default, SAS loads the numeric value of format w.d with a decimal field size into Access database files, Version 2000 or later. The decimal type field takes more space than a double type field. To manage your storage space, set the environment variable with the following statement:

```
/* To have SAS numeric values loaded into an Access database
with a double field type, to save storage space. */
OPTIONS SET=LOAD_DBL YES;

/* To reset to the default behavior */
OPTIONS SET=LOAD_DBL NO;
```

## Part 4

---

# LIBNAME PCFILES Engine and PC Files Server on Microsoft Windows

<i>Chapter 14</i>	
<b>PC Files Server Administration</b> .....	161
<i>Chapter 15</i>	
<b>LIBNAME Statement for PCFILES Engine on Linux, UNIX, and Microsoft Windows</b> .....	173
<i>Chapter 16</i>	
<b>Pass-Through Facility: PCFILES on Linux, UNIX, and Microsoft Windows</b> .....	205
<i>Chapter 17</i>	
<b>Special Query Support</b> .....	219



*Chapter 14***PC Files Server Administration**

---

<b>Overview</b> .....	<b>162</b>
PC Files Server Application .....	162
PC Files Server Operating Modes .....	163
PC Files Server Installation .....	163
<b>Windows Service</b> .....	<b>164</b>
Service Options .....	164
<b>Desktop Application</b> .....	<b>164</b>
Overview .....	164
Desktop Application Window .....	165
Window Items .....	165
<b>PC Files Server Configuration</b> .....	<b>166</b>
Configuring the PC Files Server .....	166
Port Number or Service Name .....	167
Maximum Connections .....	167
Data Encryption .....	167
<b>Authentication</b> .....	<b>167</b>
Authentication (Security Enforcement) Overview .....	167
Access to PC Files Server .....	168
Access to Individual Files .....	168
System Administrator Tasks .....	168
Security Model for Microsoft Windows Vista and Above .....	168
<b>PC Files Server Autostart</b> .....	<b>169</b>
<b>Local Security Policy Configuration</b> .....	<b>169</b>
User Authentication .....	169
Configure User Accounts .....	170
<b>Constraints</b> .....	<b>170</b>
<b>Shared Information</b> .....	<b>171</b>

---

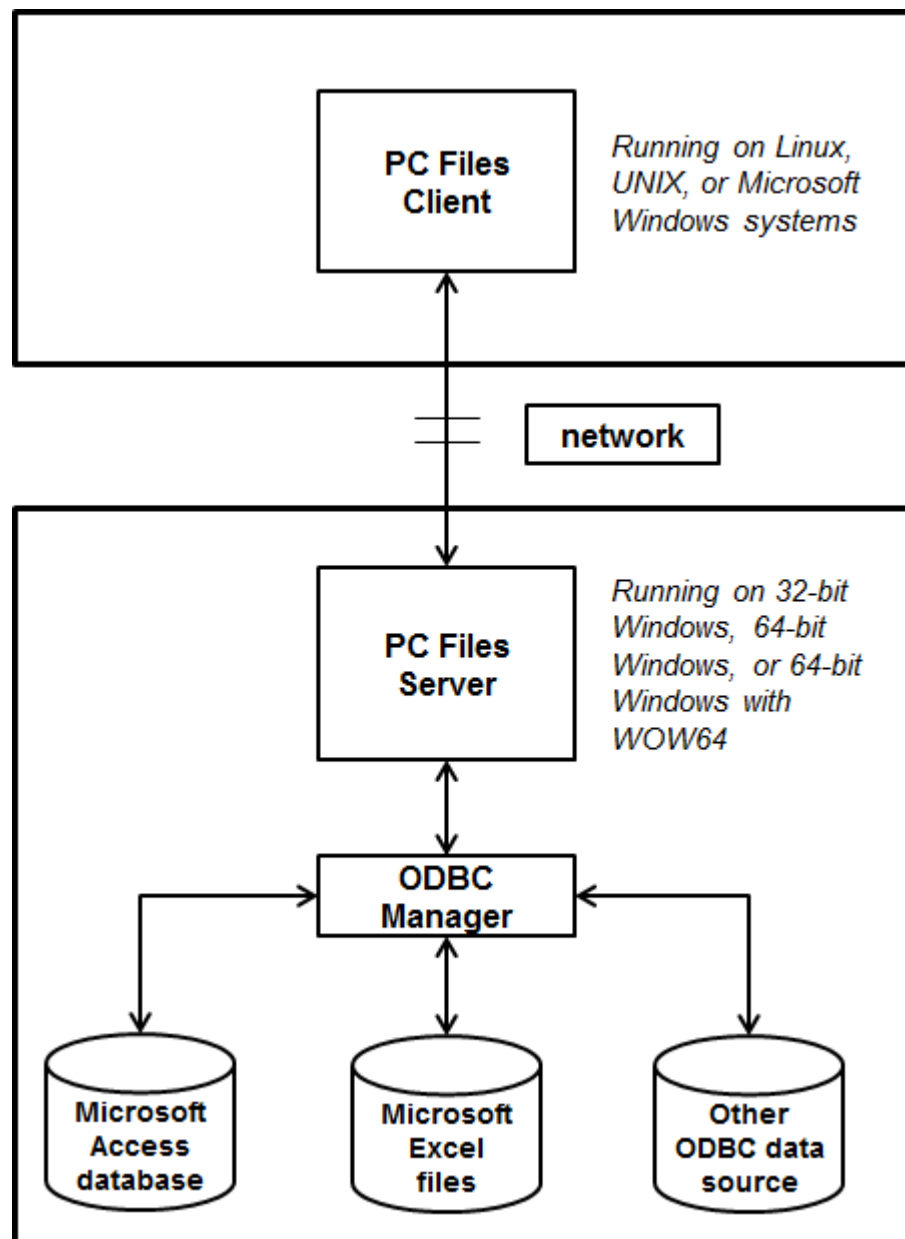
## Overview

### PC Files Server Application

PC Files Server is an application that receives client requests to access Microsoft Windows-specific data files, such as Microsoft Excel and Microsoft Access. It runs on both 32- and 64-bit Windows as either a 32-bit or a 64-bit application.

To access PC data, SAS/ACCESS Interface to PC Files must be installed on the client. The server must be installed and running on the Windows PC where the PC data resides.

**Figure 14.1** PC Files Server Interaction



## **PC Files Server Operating Modes**

Starting with SAS/ACCESS 9.2, the server can run in two modes: Server Mode as a desktop application (available before SAS/ACCESS 9.2), and Service Mode as a Windows service. Only one instance of the server can be running on a single PC at any given time.

The behavior of the PC Files Server is determined during installation and configuration. Areas of operation that are configured include operation as either 32- or 64-bit, the port number used for TCP/IP server connections, and the parameters to start the PC Files Server as a Windows Service or a desktop application.

## **PC Files Server Installation**

### **PC Files Server Application Modes (32-bit versus 64-bit)**

The application mode is automatically determined by any existing ACE driver; otherwise, it defaults to 32-bit.

Starting in SAS 9.3, PC Files Server can be run as a 64-bit Windows application on 64-bit PCs. It is important to note that the associated ACE driver (the ODBC driver to access the supported PC file types) must be of the same “bitness” (32 or 64) as the PC Files Server. Starting with Microsoft Office 2010, the ACE driver is supplied in either 32- or 64-bit form. The “bitness” of the PC Files Server installed is determined by any existing ACE driver. If none is found, then the 32-bit ACE driver is automatically installed, along with 32-bit PC Files Server.

If you have a need to force the installation of a particular “bitness” of PC Files Server, ensure that you manually install the appropriate ACE driver first. Please note that only one version of the ACE driver can be installed on a given system. You cannot install both the 32- and the 64-bit ACE drivers. The same applies to PC Files Server.

### **Port Number Selection Dialog Box**

Starting with PC Files Server 9.3, the default port number used for TCP/IP server connections has changed from 8621 to 9621 in order to avoid a conflict with another SAS product. LIBNAME and PROC IMPORT and PROC EXPORT commands default to PORT=9621 if the PORT option is omitted. PC Files Server must use the same port number that you expect the SAS/ACCESS commands to use. We suggest using the new default of 9621. Or, you can change it to suit your specific needs.

### **Windows Service Start-up Selection Dialog Box**

Check this option to automatically run PC Files Server as a Windows Service in the background. This is the default and actually starts the service and sets the service start-up type to “automatic.” To run PC Files Server manually as a desktop application, ensure that this option is cleared. Please note that only one instance of PC Files Server can run at a given time. You cannot run PC Files Server as a Windows Service and a desktop application at the same time.

---

## Windows Service

The server can be manually set to run as a Windows service. To run as a Windows service: select **Start** ⇒ **Control Panel** ⇒ **Administrative Tools** ⇒ **Services**. Locate **SAS PC Files Server** in the **Name** column. Select **Properties**.

### Service Options

**Service name:** Displays the name of the service (Default). SAS PC Files Server

**Display name:** Specifies the name of the service. The name appears in the Name column in the details pane.

**Description:** Description (Default): Enables SAS/ACCESS Interface to PC files, such as Excel and Microsoft Access.

**Path to executable:** specifies the path and filename of the service.

```
'C:\Program Files\SAS\PCFilesServer\9.3\pcfservice.exe' -name  
"SAS PC Files Server"
```

*Note:* The path and filename cannot be changed.

**Start-up type:** Displays the start-up type of the service.

- **Automatic.** Specifies that PC Files Server starts automatically when the system starts.
- **Manual.** Specifies that a user or a dependent service can start PC Files Server.
- **Disabled.** Prevents the system, a user, or any dependent device from starting PC Files Server.

**Service status** displays the current status of the Windows service, as follows:

1. **Started.** The service is running.
2. **Stopped.** The service is not running.
3. **Paused.** The service is paused.
4. **Resuming.** The service is resuming after being paused.

---

## Desktop Application

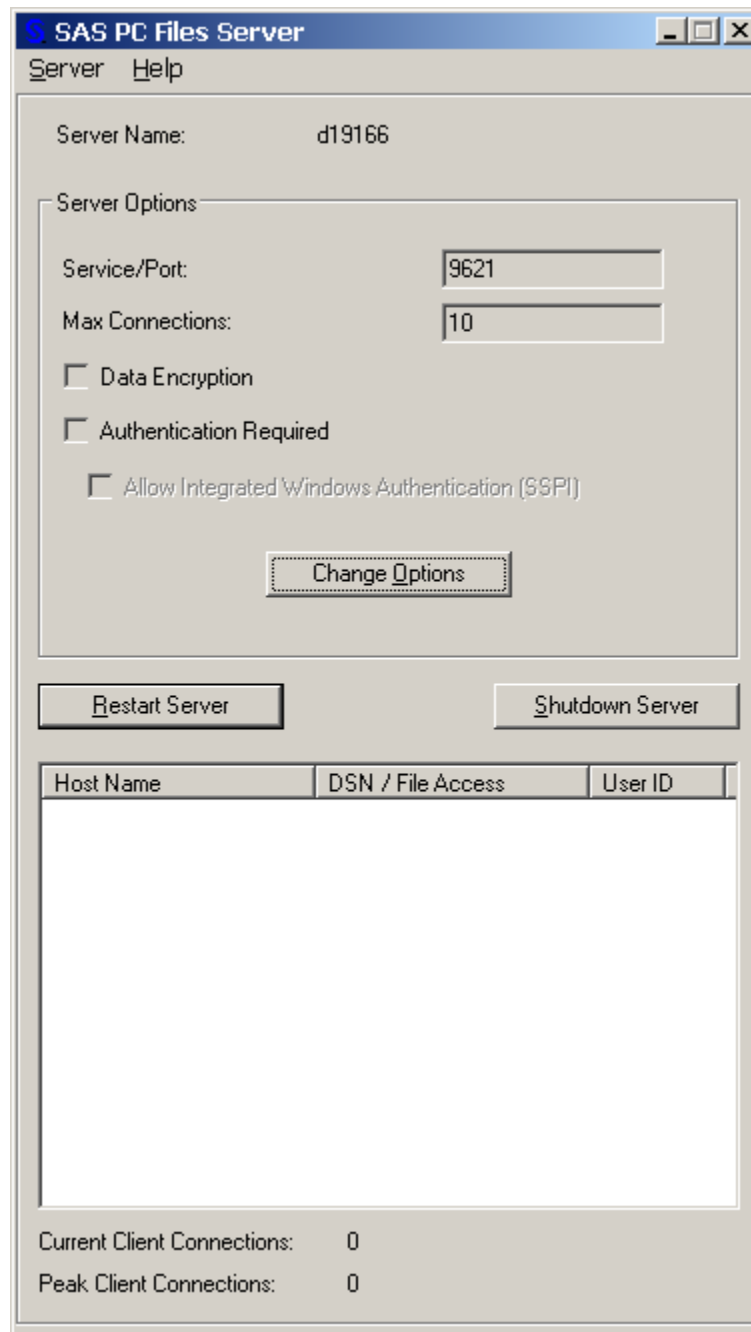
### Overview

To run server mode, stop the service mode. **Start** ⇒ **Control Panel** ⇒ **Administrative Tools** ⇒ **Services**.

Locate, and then open the **Properties window** for **SAS PC Files Server**. Click **Stop** to stop the service.

## Desktop Application Window

To start PC Files Server as a desktop application (in server mode), select **Start** ⇒ **SAS** ⇒ **PC Files Server**. The PC Files Server desktop application window displays.



## Window Items

- **Server Name:** Names the Windows PC where PC Files Server is running.
- **Service | Port:** A communications end point at which a server listens for a request for service from the client application. The default port is 9621.

- **Max Connections:** The maximum number of concurrent connections that this server supports.
- **Data Encryption:** Encrypts data during transfer.
- **Authentication Required:** Requires users to provide credentials before connecting to PC Files Server. These can be in the form of a user ID and password or integrated windows authentication (SSPI).
- **Allow Integrated Windows Authentication (SSPI) :** Windows 64-bit users can process credentials between Windows PCs without having to explicitly give a user ID and password.

*Note:* This option is available only when **Authentication Required** is selected.

- **Change Options:** Displays a dialog box that enables you to change the connection options. A note states that the PC Files Server must be restarted in order for the changes to take effect.
- **Restart Server:** Restarts the server including all setting changes.
- **Shutdown Server:** Stops the server and closes the application window.
- **Host Name:** Lists host names of active server connections.
- **DSN | File Access:** Displays file access requests of active server connections.
- **User ID:** Displays the user ID of active server connections.
- **Current Client Connections:** Displays the total number of active connections.

*Note:* When a single user opens multiple connections, this displays the most current information.

- **Peak Client Connections:** The greatest number of active connections during the current server session.

**TIP** To find out if the 32-bit or 64-bit version of the PC Files Server is installed, use **Help** ⇒ **About** from the application menu.

---

## PC Files Server Configuration

### Configuring the PC Files Server

Both server mode and service mode store the server configuration settings in the Windows registry. Changing the settings while in server mode affects the server running service mode. If you run the server in service mode, a separate user interface is not provided. The default configuration should be sufficient for most installations. To change the configuration options, run the server in server mode.

1. Stop the Windows service that runs the server. See “[Windows Service](#)” on page 164.
2. Start the server desktop application. See “[Desktop Application](#)” on page 164.
3. Change the options that you want to change.
4. Stop the PC Files Server desktop application.
5. Start the Windows service that runs PC Files Server.

## Port Number or Service Name

The PC Files Server Port Number must be unique on a given PC. You cannot run multiple servers of any kind that use the same port number on a single PC. You can, however, use the same server Port Number on different PCs.

The **Port Number** for PC Files Server defaults to 9621. 9621 is also the default port number that the LIBNAME statement uses for connections to PC files. If you change the port number on the server, users accessing that server must add the PORT= *number* option to their LIBNAME statements.

The **Port Number** or **Service Name** is saved in the Windows registry, and it is used each time that the server PC Files Server is run.

## Maximum Connections

**Max Connections** specifies the number of concurrent connections that the server can support. Configure the number of connections based on the load that you expect from your SAS clients.

Each command that uses the server uses one connection. There can be multiple users from different SAS sessions accessing the server concurrently. Consider all the connections when setting the **Max Connections** value. The number of **Max Connections** is saved in the Windows registry, and it is used each time the PC Files Server is run.

## Data Encryption

To enable data encryption between SAS/ACCESS Interface to PC Files on the client and PC Files Server, select the **Data Encryption** check box.

The state of the **Data Encryption** is saved in the Windows registry, and it is used each time PC Files Server is run.

---

# Authentication

## Authentication (Security Enforcement) Overview

Authentication enables PC Files Server system administrators to secure the server and enforce security. You can configure PC Files Server so that a user ID and a password are required to connect to a server and access files. You can also configure PC Files Server on specific hosts to require a user ID and password.

All the commands that allow server access support user authentication. The credentials supplied to PC Files Server are verified against the Windows login database. These are the same credentials that are required to interactively log on to a PC.

SERVERUSER=, SERVERPASS=, and SSPI= options have been added to the LIBNAME statement, and the IMPORT and EXPORT procedures. Use these options with PC Files engine to supply credentials to the PC Files Server.

See “LIBNAME Statement Syntax” on page 97.

*Note:* If the client PC is on a domain, the credentials are compared to the domain data, instead of the local data.

### **Access to PC Files Server**

If the “bitness” of SAS on Microsoft Windows conflicts with the “bitness” of the ACE ODBC driver installed and therefore the “bitness” of PC Files Server (such as running 64-bit SAS on a machine with a 32-bit ACE driver) SAS cannot directly access PC files (EXCEL or ACCESS engines), but must rather use PC Files Server to bridge the “bitness” gap (PCFILES engine).

Access to the server is granted only if the credentials supplied are valid on the target PC. When connecting from a UNIX workstation to the PC, the UNIX credentials (User ID and password) can be different from the credentials used to access the PC files.

### **Access to Individual Files**

After the server is secured, server administrators can enable security settings at the file level. When a server connection is established, access to individual files is secured using the credentials specified by the user. File access is administered as if the client is logged on to that PC.

### **System Administrator Tasks**

To enforce a security policy, the system administrator should ensure that the following configurations and settings are implemented:

- Local security policy is configured, see “[Local Security Policy Configuration](#)” on [page 169](#).
- Server authentication is configured, with PC Files Server desktop application. **Start** ⇒ **All Programs** ⇒ **SAS** ⇒ **PC Files Server**. Select **Authentication Required**. See “[Service Options](#)” on [page 164](#) for additional information.
- Set PC-to-PC Connections option to **Allow Integrated Windows Authentication (SSPI)**. This option is for clients on PCs running Windows connecting to PC Files Server. Credentials are exchanged between the server and the client. The client PC does not have to explicitly set credentials. See “[Service Options](#)” on [page 164](#).
- Access to the server requires a user ID and password using the SERVERUSER= and SERVERPASS= options. For the Windows environment, you can also use the SSPI= option.

### **Security Model for Microsoft Windows Vista and Above**

The enhanced Microsoft Windows Security Model, beginning with Microsoft Windows Vista, is designed to make it more difficult for viruses, and other software that interferes with computer functions, to install themselves on the PC. When logged in as “Administrator” or part of the “Administrators Group,” certain privileges are temporarily not available to the operating system. The privileges are returned when needed and confirmed by a “Windows needs your permission to continue” dialog box. This guarantees that the user is aware of potential security risks.

When starting the server on Windows Vista and later, manual intervention is required to enable permissions. The confirmation is not required when running the server as a

Windows service or if the Windows security features have been disabled for Windows Vista and above.

---

## PC Files Server Autostart

The PC Files Server Autostart feature provides a convenient way to use PC Files Server for the current SAS session without having to actually run it on a local machine. PC Files Server Autostart features the following enhancements:

- Starts PC Files Server in the background as needed and stops the server when finished.
- Does not require the server setup or options.
- Communicates with the SAS client using a named pipe.
- Does not transfer data over the network. This eliminates the need for data encryption.
- Runs independent of network settings and any other instances of PC Files Server.
- Always runs with the credentials of the SAS client. This eliminates the need for authentication.
- Autostart instances of PC Files Server are independent and using their own communication mechanisms. This eliminates the possibility of an auto-started server interfering with other servers.

To use PC Files Server autostart features you must:

- Run SAS on a PC running Microsoft Windows.
- Install PC Files Server on the same PC.
- Use a PC Files Server related engine to access either local files or files that are accessible with the `\\host-name \folder \filename` specification.
- Omit the `SERVER=` and `PORT=` options.
- Avoid using `SERVERUSER=`, `SERVERPASS=`, or `SSPI=` options.

In the following `LIBNAME` statement and `IMPORT` procedure, autostart is triggered by the “missing” `SERVER=` and `PORT=` options.

```
LIBNAME DB PCFILES PATH='C:\myfile.mdb';

PROC IMPORT OUT=work.test DATAFILE='C:\myfile.mdb' DBMS=ACCESSCS REPLACE;
run;
```

---

## Local Security Policy Configuration

### *User Authentication*

For server user authentication to work, the server must be able to create user-specific subprocesses with the credentials specified. Windows allows this only if certain Windows Security settings are set on the PC running the server.

When running the server exclusively as a Windows service, use the default account of SYSTEM. Changes might not be needed if the SYSTEM account has all the required privileges set.

## Configure User Accounts

The user account running the server must be in the Administrators group. To access the Administrators Group;

1. Select **Start** ⇒ **Settings** ⇒ **Control Panel** ⇒ **User Accounts** ⇒ **Users**.
2. Select the user account running the server.  
If you are on a domain, it appears as the domain name in the **Domain** column. Select the **domain-level** user account.
3. **Properties** ⇒ **Group Membership** ⇒ **Other**.
4. Open the pull-down list and select Administrators.
5. Click **OK** to close the Group Membership tab.
6. If prompted to log off, click **Cancel**.
7. Enable the following User Rights for the Administrators Group:
  - a. Act as part of the operating system.
  - b. Adjust memory quotas for a process.
  - c. Replace a process level token.

To verify or update these rights, select **Start** ⇒ **Control Panel** ⇒ **Administrative Tools** ⇒ **Local Security Policy** ⇒ **User Rights Assignment**.

8. In the **Security Settings** pane, open **Local Policies** ⇒ **User Rights Assignment**.
9. In the **Policy** column, open the user right to be changed and add **Administrators**. Ensure there is an "s" at the end of **Administrators**. Administrator (singular) is a specific user account.
10. Repeat the sequence for each of the user rights.
11. Verify that the **Administrators** group has been added to each of the three user rights, as indicated previously.
12. Add the **Authenticated Users** group to the **Log on as batch job** user right.
13. Log off and log back in for the changes to take effect.

---

## Constraints

- You cannot mix 32- and 64-bit ACE ODBC drivers on the same machine (64-bit ACE available starting with Microsoft Office 2010).
- You cannot mix 32- and 64-bit PC Files Servers on the same machine.
- The “bitness” of the ACE driver must match the “bitness” of PC Files Server. The installer enforces this but care should be taken if the ACE driver is ever replaced.

- If SAS on Windows matches the “bitness” of the installed ACE driver, then PC files can be accessed directly using the EXCEL and ACCESS engines.
- If SAS on Windows does not match the “bitness” of the installed ACE driver, then PC Files Server must be used to bridge the “bitness” gap. The Autostart feature simplifies this when running SAS on Windows.
- The server can run in server mode or service mode. However, only one instance of the server can be running on a single PC at any given time.
- Service names and port numbers must be unique on a given PC.
- To modify the settings to follow the constraints:
  1. Stop the server.
  2. Make necessary changes.
  3. Restart the server.
- Each time that you stop or restart the server, all users' sessions are closed. Closing these sessions might result in loss of data.
- Although you can change server configuration only in server mode, the updated values also apply when running in service mode.

---

## Shared Information

After the server is started and running, the clients need the following information to access the server:

- If the clients are accessing a server that requires authentication, specify the SERVERUSER= and SERVERPASS= options. For the Windows environment, you can also specify the SSPI= option.
- Server name (host name or IP address of the PC running the server).
- Port number or service name. If you do not use the default value, you must specify the port number in the LIBNAME statement on the client PC.

`PORT=value;`

- Path to any files, or ODBC data sources to which they have access that is relative to the files system on the server PC, such as “C:\Excel files\mysheet.xls”. Enclose the path in double quotation marks.



## Chapter 15

# LIBNAME Statement for PCFILES Engine on Linux, UNIX, and Microsoft Windows

---

<b>LIBNAME Statement for PCFILES Engine on Linux, UNIX, and Microsoft Windows</b> . . . . .	<b>174</b>
Overview . . . . .	174
Sorting PC Files Data . . . . .	174
Using SAS Functions with PC Files Data . . . . .	174
<b>Dictionary</b> . . . . .	<b>174</b>
Syntax for PCFILES Engine on Linux, UNIX, and Microsoft Windows . . . . .	174
SAS LIBNAME Options for PCFILES on Linux, UNIX, and Microsoft Windows . . . . .	181
Data Set Options for PCFILES on Linux, UNIX, and Microsoft Windows . . . . .	186
AUTOCOMMIT . . . . .	187
COMMAND_TIMEOUT . . . . .	187
CURSOR_TYPE . . . . .	188
DBCMMIT . . . . .	188
DBCONDITION . . . . .	189
DBCREATE_TABLE_OPTS . . . . .	190
DBENCODING . . . . .	190
DBFORCE . . . . .	191
DBGEN_NAME . . . . .	192
DBKEY . . . . .	193
DBLABEL . . . . .	193
DBMAX_TEXT . . . . .	194
DBNULL . . . . .	195
DBNULLKEYS . . . . .	196
DBSASLABEL . . . . .	196
DBSASTYPE . . . . .	197
DBTYPE . . . . .	197
ERRLIMIT . . . . .	198
INSERT_SQL . . . . .	199
INSERTBUFF . . . . .	199
NULLCHAR . . . . .	200
NULLCHARVAL . . . . .	201
READBUFF . . . . .	202
SASDATEFMT . . . . .	202

---

## LIBNAME Statement for PCFILES Engine on Linux, UNIX, and Microsoft Windows

### Overview

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. You can also read from and write to a Microsoft Access, Microsoft Excel, or ODBC table directly.

### Sorting PC Files Data

Because PC data librefs refer to database and workbook objects such as tables, they are stored in a different format than SAS data sets. This is important to remember when you access and work with PC files data.

You can sort the observations in a SAS data set and write the output to another data set. In a Microsoft Access database, sorting data has no effect on how it is stored. Because your data might not be sorted, sort the data at the time of query.

When you sort PC files data, the results might vary. The spreadsheet or database can place data with NULL values first or last in the result set.

*Note:* NULL values are translated in SAS to missing values

### Using SAS Functions with PC Files Data

Using librefs that refer to PC files data with SAS functions might return a different value than the value returned when you use the functions with a SAS data set.

For example, the PATHNAME= function returns the pathname for the assigned libref. When the libref refers to PC files data, the function might return the Microsoft Excel filename assigned for the libref.

Use 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 de-assign a libref that refers to PC files data, you must omit this argument.

---

## Dictionary

---

### Syntax for PCFILES Engine on Linux, UNIX, and Microsoft Windows

Associates a SAS libref with a workbook, database, or ODBC data source.

**Valid in:** Anywhere

---

## Syntax

Form 1: **LIBNAME** *libref* **PCFILES** *LIBNAME* *options* *connection-options*

Form 2: **LIBNAME** *libref* CLEAR | **\_ALL\_** CLEAR

Form 3: **LIBNAME** *libref* LIST | **\_ALL\_** LIST;

## Optional Arguments

### **\_ALL\_**

specifies that CLEAR= and LIST= arguments apply the argument to all librefs.

### **CLEAR**

clears one or all librefs.

Specify *libref* to clear single libref. Specify **\_ALL\_** to clear all librefs.

### **connection-options**

provides 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.

### **LIBNAME options**

defines how 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” on page 121

### **libref**

is any SAS name that serves as an alias to associate SAS with a spreadsheet, data source, or database. A SAS libref is an alias for a virtual or physical directory. Like the global SAS LIBNAME statement, a SAS/ACCESS libref is an alias for a spreadsheet, database, or data source where your data is stored.

### **LIST**

lists the attributes of one or all SAS/ACCESS libraries or SAS libraries to the SAS log.

Specify the *libref* argument to list the attributes of a single library. Specify **\_ALL\_** to list the attributes of all the librefs in your current session.

### **PCFILES**

is the SAS/ACCESS engine for the interface to PC files on Linux, UNIX, and Microsoft Windows.

## Details

### **PC Files Server Connection Options**

#### **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 you should use only when you know the exact syntax of all connection options that the ODBC driver requires for a successful connection.

#### **DBPASSWORD=** *database-file-password*

enables you to access database files with database-level security. This security level can be defined instead of user-level security.

**Alias:** DBPWD | DBPW | PASS | PASSWORD

**Restriction:** Microsoft Access Database only.

**Note:** Database password is case sensitive.

**DBSYSFILE= *workgroup-information-file***

specifies the workgroup information file. This file contains a collection of information defined for the Microsoft Access database. User, group accounts, and passwords that you create, are saved in the workgroup information file.

**Alias:** SYSTEMDB

**Restriction:** Microsoft Access database files only.

**DSN= *data-source-name***

specifies the name of the ODBC data source that is used to access PC data through an ODBC driver on the PC.

**Restriction:** This ODBC data source must be defined on the PC where the PC Files Server is running.

**MSENGINE= ACE | JET**

determines the database engine to use for accessing Microsoft Excel files or Microsoft Access databases. The Microsoft Jet engine supports Microsoft formats up to 2003. The Microsoft Ace engine supports 2007 formats and formats in subsequent releases of Windows.

**Default:** ACE

**PASSWORD= *user-password***

specifies a password required by the data source for the user account.

**Alias:** PWD | PW | PASS | PASSWORD

**Note:** Passwords are case sensitive.

**PATH= *pathname***

specifies the full path and filename for your Microsoft Access database or Microsoft Excel file.

This example assigns the libref db to a Microsoft Excel file. **LIBNAME db**

**PCFILES SERVER=D2323 PATH='c:\demo.xls';**

**Note:** Always use the .mdb or .accdb file extension for Microsoft Access files and the .xls, .xlsx, or .xlsb extension for Microsoft Excel files.

**PORT= *port-number***

specifies the port number or service name that PC Files Server is listening to on the PC.

**Alias:** SERVICE|SERVICE\_NAME

**Default:** 9621

**SERVER= *pc-server-host-name***

specifies the name of the PC running PC Files Server. This name is required for Linux and UNIX users to connect to the server.

**Restriction:** Omitting the SERVER= option on Windows clients invokes Autostart.

**Note:** The name can be a simple computer name (wpx320), a fully qualified network name (wpx320.domain.com), or an IP address.

**SERVERPASS= *server-user-password***

specifies the password for the PC Files Server for the User ID given. If the account has no password, omit this option. Always enclose the value in quotes in order to preserve the case of the password.

**Alias:** SERVERPASSWORD | SERVERPW | SERVERPWD

**Notes:**

Passwords are generally case sensitive.

Use the PASSWORD= option for database passwords.

**Example:** LIBNAME using explicit user name and password.

```
LIBNAME db PCFILES PATH='C:\myfile.mdb'
SERVER=fileserv;
SERVERUSER='mydomain\myusername'
SERVERPASS='mypassword';
```

**SERVERUSER= 'domain\server-user-name'**

specifies the domain name and User ID for the PC running PC Files Server. Always enclose the value in quotes, otherwise the backslash can be misinterpreted by the SAS parser.

**Alias:** SERVERUID

**Notes:**

If you are not on a domain, omit the domain name and the backslash.

Use the USER= option for database User IDs.

**Example:** LIBNAME using explicit user name and password.

```
LIBNAME db PCFILES PATH='C:\myfile.mdb'
SERVER=fileserv;
SERVERUSER='mydomain\myusername';
SERVERPASS='mypassword';
```

**SSPI= YES | NO**

enables the PC Files Server to allow Integrated Windows Authentication. This is a mechanism for the Windows client and server to exchange credentials.

**Default:** NO

**Restriction:** Microsoft Windows only.

**Note:** SSPI can also be enabled by specifying the –SSPI option on the SAS command line.

**Example:** LIBNAME using SSPI.

```
LIBNAME db PCFILES PATH='C:\myfile.mdb'
SERVER=localhost;
SSPI='yes';
RUN;
```

**TYPE= EXCEL | ACCESS**

specifies the file type in the PATH= statement.

**Note:** Use TYPE= if the file identified in the PATH= statement does not have an .xls or .mdb file extension.

**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 on your .mdb file, you need to use the USER= and PASSWORD= options to access your file.

**Alias:** UID

**Restriction:** Microsoft Access database files only.

**Note:** Use the SERVERUSER= option to connect to a server.

**VERSION= 2010 | 2007 | 2003 | 2002 | 2000 | 97 | 95 | 5**

sets the version for a new Excel workbook.

**Alias:** VER

**Default:** 97

**Restriction:** Excel workbooks only.

**Notes:**

Excel 2010, 2007, 2003, 2000, and 97 share the same file format. Excel 5 and 95 share a different file format.

You do not need to specify this option for an existing Excel file.

**Access Data Directly from a PC File**

You can use SAS/ACCESS Interface to PC Files on Linux and UNIX to directly access PC data from Linux and UNIX. You can read from and write to a variety of PC file data residing on a PC, including Excel, Microsoft Access, and any other ODBC data source.

The engine uses ODBC to support assigning a libref to Excel and Microsoft Access files on a PC from Linux and UNIX. You can reference spreadsheets, databases, and other ODBC data sources directly in a DATA step or SAS procedure. You can also directly read from and write to a Microsoft Access file or a Microsoft Excel file.

**Disassociating a Libref from a SAS Library**

To clear a libref, use a LIBNAME statement, specifying the libref, and the CLEAR option as shown;

```
LIBNAME mypclub CLEAR;
```

Clear all librefs by submitting **LIBNAME CLEAR;**

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

**Writing SAS Library Attributes to the SAS Log**

Use a LIBNAME statement to write the attributes of one or more SAS/ACCESS libraries or SAS libraries to the SAS log. Specify *libref* to list the attributes of a single SAS/ACCESS library or SAS library, as shown;

```
LIBNAME mypclub LIST;
```

```
LIBNAME _ALL_ LIST;
```

**Examples****Example 1: Assigning a Libref to a Microsoft Access Database**

```
LIBNAME mymdb PCFILES SERVER=D2323 PATH='c:\demo.mdb';
```

The demo.mdb database contains a number of objects, including several tables, such as Staff. After you assign a libref, you can reference the Microsoft Access table like a SAS data set. You can use it as a data source in any DATA step or SAS procedure. In this 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;
```

Create a SAS data set with Microsoft Access.

```
DATA newds;
  SET mymdb.staff (KEEP=idnum lname fname);
RUN;
```

SAS procedures such as PROC SQL, PROC PRINT, PROC CONTENTS, and PROC DATASETS, use the libref.

---

**List all database objects in the library with the DATASETS procedure:**

```
PROC DATASETS LIBRARY=mymdb;
QUIT;
```

To improve performance, it is recommended that you use the data set options INSERTBUFF= for writing and READBUFF= for reading and set an appropriate value.

---

**Create a table named Invoice in a Microsoft Access database from a SAS data set named Invoice:**

```
PROC SQL;
CREATE TABLE mymdb.Invoice (INSERTBUFF=25) as SELECT * FROM invoice;
QUIT;
```

**Example 2: Assigning a Libref to a Microsoft Excel Workbook**

Create a libref, myxls, for a Microsoft Excel workbook:

```
LIBNAME myxls PCFILES SERVER=D2323 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 Excel spreadsheet like a SAS data set and use it as a data source in any DATA step or SAS procedure. In this example, a SAS data set is created from a Microsoft Excel sheet:

```
DATA a;
SET myxls.'sheet1$'n;
RUN;
```

When using a LIBNAME statement with Excel, refer to Excel sheets as n-literals because of the “\$” character. If you are referencing a named range in a Microsoft Excel spreadsheet, it is not necessary to refer to it as an n-literal.

---

**Reference a named range called page one in a Microsoft Excel workbook:**

```
DATA a;
SET myxls.pageone;
RUN;
```

Create a Microsoft Excel file and use a SAS data set to populate a sheet in that file. Create a named range for the sheet:

```
DATA myxls.air;
SET sashelp.air;
RUN;
```

Use the libref with any SAS procedures such as PROC SQL, PROC PRINT, PROC CONTENTS, and PROC DATASETS.

---

**This SAS program uses the DATASETS procedure to list all database objects in the library.**

```
PROC DATASETS LIBRARY=mymdb;
QUIT;
```

To improve performance, it is recommended that you use the data set option READBUFF= and set an appropriate value. This example reads in data from a range called Invoice in a Microsoft Excel workbook.

---

**When writing to a Microsoft Excel file, the PC Files Server does not support the INSERTBUFF= option with value greater than 1.**

```
PROC SQL;
    SELECT * FROM myxls.Invoice (READBUFF=25);
QUIT;
```

---

### **Example 3: Assigning a Libref to a Microsoft SQL Server Database**

**Create a libref, myqlsrv, to a SQL Server database via ODBC, using the server on the PC:**

```
LIBNAME myqlsrv PCFILES SERVER=D2323 DSN=MQIS USER=scott
    PWD=tiger SCHEMA=dbo;
```

---

**Using the myqlsrv libref, create a SAS data set called sqltest from the crime table in the SQL Server database:**

```
DATA work.sqltest;
    SET myqlsrv.crime;
RUN;
```

---

**or**

```
PROC sql;
    CREATE TABLE work.sqltest AS SELECT * FROM myqlsrv.crime;
QUIT;
```

---

**Using the myqlsrv libref, create a SQL Server table called newtable from the SAS data set, sqltest:**

```
DATA myqlsrv.newtable;
    SET sqltest;
RUN;
```

---

### **Example 4: Assigning a Libref to an Oracle Database**

**Create a libref, ora, to an Oracle database table via ODBC, using the PC Files Server on the PC:**

```
LIBNAME ora PCFILES SERVER=D2323 DSN=ORA9MS
    USER=scott 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;
```

---

## SAS LIBNAME Options for PCFILES on Linux, UNIX, and Microsoft Windows

The following SAS LIBNAME statement options provide enhanced control over the way that SAS processes PC files data.

**See:** [“LIBNAME Statement Syntax” on page 101](#) .

---

### Details

Many of these LIBNAME options are also available as data set options. See [“Overview of Data Set Options” on page 103](#) for additional information.

*Note:* Note that these are advanced options that do not need to be specified for many of the tasks that you perform.

#### **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 they are submitted.

##### **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.

**Alias:** TIMEOUT

**Default:** 0 (no time-out)

#### **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 these conditions are met:

- the participating librefs are created by LIBNAME statements that specify identical values for the CONNECTION= option and CONNECTION\_GROUP= option.
- 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= option when you specify CONNECTION\_GROUP= option.

**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=DYNAMIC | FORWARD\_ONLY | KEYSSET\_DRIVEN | STATIC**

specifies the cursor type for read-only cursors and for cursors to be updated.

**DYNAMIC**

specifies that the cursor reflects all 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 SQL server interfaces.

**FORWARD\_ONLY**

specifies that the cursor behaves like a DYNAMIC cursor, except that it supports only fetching of rows sequentially.

**KEYSET\_DRIVEN**

specifies that the cursor determines which rows belong to the result set when you open the cursor. 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 you open the cursor. 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.

**Alias:** CURSOR

**DBCMMIT= *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 DBCMMIT=0, a commit is issued only once (after the procedure or DATA step completes). If the DBCMMIT= option is explicitly set, SAS/ACCESS fails any update that has a WHERE clause.

*Note:* If you specify both DBCMMIT= and ERRLIMIT=, the DBCMMIT= is issued before the rollback. Because the DBCMMIT= option is issued before the rollback, the DBCMMIT= option overrides the ERRLIMIT= option (rollback) 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 that the names 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 applies only when you are reading, appending, and updating character data in a Microsoft Access database or Excel workbook from SAS. Although you can specify a value less than 256, it is not recommended for reading data from a 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.

If you specify DBNULLKEYS=YES and a column that is not defined as NOT NULL in the DBKEY= option, SAS generates a WHERE clause that finds NULL values.

If you specify DBKEY=COLUMN and COLUMN is not defined as NOT NULL, SAS generates a WHERE clause that enables SAS to prepare the statement once. The statement can be used any value in the column. For example **WHERE ((COLUMN =?) OR ((COLUMN IS NULL) AND (? IS NULL)))**;

**Note:** This syntax has the potential to be much less efficient than the shorter form of the WHERE clause.

**NO**

When you specify DBNULLKEYS=NO or specify a column as NOT NULL in the DBKEY= option, SAS generates a simple WHERE clause. If you know that there are no NULL values for the columns that you specify in the DBKEY= option, you can use DBNULLKEYS=NO.

If you specify DBNULLKEYS=NO and DBKEY=COLUMN, SAS generates a shorter form of the WHERE clause. This is true whether DBKEY=COLUMN is defined as NOT NULL or not.

**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.

**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***

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 these 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 engine.

**NOGENSQL**

prevents PROC SQL from generating SQL to pass 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-row insert. The value for INSERTBUFF= must be a positive number. If the INSERTBUFF= value is greater than the DBCOMMIT= value, the DBCOMMIT= value overrides it. If you assign a value that is greater 1, the SAS application notes that indicate the success or failure

of the insert operation might be incorrect. Notes generated by SAS represent information for a single insert. This is also true 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. In addition, if too many rows are read at once, values returned to SAS might be out of date.

**Alias:** ROWSET | ROWSET\_SIZE

**Default:** 1

**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. If the maximum length found is greater than what is specified in the DBMAX\_TEXT= option, the smaller value is applied as the SAS variable length.

For Excel, this option applies to all character data type columns. For Microsoft Access, this applies only to the MEMO data type field. It 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 the Jet provider is used as the SAS variable width. If the returned length is greater than the length specified with the DBMAX\_TEXT= option, the smaller value is assigned to the SAS variable length.

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

**Alias:** SCANTEXT | SCANMEMO

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

**SCAN\_TIMETYPE= *YES* | *NO***

specifies whether to scan all row values for a DATE or TIME data type field, and automatically determine the TIME data type based on the setting. Option values YES turn on the scan function. Option value NO turns off the scan function.

**YES**

specifies that a Microsoft Excel column with all time values (internal value is less than 1) is assigned a TIME8. format.

**NO**

specifies that the SCAN function is not enabled.

**Alias:** SCAN\_TIME | SCANTIME

**Default:** NO

**See:** USE\_DATETYPE

**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.

**Alias:** STRDATES

**Default:** NO

**USE\_DATATYPE= YES | NO**

specifies whether to use DATE9. format for date columns in the data source table while importing data from a Microsoft Excel workbook. Specifies whether to use a format for date columns in the data source table while importing data from a Microsoft Access table

YES

specifies that the SAS DATE9. format is assigned for date columns in Excel data source table.

NO

specifies that the SAS DATE formats are not assigned to the data source table.

**Alias:** USE\_DATE | USEDATE

**Default:** YES for a Microsoft Excel workbook

NO for a Microsoft Access database

---

## Data Set Options for PCFILES on Linux, UNIX, and Microsoft Windows

You can specify SAS/ACCESS data set options on a SAS data set when you access PC files data with the LIBNAME Statement Syntax for PC Files on Linux and UNIX.

**Note:** 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.

**See:** [“Overview of Data Set Options” on page 103](#), [“Syntax for PCFILES Engine on Linux, UNIX, and Microsoft Windows” on page 174](#).

---

### Details

This generic example shows 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 *SAS Data Set Options: Reference*.

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

---

## AUTOCOMMIT

Determines whether the ACCESS engine commits (saves) updates as soon as you submit them.

**Valid in:** DATA and PROC steps

**Default:** LIBNAME option setting

---

### Syntax

AUTOCOMMIT=YES | NO

### Syntax Description

#### 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.

---

## 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:** LIBNAME option setting

**See:** To assign this option to a group of tables, use the COMMAND\_TIMEOUT option specified in [“LIBNAME Options” on page 121](#).

---

### Syntax

COMMAND\_TIMEOUT= *number-of-seconds*

### Syntax Description

#### number-of-seconds

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

---

## CURSOR\_TYPE

Specifies the cursor type for read-only cursors and for cursors to be updated.

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

**Table 15.1** OLE DB Properties Applied to an Open Row Set

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” on page 121](#).

---

## DBCMMIT

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:** LIBNAME option default: 1000

---

## 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 the DBCOMMIT= option and the ERRLIMIT= option, and these options collide during processing, then the DBCOMMIT= option is issued first and the rollback is issued second. Because the DBCOMMIT= option is issued before the ERRLIMIT= option, the DBCOMMIT= option overrides the ERRLIMIT= option in this situation.

## Example: Issue Automatic Commit Statement

```
/* a commit is issued after every 10 rows are inserted */
DATA myxls.dept (DBCOMMIT=10);
  SET mysas.staff;
RUN;
```

## See Also

- [“ERRLIMIT” on page 115](#)
- [“LIBNAME Options” on page 121](#)

---

## DBCONDITION

Specifies criteria for subsetting and ordering data.

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

**Default:** 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 option is ignored when you use DBCONDITION.

## See Also

[“DBKEY” on page 109](#)

---

## DBCREATE\_TABLE\_OPTS

Specifies data source-specific syntax to add to the CREATE TABLE statement.

**Valid in:** DATA and PROC steps

**Default:** 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” on page 121](#).

---

## DBENCODING

Specifies the encoding character set to use in the Microsoft Access database or in the Microsoft Excel workbook.

**Valid in:** DATA and PROC steps.

**Default:** NONE

---

## Syntax

**DBENCODING**=12-byte SAS encoding-value

### Syntax Description

The 12-byte SAS encoding-value is an encoding value that SAS defines. It can be up to 12 characters long. For a list of valid values, see the section “Encoding Values in SAS Language Elements” in *SAS National Language Support Reference Guide*.

### Details

Specify the encoding character set to use in your Microsoft Access database or Excel workbook file. This option enables SAS to transcode character data between the SAS session encoding and the DBENCODING value.

It is recommended that you use UNICODE=YES instead of DBENCODING. Setting UNICODE=YES is equivalent to setting DBENCODING='UTF-16'.

This enables SAS to bind text in wide character format. SAS is also enabled to transcode data between a SAS session. For example, Chinese BIG5 encoding, and Access or Excel in UNICODE encoding. In a UTF-8 session, SAS assumes that UNICODE=YES.

*Transcoding* is a process that converts text data from one encoding to another encoding.  
Note

- It does not translate from one language to another language.
- It does not translate the English language to the Japanese language.
- It does not translate the Chinese language to the Japanese language.
- It might map some Chinese characters to Japanese Hanzi characters.

---

## DBFORCE

Specifies whether to force the truncation of data during insert processing.

**Valid in:** DATA and PROC steps

**Default:** 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 width.

**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 disallowed characters.

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

**Default:** 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 `_COLn`, 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. The English characters converted from kanji are often not allowed in SAS. If you specify `DBGEN_NAME=SAS`, a column named `DEPT$AMT` is renamed to `_COLn` where *n* is the column number. If you specify `DBGEN_NAME=DBMS`, a column named `DEPT$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](#)” on page 121.

---

## DBKEY

Improves performance for a join with a large source table and a small SAS data set (specifies a column to use as an index).

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

**Default:** 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

**Default:** 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 a Microsoft Access or a Microsoft Excel file.

### Example: Specify Label Use

In this 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. When DBLABEL=YES, you can use the label 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

**Default:** 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 typically used with a very long character data type.

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

### See Also

To assign this option to a group of tables, use the DBMAX\_TEXT= option specified in [“LIBNAME Options” on page 121](#).

---

## DBNULL

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

<b>Valid in:</b>	DATA and PROC steps (when accessing PC files data using SAS/ACCESS software)
<b>Default:</b>	YES

---

### Syntax

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

### Syntax Description

#### YES

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

#### NO

specifies that a 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. 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:** only the Access engine supports this option. The Excel engine does not support this option.

### Example: Specify NULL Value Disposition

In this example, using the DBNULL option prevents the EmpId and Jobcode columns in the new MyDBLib.MyDept2 table from accepting null values. If the Employees table contains any 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 this 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 any null values in any column other than the Jobcode column, the DATA step fails.

```
DATA mydblib.mydept3 (DBNULL1=( _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:** 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, use DBNULLKEYS=YES. When you specify DBNULLKEYS=YES, and specify a column that is not defined as DBKEY=NOT NULL, SAS generates a WHERE clause that finds NULL values.

If you specify DBKEY=COLUMN and COLUMN is not defined as NOT NULL, SAS generates a WHERE clause.

Example:

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

SAS generates the WHERE clause once and uses it for any value, NULL, or NOT NULL in the column. This syntax can be much less efficient than the shorter form of the WHERE clause. When you specify DBNULLKEYS=NO, or specify a column that is NOT NULL in the DBKEY= option, SAS generates a simple WHERE clause.

If there are no NULL values in the transaction or master table for the columns, use DBNULLKEYS=NO. If you specify DBNULLKEYS=NO and DBKEY=COLUMN, SAS generates a shorter form of the WHERE clause. SAS generates the WHERE clause even if the column DBKEY specifies is defined as NOT NULL.

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

### See Also

- [“DBKEY” on page 109](#)
- [“LIBNAME Options” on page 121](#)

---

## 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:** 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 writing data into SAS from a 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:** 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 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.

*Note:* 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

**Default:** none

---

## Syntax

**DBTYPE**=(*<column-name-1>* =<*data-source-type*> ...  
*<column-name-1>* =<*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 it writes 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.

### Example: Specify the Data Type to Use

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;
```

---

## ERRLIMIT

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

**Valid in:** DATA and PROC steps

**Default:** 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 nonzero value for the DBCOMMIT= option, rollbacks affected by the ERRLIMIT= option might not be complete. Records already committed by DBCOMMIT= option are not processed again.

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

### Example: Specify Error Limit

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;
```

### See Also

[“DBCOMMIT” on page 105](#)

---

## INSERT\_SQL

Determines the method to use to insert rows into a data source.

**Valid in:** DATA and PROC steps

**Default:** 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” on page 121](#).

---

## INSERTBUFF

Specifies the number of rows in a single insert.

**Valid in:** DATA and PROC steps

**Default:** 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 greater than INSERTBUFF=1, the SAS notes indicating success or failure of the insert operation might be incorrect. These notes are generated for a single insert. This is also true, when multiple inserts are performed.

*Note:* PC Files Server does not support INSERTBUFF= option with a value higher than 1 for writing data to Excel. It ignores this option when writing data to Excel.

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

*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

- [“DBCOMMIT” on page 105](#)
- [“Overview of Data Set Options” on page 103](#)

---

## NULLCHAR

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

**Valid in:** DATA and PROC steps

**Default:** SAS

---

## Syntax

NULLCHAR= YES | NO

## Syntax Description

### 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.

in conjunction with the NULLCHARVAL= data set option, NULLCHARVAL= 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.

## See Also

[“DBKEY” on page 109](#)

---

## NULLCHARVAL

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

**Valid in:** DATA and PROC steps

**Default:** a blank character

---

## Syntax

NULLCHARVAL=<'character-string'>

## Details

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

This option works with the NULLCHAR= option. NULCHAR= determines whether a SAS character NULL value is treated as a NULL value.

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

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

## See Also

["DBKEY=" on page 109](#)

---

## READBUFF

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

**Valid in:** DATA and PROC steps

**Default:** 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. Because SAS stores the rows in memory, higher values for READBUFF= use more memory. If too many rows are selected at once, then the rows that are returned to the SAS application can 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 as specified in [“LIBNAME Options” on page 121](#).

---

## SASDATEFMT

Changes the SAS date format of a data source column.

**Valid in:** DATA and PROC steps

**Default:** 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 informat. For example, DATETIME21.2 is both a SAS format and informat, so it is valid 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 column, convert the SAS date values to appropriate values. The SAS DATEFMT= option enables you to convert date values from a SAS date format to different SAS date format.

Use SAS DATEFMT= to prevent date type mismatches under these 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 SAS DATEFMT= option does not currently handle these date types. You can use a PROC SQL view to convert the source data to a SAS date format, as you retrieve the data. You can also use a format statement in other contexts.



## Chapter 16

# Pass-Through Facility: PCFILES on Linux, UNIX, and Microsoft Windows

---

<b>Overview: Pass-Through Facility for PCFILES on Linux, UNIX, and Microsoft Windows</b> . . . . .	<b>205</b>
<b>Dictionary</b> . . . . .	<b>206</b>
Syntax for the Pass-Through Facility for PCFILES . . . . .	206
CONNECT Statement . . . . .	206
DISCONNECT Statement . . . . .	212
EXECUTE Statement . . . . .	213
CONNECTION TO Component . . . . .	214

---

## Overview: Pass-Through Facility for PCFILES on Linux, UNIX, and Microsoft Windows

The SQL procedure implements the Structured Query Language (SQL) for SAS. See the *SAS SQL Procedure User's 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.

Using the pass-through facility, you can do the following:

- Establish and terminate connections with a data source using the CONNECT and DISCONNECT statement.
- Send dynamic, non-query, data source-specific SQL statements to a data source using the EXECUTE statement [“EXECUTE Statement” on page 137](#).
- Retrieve data directly from a data source using the [“CONNECTION TO Component” on page 139](#).

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.

---

## Dictionary

---

### Syntax for the Pass-Through Facility for PCFILES

Queries data from a data source.

---

#### Syntax

**PROC SQL** *option(s)*

**CONNECT TO** *data-source-name AS alias*

( **CONNECT** *statement-arguments*

**DATABASE** *connection-arguments* )

**DISCONNECT FROM** *data-source-name*

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

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

#### Details

You can use the component with the PROC SQL SELECT statement to query data from a data source.

#### 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 statement

---

#### Syntax

**CONNECT TO** *data-source-name*

**CONNECT TO** *option(s)*

### **Optional Arguments**

#### **data-source-name**

Specifies the data source to which you want to connect. Because this method requires connecting through a 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 connections, unique connections, and so on) to the database.

#### **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. The default behavior opens a dialog box with prompts to specify connection information.

## **Details**

### **Overview**

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. See [“DISCONNECT Statement” on page 137](#).

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 CONNECT statement with the PC files engine name and then assign an alias if you want.
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 you do not specify it, default values for all 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 130](#) for more information about these macro variables.

### **Database Statement Arguments**

The arguments that are listed below are available with the pass-through facility for PC files. These arguments provide information to the pass-through facility to connect to the PC files or to the database. These options are used when connecting to PC Files Server.

#### **DSN= data-source-name**

specifies the ODBC data source name that is used to access the PC data through 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 semicolons. This is an advanced connection method that you should use only when you know the exact syntax of all connection options that the ODBC driver requires for a successful connection.

**PATH= *path-for-file***

specifies the data source file location for the Microsoft Access database file or Microsoft Excel workbook file.

**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.

**Alias:** SERVICE | SERVICE\_NAME

**Default:** 9621

**SERVER= *pc-server-host-name***

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.

You can specify this host name as a simple computer name (for example, `wxp320`), a fully qualified network name (for example, `wxp320.domain.com`), or an IP address.

**Note:** Omitting the SERVER= option on Microsoft Windows clients invokes Autostart.

**SERVERUSER= '*domain\server-user-name*'**

specifies the domain name and User ID for the PC running PC Files Server. Always enclose the value in quotation marks. Otherwise, the backslash can be misinterpreted by the SAS parser.

**Alias:** SERVERUID

**Notes:**

If you are not on a domain, omit the domain name and the backslash.

Use the USER= option for database user IDs.

**SERVERPASS= '*server-user-password*'**

specifies the password for the PC Files Server for the user ID given. If the account has no password, omit this option. Always enclose the value in quotes in order to preserve the case of the password.

**Alias:** SERVERPASSWORD | SERVERPW | SERVERPWD

**Notes:**

Passwords are generally case sensitive.

Use the PASSWORD= option for database passwords.

**SSPI= *YES* | *NO***

enables the PC Files Server to allow Integrated Windows Authentication. This is a mechanism for the Windows client and server to exchange credentials.

**Default:** NO

**Restriction:** Microsoft Windows 64-Bit only.

**Note:** SSPI can also be enabled by specifying the `–SSPI` option on the SAS command line.

**DBPASSWORD= *database-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 you can define it instead of user-level security.

**Restriction:** Microsoft Access only.

**DBSYSFILE= *workgroup-information-file***

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

**PASSWORD= *user-password***

specifies a password for the user account, if required by the data source. Passwords are case sensitive.

**MSENGINE= ACE | JET**

determines the database engine used for accessing the Microsoft Excel file or Microsoft Access database. The Microsoft Jet engine is older and supports formats up to 2003. The Microsoft ACE engine supports Microsoft Excel 2007 and Microsoft Access 2007 and older formats.

**Default:** ACE

**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 access your file.

**VERSION= 2007 | 2003 | 2002 | 2000 | 97 | 95 | 5**

sets the version of Microsoft Excel workbook. The default value is 97.

**Alias:** VER

**Restriction:** Microsoft Excel only.

**Note:** You do not need to specify this option for an existing Microsoft Excel file. If you want to create a new Microsoft Excel workbook file, you can use this option to specify the version that you want to create. Note that versions 97, 2000, and 2003 of Excel share the same file format. Versions 95 and 5 share a separate file format.

### **CONNECT Statement Arguments**

Connect Statement arguments are supported by 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 they are submitted.

**YES**

specifies that updates are committed (saved) to the table as soon as they are submitted. 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 before a data source command times out.

**Alias:** TIMEOUT

**Default:** 0 (no time-out)

**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.

- The CONNECT statements must use identical values for the CONNECTION= option.
- The CONNECT statement must use identical values for the CONNECTION\_GROUP= option.
- Database connection arguments must be identical.

**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= *DYNAMIC* | *FORWARD\_ONLY* | *KEYSET\_DRIVEN* | *STATIC***

specifies the cursor type for read-only cursors and for cursors to be updated.

**DYNAMIC**

specifies that the cursor reflects all 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 supports only 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.

**Alias:** CURSOR

**Default:** None

**DBGEN\_NAME= DBMS | SAS**

specifies that the data source columns are renamed, and specifies the format that the new names 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 applies only when you are reading, appending, and updating Microsoft Access or Excel character data from SAS.

**Default:** 1,024

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

**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

**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. In addition, if too many rows are read at once, values returned to SAS might be out of date.

**Alias:**

ROWSET=

ROWSET\_SIZE=

**Default:** 1

**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.

**Alias:** STRDATES

**Default:** NO

**USEDATE= YES | NO**

specifies whether to assign the DATE. format or the DATETIME. format for datetime columns in the data source table while importing data from a Microsoft Access database or a Microsoft Excel workbook.

**YES**

specifies that the DATE. format is assigned to datetime columns in the data source table.

**NO**

specifies that the DATETIME. format is assigned for datetime columns in the data source table.

**Alias:** USE\_DATE | USE\_DATATYPE

**Default:** NO

**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**

Ends the connection to the data source.

**Valid in:** SQL procedure.

---

**Syntax**

**DISCONNECT FROM** *<data-source-name>* *<alias>*

**Syntax Description*****Data-source-name***

specifies the *data-source-name* from which you want to disconnect. The DISCONNECT statement's *data-source-name* must match the *data-source-name* that you specified in the CONNECT statement.

***Alias***

specifies the data source *alias* from which you want to disconnect. The DISCONNECT statement's *alias* must match the *alias* that you specified 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 the procedure ends. The SQL procedure continues to execute until you submit a QUIT statement, a SAS procedure, or a DATA step.

The contents of the SQLXRC and SQLXMSG macro variables can be written to the SAS log using the macro. The contents are reset after each pass-through facility statement is executed.

See “Return Codes” on page 130 for additional information.

## Example: Disconnect and Quit

SQL processing uses the DISCONNECT statement to end the connection with the database. Use the QUIT statement to quit the SQL procedure after the connection ends:

```
DISCONNECT FROM db;
QUIT;
```

---

## EXECUTE Statement

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

**Valid in:** SQL procedure steps.

---

### Syntax

```
EXECUTE (data-source-specific-SQL-statement(s))
  BY
  <data-source-name> <alias>
```

### Syntax Description

#### *Data-source-specific-SQL-statements*

specifies a dynamic non-query, data source-specific SQL statement. Depending on your data source, the SQL statement can be case sensitive. The statement is passed to the data source exactly as you type it.

#### 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

drops 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

inserts rows to a data source table.

#### REVOKE

revokes the access or modification privileges that were given to users by the GRANT statement.

#### UPDATE

updates the data in the specified columns of a row in a data source table.

#### Requirements:

At least one statement is required.

The statement must be enclosed in parentheses.

**Data-source-name**

specifies the *data-source-name* to which you direct the data source-specific SQL statements. The EXECUTE statement's *data-source-name* must match the *data-source-name* specified in the CONNECT statement.

**Alias**

specifies the data source *alias* that was defined in the CONNECT statement. The EXECUTE statement's *alias* must match the *alias* that you specified in the CONNECT statement.

**Details**

The EXECUTE statement sends dynamic non-query, 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 an SQL view.

The contents of the SQLXRC and SQLXMSG macro variables can be written to the SAS log using the macros. The contents are reset after each pass-through facility statement is executed.

**Example: Drop and Create a Table and Insert a Data Row**

Use the EXECUTE statement to drop a table, create a table, and insert a row of data after the connection:

```
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:** SQL procedure STEP statements.

---

**Syntax**

**CONNECT TO** a *data-source AS alias* (*connect statement arguments*)  
(*database connection-options*)

**Summary of Optional Arguments**

[ALIAS](#)

[DATABASE CONNECTION ARGUMENTS](#)

[DATA SOURCE NAME](#)

[CONNECTION COMPONENT](#)

**Optional Arguments****ALIAS**

specifies the data source alias for the connection. If you specify an alias, the keyword AS must appear before the alias.

**Restriction:** ALIAS is not supported if the CONNECT statement is omitted.

**Requirement:** The range of the ALIAS is between 1 and 32 characters.

**Note:** The data source name is used as the name of the pass-through connection if an alias is not specified.

**CONNECTION COMPONENT**

specifies arguments that indicate whether you can make multiple connections, shared connections, or unique connections, to the database.

**DATA SOURCE NAME**

specifies the data source name to which you want to connect and direct the data source-specific SQL statements.

**Requirement:** You must use back quotation marks ( ` ), not single (forward) quotation marks, to enclose any data source name that contains a space.

**Note:** The data source name becomes the name of the pass-through connection if an alias is not specified.

**DATABASE CONNECTION ARGUMENTS**

specifies the data source-specific arguments to the pass-through facility that are needed by the SQL procedure to connect to a data source.

**Database Connection Arguments**

Connection arguments provide database connection information to the pass-through facility to connect to a Microsoft Access database or a Microsoft Excel workbook file.

**INIT= "initialization-string"**

specifies the initialization string when connecting to a data source.

**Note:** This statement option applies to the INIT= option and the UDL= option.

**MSENGINE= ACE | JET**

determines the database engine to use for accessing Microsoft Excel files or Microsoft Access databases. The Microsoft Jet engine supports Microsoft formats up to 2003. The Microsoft Ace engine supports 2007 formats and formats in subsequent releases of Windows.

**Default:** ACE

**Restriction:** It is recommended that this option is used to create only a Windows 95 format file.

**PATH= data-source-path**

specifies the path of the Microsoft Access database or the Microsoft Excel workbook file.

**PROMPT= YES | NO | REQUIRED | NOPROMPT | PROMPT | UDL**

specifies whether user is prompted for data source connection information.

YES enables prompting with a Data Link Properties dialog box. To write the initialization string to the SAS log, submit this code immediately after connecting to the data source:

```
%PUT %SUPERQ (SYSDBMSG) ;
```

NO prompting is not available. You must specify the data source as a physical filename or complete path.

REQUIRED connect with a valid data-source-name. If a valid connection is not specified, you are prompted for the connection options. The prompt enables you to change the data source file and other properties.

NOPROMPT disables the display of the Data Link Properties window. Prompting is not available.

PROMPT enables the display of the Data Link Properties window. Prompting is available.

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

*Note:* This statement option applies to the INIT= argument and the UDL= argument.

**UDL= "path and filename"**

specifies the path and filename for a UDL (a Microsoft data link file). This option does not support SAS filerefs. The macro variable SYSDBMSG is set upon successful completion.

```
UDL_FILE='C:\WinNT\profiles\me\desktop\MyDBLink.udl';
      %PUT %SUPERQ(SYSDBMSG);
```

**Alias:** UDL\_FILE

**See:** Microsoft Data Link API documentation.

**Additional Options for Microsoft Access Database Only**

**DBPASSWORD= *database-file-password***

enables you to access database files with database-level security. This security level can be defined instead of user-level security.

**Alias:** DBPWD | DBPW | PASS | PASSWORD

**Restriction:** Microsoft Access Database only.

**Note:** Database password is case sensitive.

**DBSYSFILE= *workgroup-information-file***

specifies the workgroup information file. This file contains a collection of information defined for the Microsoft Access database. User, group accounts, and passwords that you create are saved in the workgroup information file.

**Alias:** SYSTEMDB

**Restriction:** Microsoft Access database files only.

**PASSWORD= *user-password***

specifies a password required by the data source for the user account.

**Alias:** PWD | PW | PASS | PASSWORD

**Note:** Passwords are case sensitive.

**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 on your .mdb file, you need to use the USER= and PASSWORD= options to access your file.

**Alias:** UID

**Restriction:** Microsoft Access database files only.

**Note:** Use the SERVERUSER= option to connect to a server.

## Details

The CONNECTION component specifies the data source connection to use or to create. CONNECTION enables you to retrieve data source data directly through an SQL procedure query.

- The CONNECTION component can be used in any FROM clause, including those in nested queries (subqueries).
- You can store a pass-through facility query in an SQL view and then use that view in SAS programs.
- When you create an SQL view, any options that you specify in the corresponding CONNECTION statement are stored too. Thus, when the 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 component.

## Example: Connect and Query a Table

Use the CONNECTION component to query a table or a subtable after the connection:

```
SELECT * FROM CONNECTION TO db(SELECT * FROM `my invoice`);  
SELECT * FROM CONNECTION TO db  
(SELECT `Invoice Number`, Amount from `my invoice`);
```



## Chapter 17

## Special Query Support

---

Special PCFILES Queries .....	219
-------------------------------	-----

---

## Special PCFILES Queries

SAS/ACCESS Interface to PC Files on UNIX supports special queries. 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). This is done in order to accommodate databases that are not structured as SQL tables. You can use these special queries on non-SQL and SQL databases. The general format of special queries is as follows:

**PCFILES:: SQLAPI** *'parameter 1,... parameter n'*

**PCFILES::**

is case sensitive and is required to distinguish special queries from regular queries.

**SQLAPI**

is case sensitive and is the specific API that is being called.

**' parameter n'**

is a series of quoted strings 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.
- An 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`.

Depending on the DBMS that you are using, valid values for the Schema argument might be a user ID, a database name, or a library. All arguments are optional. If you specify some but not all 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.

**PCFILES:: SQLTables** < "Catalog", "Schema", "Table-name", "Type" >  
 returns a list of all 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 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 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", "FK-schema", "FK-table-name" >

returns a list of all 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 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 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 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 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.

The following example connects to a Microsoft Excel workbook. The example does the following:

- lists the columns of Sheet1\$
- lists the available type info
- creates a SAS data set with the list of tables (sheets) in the Microsoft Excel workbook sheet

- the PRINT procedure prints the created data set

PC Files Server must be running on the server specified for this example to work.

```
PROC SQL;
  CONNECT TO PCFILES AS DB (SERVER=d1234 PATH='c:\xl.xls' );
  SELECT * FROM CONNECTION TO DB (PCFILES::SQLColumns "","","Sheet1$","");
  SELECT * FROM CONNECTION TO DB (PCFILES::SQLGetTypeInfo);
  CREATE TABLE work AS SELECT * FROM CONNECTION TO DB
    (PCFILES::SQLTables "","","","");
QUIT;
PROC PRINT DATA=work;
RUN;
```



## Part 5

---

# ACCESS and DBLOAD Procedures

<i>Chapter 18</i>	
<b>The ACCESS Procedure for PC Files</b> .....	225
<i>Chapter 19</i>	
<b>The DBLOAD Procedure</b> .....	249
<i>Chapter 20</i>	
<b>File-Specific Reference for the ACCESS and DBLOAD Procedures</b> . .	259



## Chapter 18

# The ACCESS Procedure for PC Files

---

<b>Overview: The ACCESS Procedure for PC Files</b> . . . . .	<b>226</b>
Overview: ACCESS Procedure . . . . .	226
Using ACCESS Procedure Statements . . . . .	226
<b>SAS/ACCESS Descriptors for PC Files</b> . . . . .	<b>227</b>
Overview . . . . .	227
Access Descriptors . . . . .	227
View Descriptors . . . . .	228
Extracting Data Using a View . . . . .	228
Tasks and Associated Statements . . . . .	229
<b>Syntax: The ACCESS Procedure for PC Files</b> . . . . .	<b>229</b>
PROC ACCESS Statement . . . . .	230
ASSIGN Statement . . . . .	231
CREATE Statement . . . . .	231
DROP Statement . . . . .	234
FORMAT Statement . . . . .	235
LIST Statement . . . . .	236
MIXED Statement . . . . .	237
PATH Statement . . . . .	237
QUIT Statement . . . . .	238
RENAME Statement . . . . .	239
RESET Statement . . . . .	240
SELECT Statement . . . . .	241
SUBSET Statement . . . . .	241
TYPE Statement . . . . .	242
UNIQUE Statement . . . . .	242
UPDATE Statement . . . . .	243
<b>SAS Passwords for Descriptors</b> . . . . .	<b>246</b>
<b>Performance and Efficient View Descriptors for PC Files</b> . . . . .	<b>246</b>
General Guidelines . . . . .	246
Extracting Data Using a View . . . . .	247

---

## Overview: The ACCESS Procedure for PC Files

### Overview: ACCESS Procedure

This enables you to directly read, write, or extract PC files data into a SAS data set. The descriptor files are compatible with SAS 6. The ACCESS procedure can be used with Microsoft Excel (4, 5, 95), Lotus 1-2-3 (WK1, WK3, WK4), DBF, and DIF file formats.

**CAUTION:**

**The ACCESS Procedure for PC Files is obsolete.** These procedures are generally not used anymore and are not supported. They are documented here for the sake of completeness. You are encouraged to rely on more recent functionality to produce similar results.

The ACCESS procedure enables you create access descriptors, view descriptors, and SAS data files. Descriptor files describe PC files data to enable you to directly read, update, or extract PC files data while working within a SAS program.

See [“SAS/ACCESS Descriptors for PC Files” on page 227](#).

**CAUTION:**

**Altering a PC file might invalidate defined descriptors.** Altering the format of a PC file that has descriptor files defined for it might cause the descriptors to be out-of-date or invalid. If you add a column to a file and an existing access descriptor is defined for that file, the existing access descriptor and view descriptors do not show the new column. To show and select the new column, you can recreate the descriptors.

The are compatible with SAS 6. The view descriptor saves a column name with up to eight characters in uppercase. Any column name longer than eight characters is truncated. When duplicate names occur after truncation, a unique name is generated with a number appended to it. The view descriptor saves full column names as is in the label fields. Full column names are read from the SAS data set variable labels.

### Using ACCESS Procedure Statements

The following table presents a task-oriented overview of the ACCESS Procedure statements. The statements enable you to create or modify access and view descriptors. See ["ACCESS Procedure Syntax" on page 229](#) for the complete syntax for this procedure.

**Table 18.1** ACCESS Procedure Options and Statements

Task	Options and Statements to Use
create an access descriptor	<pre>PROC ACCESS DBMS=DBF   DIF   WKn   XLS; CREATE libref.member-name.ACCESS;     required-database-description-statements;     optional-editing-statements; RUN;</pre>

---

Task	Options and Statements to Use
create an access descriptor and a view descriptor	<pre>PROC ACCESS DBMS=DBF DIF WKn  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;</pre>
create a view descriptor from an existing access descriptor	<pre>PROC ACCESS DBMS=DBF DIF WKn XLS ACCDISC=libref.access-descriptor; CREATE libref.member-name.VIEW; SELECT column-list;     optional-editing-statements; RUN;</pre>

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 231 for additional information.

---

## SAS/ACCESS Descriptors for PC Files

### Overview

There are two types of descriptor files: access descriptors and view descriptors.

### 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 describes only a PC file 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. Instead, use an access descriptor to create other SAS files, called view descriptors, that you use in SAS programs.

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.

SAS/ACCESS descriptor files are the tools that the ACCESS procedure uses to establish a connection to a PC file. There are two types of descriptor files: access descriptors and view descriptors. Use the ACCESS procedure to create descriptors.

## **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 that 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 view descriptor data with SAS data, for example.

You can use a view descriptor to update data directly in some of the PC file formats, such as the DBF file format.

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.

When you create a view descriptor, the RESET statement clears any columns that were included in the SELECT statement.

When creating the view descriptor, if you reset a SAS variable and select it again within the same procedure execution, the SAS variable names and formats are reset to their default values. The defaults 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.

If you specified ASSIGN= YES when you created the access descriptor, the RESET = option has no effect on the view descriptor.

Use a view descriptor to update data directly in some of the PC file formats, such as the DBF file format.

Create a SAS data set from data stored in a PC file. When a view descriptor is used to copy PC files data into a SAS data set is called extracting the data.

## **Extracting Data Using a View**

It might be more efficient to use a view descriptor to extract PC files data and place it in a SAS data file.

A PC file is read every time a view descriptor is referred to in a SAS program and is executed. It is better to extract data under these circumstances:

- If the file is large and you use the data repeatedly in SAS programs.
- If you use sorted data several times in a SAS program.

- For added security, you can assign a password to the extracted SAS data file.

### Tasks and Associated Statements

“File Format-Specific Reference for the IMPORT and EXPORT Procedures” on page 29 for additional information.

**Table 18.2** ACCESS Procedure Tasks

Task	Options and Statements
create an access descriptor	<pre> <b>PROC ACCESS</b>   <b>DBMS=</b> DBF   DIF   WK <i>n</i>   XLS;   <b>CREATE</b> <i>libref.member-name</i>;   <b>ACCESS</b> <i>database description-statements (required) edit (options)</i> ; <b>RUN;</b> </pre>
create an access descriptor and a view descriptor	<pre> <b>PROC ACCESS</b>   <b>DBMS=</b> DBF   DIF   WK <i>n</i>   XLS;   <b>CREATE</b> <i>libref.member-name</i>   <b>CREATE</b> <i>libref.member-name</i>. VIEW;   <b>ACCESS</b> <i>database description-statements (required) edit (options)</i> ;   <b>SELECT</b> <i>column-list edit (options)</i> ; <b>RUN;</b> </pre>
create a view descriptor from an existing access descriptor	<pre> <b>PROC ACCESS</b>   <b>DBMS=</b> DBF   DIF   WK   <i>n</i>   XLS   <b>ACCDESC=</b> <i>libref.access-descriptor</i>;   <b>CREATE</b> <i>libref.member-name</i>. VIEW;   <b>SELECT</b> <i>column-list edit (options)</i> ; <b>RUN;</b> </pre>

*Note:* 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.

---

## Syntax: The ACCESS Procedure for PC Files

```

PROC ACCESS
  DBMS= DBF | DIF | WK n | XLS;
  CREATE libref.member-name;
  ACCESS database description-statements (required) edit (options);
RUN;

```

---

## PROC ACCESS Statement

Access data from PC files.

**Requirement:** This statement is required.

**Supports:** DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 files under Windows operating system.

---

### Syntax

```
PROC ACCESS <option(s)>
```

### Optional Arguments

**DBMS=** *pc-file-format*

specifies the 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<sub>n</sub> files, or DBMS=XLS for XLS files.

**Valid in:** PROC SQL

**Restriction:** The ACCESS Procedure is supported only in V6 SAS.

**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. 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.

**READ**

specifies permission to read an existing access descriptor.

**WRITE**

specifies permission to write to an existing access descriptor.

**ALTER**

specifies permission to alter an existing access descriptor.

**Alias:** AD= and ACCESS=

**VIEWDESC=** *libref.view-descriptor*

specifies a view descriptor as input for the OUT= option.

**See:** 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= option.

```
PROC ACCESS
    VIEWDESC=vlib.invg4
    OUT=dlib.invg4;
RUN;
```

**See:**

[Chapter 5, “The EXPORT Procedure,” on page 21](#)

[Chapter 4, “The IMPORT Procedure,” on page 13](#)

---

## ASSIGN Statement

Indicates whether SAS variable names and formats are automatically generated.

---

### Syntax

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 eight 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 231](#) for additional information.

You can use the value NO (or N) to modify SAS variable names and formats when you create an access descriptor. Use NO (or N) when you create view descriptors that are based on this access descriptor. When creating an access descriptor, use the RENAME statement to change SAS variable names. 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 eight characters of the PC file column names. 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 the variable names in the view descriptors.

SAS variable names are generated as follows:

- If the column name is longer than eight characters, SAS uses only the first eight characters. If truncating results in duplicate names, numbers are appended to the ends of the names to prevent duplicate names.
- If the column name in the PC file contains blank characters, SAS ignores it.
- If the column name in the PC file starts with a digit (0 through 9), SAS adds the character Z before it.
- If the column name contains characters that are invalid in SAS names (including national characters), SAS replaces the invalid characters with underscores (\_).

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.

**Note:** for DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments access descriptor or view descriptor

---

## Syntax

```
CREATE libref.descriptor-name. ACCESS | VIEW
```

## Details

### Overview

Use CREATE to create an access or view descriptor for a PC file that 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 library, which is associated with a directory on the local system disk where the descriptor is created. The *libref* must have been created already 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 the CREATE and the UPDATE in the same PROC ACCESS block with one restriction: a CREATE statement for a view descriptor should 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 the CREATE and the UPDATE statements, either statement can be the first in the block.
2. 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. 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. 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. If you recreate a file and add a new column to the file, an existing access descriptor defined does not show that column, but the descriptor can still be valid. If you recreate a file and delete an existing column from the file, the descriptor is invalid. If the deleted column is included in a view

descriptor that 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, place the statements or groups of statements in the order as follows:

1. Create the access descriptor as described in “[Creating Access Descriptors](#)” on page 232, except omit the RUN statement.
2. 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. 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. You can specify FORMAT, RENAME, and UNIQUE only when you specify ASSIGN= NO in the access descriptor that this view descriptor references. QUIT is also an editing statement. However, if you use it, it terminates PROC ACCESS without creating your descriptor.

Statement order within this group usually does not matter. See the individual statement descriptions for any restrictions.

4. Specify the RUN statement. RUN executes PROC ACCESS.

To create a view descriptor based on an access descriptor created in a separate PROC ACCESS step, specify the name in the ACCDESC= option. 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 any statements are checked for errors when you submit PROC ACCESS. If you create multiple descriptors in the same procedure, each CREATE statement and its statements are also checked for errors.

If no errors are found when the RUN statement is processed, all descriptors are saved. If errors are found, they are written to the SAS log, and processing is terminated.

After you correct the errors, resubmit your statements.

## Examples

### Example 1: Create an Access Descriptor for a Worksheet File

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

PROC ACCESS DBMS=WK4;
  CREATE adlib.product.access;
  PATH='c:\sasdmo\specprod.wk4';
  GETNAMES=yes;
  ASSIGN=yes;
  RENAME= productid prodid
         fibername fiber;
  FORMAT productid 4.
         weight     e16.9
```

```

        fibersize e20.13
        width      e16.9;
RUN;

```

### **Example 2: Create an Access Descriptor for a Microsoft Excel Worksheet**

This 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;

```

### **Example 3: Create a View Descriptor from an Access Descriptor**

This example creates a view descriptor VLib.BDays from the AdLib.Employ access descriptor. It was created in the previous PROC ACCESS step. You could also use FORMAT 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.

**Note:** for DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments access descriptor, view descriptor RESET, SELECT, UPDATE

---

## Syntax

```
DROP 'column-identifier1' ... 'column-identifierN';
```

## 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.

You can specify the DROP statement only when you create or update an access descriptor or when you update a view descriptor. DROP is not allowed when you create 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 231](#) for additional information.

The *column-identifier* argument can be the column name or the positional equivalent from the LIST statement. This is the number that represents column placement in the access descriptor or view descriptor. To drop the third and fifth columns, submit this 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 column attributes (such as the SAS variable name format) to their default values.

---

## FORMAT Statement

Changes a SAS format for a PC file column.

- Notes:** When you use the FORMAT statement with access descriptors, the FORMAT statement also re-selects columns that were previously dropped with the DROP statement.
- for DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments access descriptor or view descriptor ASSIGN, DROP, RESET

---

## Syntax

```
FORMAT | FMT 'column-identifier1' SAS format-name 'column-identifierN' SAS format-name
```

## Details

The Format Statement changes a SAS format for a PC file column.

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 231 for additional information .

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

---

## LIST Statement

Lists columns in the descriptor and gives information about them.

**Default:** ALL

**Note:** for DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments access descriptor or view descriptor

---

## Syntax

```
LIST ALL | VIEW 'column-identifier'
```

## Required Arguments

### ALL

lists all 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 columns that are selected for the view descriptor, along with their positional equivalents, their SAS names and formats, and any sub-setting 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. This 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 this statement: **LIST 5;**

## 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 information about the descriptor that you are creating.

The LIST statement can take one or more of these arguments:

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.

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

**Note:** For WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments

---

## Syntax

MIXED= YES | NO | Y | N

## Details

You use the MIXED= option 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 XLS Files” on page 270](#) for additional information.

---

## PATH Statement

Specifies the path and filename of the file to access.

**Requirement:** This statement is required.

**Note:** for DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, Excel 95 file formats under Windows operating environments access descriptor

---

## Syntax

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

### Required Arguments

**PATH= *'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.

**PATH= *'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.

**PATH= *fileref***

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

### Details

The PATH= statement indicates the path and name of the file that 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 231](#) for additional information.

---

## QUIT Statement

Terminates the procedure.

**Alias:** EXIT

**Note:** for DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments access descriptor or view descriptor

---

## Syntax

QUIT;

### Details

The QUIT statement terminates the ACCESS procedure and descriptor creation.

---

## RENAME Statement

Modifies the SAS variable name.

**Note:** For DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments, access descriptor or view descriptor ASSIGN, RESET

---

### Syntax

```
RENAME= 'column-identifier1' 'SAS variable-name1' , 'column-identifierN' '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 231](#) for additional information.

Two factors affect the use of the RENAME statement: whether you specify the ASSIGN statement when you are creating an access descriptor, and the type of descriptor that 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 a SAS/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. 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 column name or the positional equivalent from the LIST statement. This is the number that represents the column's place in the descriptor. To rename SAS variables that are associated with the seventh column and the nine-character FIRSTNAME column in a descriptor, submit this 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. If you rename the SAS variable associated with a column, you do not have to issue a SELECT statement for that column.

When creating an access descriptor, the RENAME statement also re-selects previously dropped columns that were dropped with the DROP statement.

---

## RESET Statement

Resets PC file columns to their default settings.

**Restriction:** Not allowed with UPDATE

**Interaction:** ASSIGN, DROP, FORMAT, RENAME, SELECT

**Notes:** For DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments, access descriptor or view descriptor, ASSIGN, DROP, FORMAT, RENAME, SELECT UPDATE  
Applies to access descriptor or view descriptor

---

### Syntax

```
RESET ALL | 'column-identifier1' ... 'column-identifierN'
```

### Required Arguments

#### ALL

for access descriptors, resets all PC file columns that are defined to their default names and format settings and re-selects any dropped columns.

For view descriptors, ALL resets all columns that are selected so that no columns are selected for the view. You can then use the SELECT statement to select new columns.

See the "[SELECT Statement](#)" on page 241.

#### *column-identifier*

can be either the column name or the positional equivalent from the LIST statement. This is the number that represents the column's place in the access descriptor. To reset the SAS variable name and format associated with the third column, submit this statement: **RESET 3;**

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

### Details

The RESET statement resets either the attributes of all 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 231 for additional information.

---

## SELECT Statement

Selects PC file columns for the view descriptor.

**Requirement:** This statement is required.

**Note:** For DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments view descriptor RESET UPDATE

---

### Syntax

**SELECT ALL** | '*column-identifier1*', '*column-identifierN*';

### 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 you can use it only when you create view descriptors. You cannot use the SELECT statement when you update 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 231 for additional information.

The SELECT statement can take one or more of these arguments:

**ALL**

includes in the view descriptor all 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. This is the number that represents the column's place in the access descriptor on which the view is based. To select the first three columns, submit this 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. Submit these SELECT statements to select columns 1, 5, and 6. **SELECT 1;** **SELECT 5 6;**

To clear all your selections when creating a view descriptor, use the RESET ALL statement. You can use another SELECT statement to select new columns.

---

## SUBSET Statement

Adds or modifies selection criteria for a view descriptor.

**Note:** for DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments view descriptor

---

## Syntax

**SUBSET** *selection-criteria*;

## Details

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 data (rows) in the PC file.

### CAUTION:

**An editing statement, such as SUBSET, must follow the CREATE statement when you create a view descriptor.**

See "[CREATE Statement](#)" on page 231 for additional information.

---

## TYPE Statement

Changes the expected data types of SAS variables.

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

---

## Syntax

**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.

**Alias:** UN

**Note:** for DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Windows operating environments view descriptor ASSIGN UPDATE

---

## Syntax

**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 the "[CREATE Statement](#)" on page 231 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 option, the UNIQUE= option cannot be used 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, resolve any duplicate SAS variable names in the view descriptor. Use the UNIQUE= option to generate unique names automatically, or you can use the RENAME= option to resolve duplicate names yourself.

See the "[RENAME Statement](#)" on page 239 for information about that statement.

If duplicate SAS variable names exist in the Access Descriptor that you are using to creating a View Descriptor, specify the UNIQUE= option to resolve the duplication. When you specify tUNIQUE= YES, the SAS/ACCESS interface appends numbers to any duplicate SAS variable names, thus making each variable name unique.

See the "[CREATE Statement](#)" on page 231 .

If you specify UNIQUE= NO, the SAS/ACCESS interface continues to allow duplicate SAS variable names to exist. To create the View Descriptor, resolve duplicate names before saving.

*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.

**Note:** For DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95, file formats under Windows operating environments access descriptor or view descriptor ASSIGN, RESET, SELECT, UNIQUE

---

## Syntax

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

## Details

### Overview

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 created previously by a LIBNAME statement.

#### *descriptor-name*

specifies the descriptor that you are updating, which already exists in *libref*.

#### ACCESS

indicates that you are updating an access descriptor while VIEW indicates you are updating a view descriptor.

Multiple UPDATE statements can 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:

Because the UPDATE block does not validate the updated descriptor, the order of description and editing statements does not matter.

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 can be the first in the block.
2. Data source description statements: All are allowed.
3. Editing statements: These editing statements are not allowed: ASSIGN, LIST, RESET, SELECT, VIEW.
  - The descriptor column name can be up to eight characters in uppercase.
  - Column names longer than eight characters are truncated to eight characters.
  - Duplicate names generated by the truncation, have an eight-character name where the eighth character is a number.
  - The view descriptor saves column names as is, in the labels.
  - Full column names are read from the SAS data set variable labels.

### Updating View Descriptors

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 can be the first in the block.

1. Data source description statements: All are allowed.

2. These editing statements are not allowed: ASSIGN, DROP, RESET, SELECT, and UNIQUE.

### Examples

#### Updating an Existing ACCESS Descriptor

This example updates 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;
```

#### Update an Access Descriptor and a View Descriptor

This example updates the Employ access descriptor and the View Descriptor for Employ.

```
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;
```

#### Update a Second View Descriptor

Update a second view descriptor that is based on Employ, named BDays, that is also located in 'C:\SASVIEWS'. When you update a view, it is not necessary to specify the access descriptor using the ACCDESC= option in the PROC ACCESS statement. Note that FORMAT can be used because the Employ access descriptor was created with ASSIGN= NO.

```
LIBNAME vlib 'C:\SASVIEWS';
PROC ACCESS DBMS=XLS;
  UPDATE vlib.bdays.view;
  FORMAT empid 6.
         birthdate datetime7.;

RUN;
```

---

## 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.

**Table 18.3** Password and Descriptor Interaction

Descriptor	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

In this example, the DATASETS procedure 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;
```

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

---

## 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.

### **Extracting Data Using a View**

It might be more efficient to use a view descriptor to extract PC files data and place it in a SAS data set.

- **A PC file is read every time a view Descriptor is referred to in a SAS program and is executed. It is better to extract data under these circumstances:**
- If the file is large and you use the data repeatedly in SAS programs.
- If you use sorted data several times in a SAS program.
- For added security, you can assign a password to the extracted SAS data set.



## Chapter 19

# The DBLOAD Procedure

---

<b>Overview: DBLOAD Procedure</b> . . . . .	<b>249</b>
What Does the DBLOAD Procedure Do? . . . . .	249
PROC DBLOAD Operating Environments . . . . .	250
PROC DBLOAD Naming Conventions . . . . .	250
PROC DBLOAD Statement . . . . .	250
<b>Syntax: The DBLOAD Procedure</b> . . . . .	<b>250</b>
PROC DBLOAD Statement . . . . .	251
ACCDESC Statement . . . . .	252
DELETE Statement . . . . .	252
ERRLIMIT Statement . . . . .	253
FMTLIB Statement . . . . .	253
LABEL Statement . . . . .	254
LIMIT Statement . . . . .	254
LIST Statement . . . . .	254
LOAD Statement . . . . .	255
PATH Statement . . . . .	256
QUIT Statement . . . . .	256
RENAME Statement . . . . .	256
RESET Statement . . . . .	257
WHERE Statement . . . . .	258

---

## Overview: DBLOAD Procedure

### *What Does the DBLOAD Procedure Do?*

The DBLOAD procedure loads data to PCs and creates PC files. This data can be from: a SAS data set, a PROC SQL view, a DATA step view, or a view descriptor from any SAS/ACCESS for Relational Databases interface product.

The DBLOAD procedure associates each SAS variable with a PC file column. Then it assigns a default name and data type to each column. Use the default information or change it as necessary. Once the columns are customized, the procedure creates the PC file and loads it with the input data.

### **PROC DBLOAD Operating Environments**

The DBLOAD procedure for PC files is available only under Microsoft Windows operating environments. You can use the DBLOAD procedure with DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats.

There are alternate ways for accessing data in PC file formats under the UNIX, Microsoft Windows, and IBM z/OS operating environments. Refer to *Base SAS Procedures Guide* and to the SAS documentation for your operating environment for more information about:

- SAS data sets
- SAS libraries
- naming conventions
- help with terminology used in this procedure description

### **PROC DBLOAD Naming Conventions**

When the DBLOAD procedure is used to load a SAS data set into a PC file, the SAS variable names cannot exceed eight characters. This restriction is for compatibility with SAS 6 naming conventions.

### **PROC DBLOAD Statement**

Not all DBLOAD procedure statements are available with all PC file formats. Additional statements might be used based on the PC file type.

---

## **Syntax: The DBLOAD Procedure**

```

PROC DBLOAD DATA=input data DBMS=data-base-identifier;
PATH='path and filename.PC file-extension' | 'filename' | fileref;
LOAD;

ACCDESC='<libref>.access-descriptor';
DELETE variable-identifier-1 <...variable-identifier-n>;
ERRLIMIT=error-limit;
FMTLIB=<libname.>member;
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 variable-identifier-1 <...variable-identifier-n>;
WHERE SAS where-expression;

```

Statement	Task
PROC DBLOAD Statement	Load data to PCs and creates PC files
ACCDESC Statement	Create an access description
DELETE Statement	Delete variables from the new data set
ERRLIMIT Statement	Limit loading obs after n number of errors
FMTLIB Statement	Specify a format library
LABEL Statement	Use SAS labels as default column names
LABEL Statement	Limit number of observations loaded into new file
LIST Statement	List information about SAS variables to be loaded into new file
LOAD Statement	Create a file and transfer data to it from an input SAS data set
PATH Statement	Indicate the path and name of the PC file that you create and load
QUIT Statement	Exit the procedure without further processing
RENAME Statement	Rename the PC File columns associated with the listed SAS variables
RESET Statement	Reset column names, data types, and enable setting of null values
WHERE Statement	Load a subset of observations into the PC file

---

## PROC DBLOAD Statement

Loads data to PCs and creates PC files.

---

### Syntax

```
PROC DBLOAD DATA=input data DBMS=pc-file-format;
```

### Required Arguments

#### DATA= *INPUT DATA*

specifies the input data. The input data can be retrieved from a SAS data set, an SQL view, a DATA step view, a SAS/ACCESS view descriptor, or and Microsoft Excel file

**Default:** the last SAS data set that was created.

**Restriction:** If the data set is permanent, specify its two-level name *libref.SAS dataset*.

#### DBMS= *PC-FILE-FORMAT*

specifies the PC file format that you want to access.

**Table 19.1** PC File Format Summary

File Type	Statement
DBF	DBMS = DBF
DIF	DBMS = DIF
EXCEL 97-2003	DBMS = EXCEL
WKn	DBMS = WK1
WKn	DBMS = WK3
WKn	DBMS = WK4
MDB	DBMS = MDB
XLS	DBMS = XLS

---

## ACCDESC Statement

Creates an access descriptor based on the PC file that you are creating and loading.

**Alias:** ACCESS  
AD

---

### Syntax

```
ACCDESC=<libref>.access-descriptor;
```

### Details

After the 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.

---

## DELETE Statement

Prevents variables from being loaded into the new PC file.

**Restriction:** An editing statement, such as DELETE, must be specified after the database-description statements when you create and load a file.

**Interaction:** RENAME, RESET

---

## Syntax

**DELETE** *variable-identifier-1*<...*variable-identifier-n*>;

## Details

The DELETE statement deletes the specified SAS variables from the PC file being created. The *variable-identifier* can be either the SAS variable name or the positional equivalent from a LIST statement. The positional equivalent is the number that represents the variable's place in the data set.

For example, if you want to delete the third variable, submit this statement: **DELETE 3**;

You can delete as many variables as you want in one DELETE statement. If you delete more than one variable, separate the identifiers with spaces, not commas.

If you delete a variable from the list of variables, the positional equivalents of the variables do not change. For example, if you delete the second variable, the third variable is still referenced by the number 3, not 2.

## Example

The DELETE statement deletes the third variable. The RENAME statement, includes the third variable, assigns the default type, and assigns EMPNAME as the name. The RENAME statement enables you to include variables that you have previously deleted.

```
DELETE 3; RENAME 3='empname';
```

---

## ERRLIMIT Statement

Stops loading observations after the specified number of errors has occurred while inserting rows into a file.

**Default:** 100

**Restrictions:** ERRLIMIT must be a nonnegative integer.

An editing statement, such as ERRLIMIT must be specified after the database-description statements when you create and load a file.

**Notes:** Applies to DBF, DIF, WK1, WK3, WK4, Excel 4, Excel 5, and Excel 95 file formats under Microsoft Windows operating environments

Specify ERRLIMIT = 0 to allow an unlimited number of errors to occur.

---

## Syntax

**ERRLIMIT**=*error-limit*;

---

## FMTLIB Statement

Specifies the name of a format library to search.

---

## Syntax

**FMTLIB**=<*libref*> *member*;

---

## LABEL Statement

Causes DBMS column names to default to SAS labels.

**Default:** SAS labels.

**Restrictions:** 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.

**Interaction:** RENAME, RESET

---

### Syntax

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.

---

## LIMIT Statement

Limits the number of observations that can be loaded into the new file.

**Default:** 5000

**Restrictions:** The *load-limit* must be a nonnegative integer.  
If you omit the LIMIT statement, a maximum of 5,000 observations are inserted.

**Note:** To load all observations from your input data set, specify LIMIT=0.

---

### Syntax

LIMIT=*load-limit*;

### Details

The LIMIT= statement places a limit on the number of observations that can be loaded into the new DBMS table. The *load-limit* argument must be a nonnegative integer. To load all observations from your input data set, specify LIMIT=0.

The maximum number for the limit statement varies with each PC file.

---

## LIST Statement

Lists information about SAS data sets.

**Default:** ALL

---

## Syntax

**LIST** ALL | COLUMNS | FIELDS | *variable-identifier*;

### Required Arguments

#### ALL

lists information about all variables in the input SAS data set, regardless of whether those variables are selected for the load.

#### COLUMNS

lists information only about the input SAS variables that are selected for loading. This argument does not apply to DBF files.

#### FIELDS

lists information only about the input SAS variables that are selected for the load.

#### *variable-identifier*

lists information only about the specified variable. The *variable-identifier* 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.

## Example

The LIST statement lists information for the column associated with the third SAS variable

```
LIST 3;
```

---

## LOAD Statement

Create a file and transfer data to it from the input data set after the DBLOAD procedure is submitted.

---

## Syntax

**LOAD**;

## Details

This statement causes the interface view engine to create a file and transfer data to it from the input data set, after the DBLOAD procedure is submitted. 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:

- Database-description statements: PATH and your PC file specific statements.
- 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.

- When creating and loading statement, LOAD must appear last before RUN in order to create and load the new table.

- The RUN statement is used to process the DBLOAD procedure.

---

## PATH Statement

Indicates the path and name of the PC file to create and load.

**Restriction:** 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.

**Requirement:** This statement is required.

**Note:** The length of the filename can vary with the operating environment.

**See:** SAS documentation for your operating environment for any restrictions.

---

## Syntax

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

### Required Arguments

#### *path and filename.pc-file-extension*

specifies the fully qualified path and filename. Enclose the entire path and filename in quotation marks, including the appropriate PC file extension. 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 a SAS name, or if it begins with a number, enclose the filename in quotation marks

#### *fileref*

specifies a fileref that references the path and name of the file.

---

## QUIT Statement

Exits the procedure without further processing.

**Alias:** EXIT

---

## Syntax

**QUIT**;

---

## RENAME Statement

Renames PC file columns that are associated with the listed SAS variables.

**Alias:** COLUMN

**Restriction:** An editing statement, such as RENAME, must be specified after the database-description statements when you create and load a PC file.

**Interaction:** DELETE, LABEL, RESET

**Note:** If you omit RENAME, column names default to the corresponding SAS variable names, unless you specify a LABEL statement.

## Syntax

**RENAME** *variable-identifier-1=column-name-1 <...variable-identifier-n=column-name-n>*;

## Details

The *column-name* 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 *variable-identifier* can be a SAS variable name or the positional equivalent from the LIST statement. The positional equivalent is the number that represents where to place the variable in the data set.

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.

## Example

The RENAME statement renames the column associated with the third SAS variable.

```
RENAME 3="employname";
```

## RESET Statement

Resets column names and data types, and the ability to accept null values to their default values.

**Restriction:** You must use the RESET statement after the LABEL statement for the LABEL statement to take effect.

**Interaction:** DELETE, LABEL, RENAME

**Note:** You can reset as many columns as you want in one RESET statement.

## Syntax

**RESET** ALL | *variable-identifier-1 <...variable-identifier-n>*;

## Details

If you specify ALL, all columns are reset to their default values. In addition, any deleted columns are restored with their default values. An editing statement, such as RESET, must be specified after the database-description statements when you create and load a PC file.

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.

## Example

The RESET statement resets the column associated with the third SAS variable

```
RESET 3;
```

---

## WHERE Statement

Loads a subset of observations into the PC file.

**Restriction:** An editing statement, such as WHERE, must be specified after the database-description statements when you create and load a PC File.

**See:** *SAS Statements: Reference*

---

## Syntax

```
WHERE SAS-where-expression;
```

## Details

The *SAS where-expression* must be a valid statement that uses SAS variable names (not column names) as defined in the input data set.

## Example

The WHERE statement loads only the observations where the SAS variable Country has the value Brazil.

```
WHERE country='Brazil';
```

## Chapter 20

# File-Specific Reference for the ACCESS and DBLOAD Procedures

---

<b>Overview: ACCESS Procedure and DBLOAD Procedure</b> .....	<b>260</b>
<b>ACCESS Procedure: XLS Files</b> .....	<b>260</b>
Access Descriptors .....	260
Arguments .....	260
Syntax .....	261
SAS Formats: XLS Specifics .....	262
SAS Formats: Customized for XLS Strings .....	265
<b>DBLOAD Procedure: XLS Specifics</b> .....	<b>266</b>
DBLOAD Procedure Statements and Options .....	266
DBLOAD Procedure Data Conversions for XLS Files .....	267
Setting Environment Variables for XLS Files .....	270
<b>ACCESS Procedure: WKn Specifics</b> .....	<b>271</b>
ACCESS Procedure Syntax for WKn Files .....	271
ACCESS Procedure Data Conversions for WKn Files .....	274
<b>DBLOAD Procedure: WKn Specifics</b> .....	<b>275</b>
DBLOAD Procedure Syntax for WKn Files .....	275
DBLOAD Procedure Data Conversions for WKn Files .....	276
Setting Environment Variables for WKn Files .....	277
<b>ACCESS Procedure: DBF Specifics</b> .....	<b>279</b>
ACCESS Procedure Syntax for DBF Files .....	279
ACCESS Procedure Data Conversions for DBF Files .....	280
<b>DBLOAD Procedure: DBF Specifics (Windows)</b> .....	<b>280</b>
DBLOAD Procedure Syntax for DBF Files .....	280
DBLOAD Procedure Data Conversions for DBF Files .....	282
Setting Environment Variables for DBF Files .....	282
<b>ACCESS Procedure: DIF Specifics</b> .....	<b>283</b>
ACCESS Procedure Syntax for DIF Files .....	283
ACCESS Procedure Data Conversions for DIF Files .....	285
Datetime Conversions in the ACCESS Procedure .....	285
<b>DBLOAD Procedure: DIF Specifics</b> .....	<b>286</b>
DBLOAD Procedure Syntax for DIF Files .....	286
Datetime Conversions in the DBLOAD Procedure .....	287
Setting Environment Variables for DIF File Data Types .....	288

---

## Overview: ACCESS Procedure and DBLOAD Procedure

The ACCESS procedure (Microsoft Windows operating environments) creates descriptor files that describe data in a PC file to SAS. This enables you to directly read, update, or extract PC files data into a SAS data set. The ACCESS procedure can be used with Microsoft Excel (4, 5, 95), Lotus 1-2-3 (WK1, WK3, WK4), DBF, and DIF file formats.

---

## ACCESS Procedure: XLS Files

The ACCESS procedure for PC Files creates descriptor files that describe XLS data. The ACCESS Descriptors section provides XLS-specific syntax for the ACCESS procedure and describes ACCESS procedure data conversions. See [Chapter 18, “The ACCESS Procedure for PC Files,”](#) on page 226 for additional information.

### Access Descriptors

To create an access descriptor, use the ACCESS procedure with the DBMS=XLS option. There are six database-description statements:

- GETNAMES
- PATH
- RANGE
- SCANTYPE
- SKIPROWS
- WORKSHEET

These database-description statements supply XLS-specific information to SAS. The statements must immediately follow the CREATE statement. In addition to the database-description statements, editing statements can follow the database-description statements. In addition to the database-description statements, editing statements can follow the database-description statements.

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

### Arguments

Use the Access procedure 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. 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 SAS data sets. The Access procedure supports most of the XLS number formats and automatically converts them to the corresponding SAS formats. XLS data strings that are longer than 200 characters are

truncated during conversion to SAS data sets. XLS files that create SAS data sets can contain up to 256 variables and 16,384 observations.

*w* is based on Excel column width. The Excel format string controls *.d*.

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. You can drop the hidden column when you are creating the access descriptor. 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 display.

You can change the default value from NO to YES by setting the SS\_MIXED environment variable.

See [“Setting Environment Variables for XLS Files” on page 270](#).

Set the SS\_MIXED environment variable to YES and numeric values in XLS files are converted to character strings if the corresponding SAS variable type is character.

## Syntax

```
PROC ACCESS DBMS= XLS | EXCEL;
  CREATE 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>;
  UPDATE libref.member-name.ACCESS | VIEW;
  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'>
  SKIPROWS= <number-of-rows-to-skip>
  SUBSET <selection-criteria>
  TYPE 'column-identifier-1'. C | N <'column-identifier-n'>,C | N
  UNIQUE=YES | NO | Y | N
  WORKSHEET=<'worksheet-name'> RUN;
```

## SAS Formats: XLS Specifics

**Table 20.1** SAS Formats for XLS File Data

XLS File Data		SAS Variable Format	
Data Type	XLS Format String	Type	Format
Char <sup>1</sup>	@ <sup>2</sup>	Char	\$w.
Numeric <sup>3</sup>	General	Num	BEST
Numeric	0	Num	w.d
Numeric	0.00	Num	w.d
Numeric	#,##0	Num	COMMAw.d
Numeric	#,##0.00	Num	COMMAw.d
Numeric	#,##0_);(#,##0)	Num	NEGPARENw.d
Numeric	#,##0_);[Red](#,##0)	Num	NEGPARENw.d
Numeric	#,##0.00_);(#,##0.00)	Num	NEGPARENw.d
Numeric	#,##0.00_);[Red](#,##0.00)	Num	NEGPARENw.d
Numeric	\$\$#,##0_);(\$#,##0)	Num	DOLLARw.d
Numeric	\$\$#,##0_);[Red](\$#,##0)	Num	DOLLARw.d
Numeric	(\$#,##0.00_);(\$#,##0.00)	Num	DOLLARw.d
Numeric	(\$#,##0.00_);[Red](\$#,##0.00)	Num	DOLLARw.d
Numeric	_((\$#,##0_);_(\$#(##0);_(\$*"-"_);_(@_)	Num	DOLLARw.d
Numeric	_((\$#,##0_);_(\$#(##0);_(\$*"-"_);_(@_)	Num	NEGPARENw.d

<sup>1</sup> Label data.

<sup>2</sup> The XLS character format for Excel Version 5.

<sup>3</sup> Number, formula, or missing data.

XLS File Data		SAS Variable Format	
Data Type	XLS Format String	Type	Format
Numeric	_\$*#,##0.00_);_ (\$*(#,##0.00);_ (\$*"-"??);_(@_)	Num	DOLLARw.d
Numeric	_(*#,##0.00_);_ (*(#,##0.00);_(*"-"??_ );_(@_)	Num	NEGPARENw.d
Numeric	0%	Num	PERCENTw.d
Numeric	0.00%	Num	PERCENTw.d
Numeric	0.00E+00	Num	Ew.d
Numeric	##0.0E+0	Num	Ew.d
Numeric	m/d/yy	Num	MMDDYYw.
Numeric	d-mmm-yy	Num	MMDDYYw.
Numeric	d-mmm	Num	DATEw.
Numeric	mmm-yy	Num	MONYYw.
Numeric	h:mm AM/PM	Num	TIMEw.
Numeric	h:mm:ss AM/PM	Num	TIMEw.
Numeric	h:mm	Num	TIMEw.
Numeric	hh:mm	Num	TIMEw.
Numeric	h:mm:ss	Num	TIMEw.
Numeric	hh:mm:ss	Num	TIMEw.
Numeric	m/d/yy h:mm	Num	DATETIMEw.
Numeric	ddmmmyy	Num	DATEw.
Numeric	ddmmmyyyy:hh:mm:ss	Num	DATETIMEw.
Numeric	dd	Num	DATEw.
Numeric	dd/mm/yy	Num	DDMMYYw.
Numeric	dddd	Num	DATEw.
Numeric	mm/dd/yy	Num	MMDDYYw.

XLS File Data		SAS Variable Format	
Data Type	XLS Format String	Type	Format
Numeric	<i>mm:ss</i>	Num	MMSS $w$ .
Numeric	<i>mm yy</i>	Num	MONYY $w$ .
Numeric	<i>mm yyyy</i>	Num	MONYY $w$ .
Numeric	<i>mm.yy</i>	Num	MONYY $w$ .
Numeric	<i>mm.yyyy</i>	Num	MONYY $w$ .
Numeric	<i>mm-yy</i>	Num	MONYY $w$ .
Numeric	<i>mm-yyyy</i>	Num	MONYY $w$ .
Numeric	<i>mmyy</i>	Num	MONYY $w$ .
Numeric	<i>mmyyyy</i>	Num	MONYY $w$ .
Numeric	<i>mm.yy</i>	Num	MONYY $w$ .
Numeric	<i>mm.yyyy</i>	Num	MONYY $w$ .
Numeric	<i>mm/yy</i>	Num	MONYY $w$ .
Numeric	<i>mm/yyyy</i>	Num	MONYY $w$ .
Numeric	<i>mmmm</i>	Num	MONYY $w$ .
Numeric	<i>m</i>	Num	MONYY $w$ .
Numeric	<i>mmmyy</i>	Num	MONYY $w$ .
Numeric	<i>mmmyyyy</i>	Num	MONYY $w$ .
Numeric	<i>dddd, mmmm dd, yyyy</i>	Num	MONYY $w$ .
Numeric	<i>dddd, dd mmmm yyyy</i>	Num	MONYY $w$ .
Numeric	<i>mmmm dd, yyyy</i>	Num	MONYY $w$ .
Numeric	<i>dd mmmm yyyy</i>	Num	MONYY $w$ .
Numeric	<i>yy</i>	Num	YYMMDD $w$ .
Numeric	<i>yyyy</i>	Num	YYMMDD $w$ .
Numeric	<i>yy mm</i>	Num	YYMMDD $w$ .
Numeric	<i>yyyy mm</i>	Num	YYMMDD $w$ .

XLS File Data		SAS Variable Format	
Data Type	XLS Format String	Type	Format
Numeric	<i>yy:mm</i>	Num	YYMMDDw.
Numeric	<i>yyyy:mm</i>	Num	YYMMDDw.
Numeric	<i>yy-mm</i>	Num	YYMMDDw.
Numeric	<i>yyyy-mm</i>	Num	YYMMDDw.
Numeric	<i>yymm</i>	Num	YYMMDDw.
Numeric	<i>yyyymm</i>	Num	YYMMDDw.
Numeric	<i>yy.mm</i>	Num	YYMMDDw.
Numeric	<i>yyyy.mm</i>	Num	YYMMDDw.
Numeric	<i>yy/mm</i>	Num	YYMMDDw.
Numeric	<i>yyyy/mm</i>	Num	YYMMDDw.
Numeric	<i>yy-mm-dd</i>	Num	YYMMDDw.
Numeric	<i>yymmm</i>	Num	YYMMDDw.
Numeric	<i>yyyymmm</i>	Num	YYMMDDw.

### SAS Formats: Customized for XLS Strings

**Table 20.2** SAS Variable Formats for Customized XLS Format Strings

XLS File Data		SAS Variable Format	
Data Type	XLS Format String	Type	Format
Numeric	"\$"	Num	DOLLARw.d
Numeric	"E"	Num	Ew.d
Numeric	"m, d and y"	Num	MMDDYYw.
Numeric	"m and h"	Num	TIMEw.d
Numeric	"m and s"	Num	TIMEw.d
Numeric	"m and y"	Num	MONYYw.
Numeric	"m"	Num	DATEw.

XLS File Data		SAS Variable Format	
Data Type	XLS Format String	Type	Format
Numeric	"d"	Num	DATEw.
Numeric	"y"	Num	DATEw.
Numeric	"0.0"	Num	w.d
Numeric	Fraction values (#/?/?)	Num	BESTw.d
Numeric	Percent values (0.0%)	Num	PERCENTw.d
Numeric	All others	Num	BESTw.d

---

## DBLOAD Procedure: XLS Specifics

### DBLOAD Procedure Statements and Options

[Chapter 19, “The DBLOAD Procedure,” on page 249](#) enables you to read data, format data, and set environment variables for XLS-specific data.

The QUIT statement is also available in the DBLOAD procedure. QUIT causes the procedure to terminate. QUIT is used most often in interactive line mode and batch mode to exit the procedure without exiting SAS.

The DBLOAD procedure chooses the default version of Excel depending on your operating environment. For Windows, DBLOAD uses Excel 5. Excel 5 files have the identical format to Excel 95 files.

The DBLOAD procedure does not support Excel 97 or later files. For information about accessing these files, see the [“Supported Data Sources and Environments” on page 7](#).

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

*option(s)* can be one or more options.

**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. Change the SAS format 12.1 to DOLLAR15.2. The format changes from a fixed numeric format, width 12, and one decimal, to a currency format, width of 15 and two decimals.

**PUTNAMES = YES|NO|Y|N**

writes column names to the first row of the XLS file. The column names default to SAS variables names unless you specify the LABEL statement. You can modify the column names using the RENAME statement.

The PUTNAMES statement is optional. Omit PUTNAMES and data is read beginning in the first row of the XLS file. No column names are written to the file.

You can change the default value to YES by setting the SS\_NAMES environment variable. For more information, see [“Setting Environment Variables for XLS Files”](#) on page 270.

VERSION= *Excel-product-number*

specifies the version number of the Excel product that you are using, such as Excel 5. The *Excel-product-number* argument can be one of the values in the following table:

**Table 20.3** Excel Versions

Value	Description
3	Microsoft Excel Version 3
4	Microsoft Excel Version 4
5	Microsoft Excel Version 5
7	Microsoft Excel 95 (also called Microsoft Excel Version 7)

## 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. You can temporarily assign other formats and column widths to SAS variables by using the FORMAT statement. The loaded XLS file columns have the formats that you want.

*Note:* The FORMAT statement in the DBLOAD procedure 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.

The following table shows SAS variable types and formats and the XLS data types, formats, and column widths to which you can assign them. XLS date and time values are numeric data.

**Table 20.4** 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

SAS Variable Format		XLS File Data	
Type	Format	XLS Format String	Data Type
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
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

SAS Variable Format		XLS File Data	
Type	Format	XLS Format String	Data Type
Num	MONYYw.	<i>mmmyy</i>	NUMBER
Num	NEGPARENw.d	<i>#,##0_);(#,##0)</i>	NUMBER
Num	NENGOW.	<i>m/d/yy</i>	NUMBER
Num	PERCENTw.d	<i>0%</i>	NUMBER
Num	QTRw.	<i>m/d/yy</i>	NUMBER
Num	QTRRW.	<i>m/d/yy</i>	NUMBER
Num	SSNW.	<i>000-00-0000</i>	NUMBER
Num	TIMEw.d	<i>h:mm:ss</i>	NUMBER
Num	TODw.	<i>h:mm:ss</i>	NUMBER
Num	W	<i>0</i>	NUMBER
Num	WEEKDATEw.	<i>dddd, mmmm dd, yyyy</i>	NUMBER
Num	WEEKDATXw.	<i>dddd, dd mmmm yyyy</i>	NUMBER
Num	WEEKDAYw.	<i>m/d/yy</i>	NUMBER
Num	WORDDATEw.	<i>mmmmdd, yyyy</i>	NUMBER
Num	WORDDATXw.	<i>dd mmmm yyyy</i>	NUMBER
Num	YEARw.	<i>yy or yyyy</i>	NUMBER
Num	YYMM	<i>yy mm</i>	NUMBER
Num	YYMMC	<i>yy:mm</i>	NUMBER
Num	YYMMD	<i>yy-mm</i>	NUMBER
Num	YYMMN	<i>yy mm</i>	NUMBER
Num	YYMMP	<i>yy.mm</i>	NUMBER
Num	YYMMS	<i>yy/mm</i>	NUMBER
Num	YYMDDw.	<i>yy-mm-dd</i>	NUMBER
Num	YYMONw.	<i>yy mmm</i>	NUMBER
Num	Zw.d	<i>0w.d</i>	NUMBER

SAS Variable Format		XLS File Data	
Type	Format	XLS Format String	Data Type
Num	FRACT $w$ .	# ?/?	NUMBER

Excel column widths are set to  $w$  and display in the column. If the data is larger than column width, it displays as pound signs (###). In that case, you can view it by adjusting the column width.

## Setting Environment Variables for XLS Files

### SS\_MIXED SS\_NAMES SS\_SCAN

You can change the default behavior of the ACCESS procedure and the DBLOAD procedure 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 defines how the interface works. The configuration file omits these three environment variables. 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 the Access procedure.

#### SS\_NAMES= YES|NO

YES allows the Access procedure to generate SAS variable names from column names. The first row of the worksheet or the specified range of the worksheet is used to generate SAS variable names. Data is then read from the second row.

YES in the DBLOAD procedure writes column names using SAS variable names. YES also writes SAS variable labels to the first row of the XLS file. SAS reads the data from the data set, and writes it to the XLS file beginning with the second row.

NO in the Access procedure 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 the DBLOAD procedure 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= option in the Access procedure and the PUTNAMES= option in the DBLOAD procedure.

#### 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 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. If SKIPROWS= is specified, the first row is after skipping the number of rows specified. NO is the default.

*Number-of-rows* scans only the type and format of the specified number of rows. Setting the number of rows is more efficient because data is read from only the specified number of rows rather than the entire file.

Setting the SS\_SCAN environment variable changes the default value of the SCANTYPE statement in the Access procedure.

---

## ACCESS Procedure: WK $n$ Specifics

See the ACCESS procedure for PC files for general information about this feature. This is the WK $n$ -specific syntax for the ACCESS procedure and the description of the ACCESS procedure data conversions.

### ACCESS Procedure Syntax for WK $n$ Files

To create an access descriptor, use the DBMS= WK  $n$  option and these database-description statements:

- GETNAMES
- PATH
- RANGE
- SCANTYPE
- SKIPROWS
- 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 that you can use editing statements when you create an access descriptor. These editing statements must follow the database-description statements.

Database-description statements are required only 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;
ASSIGN YES | NO | Y | N
CREATE <libref.member-name.>ACCESS | VIEW;
DROP <'column-identifier-1'>...<'column-identifier-n'>
FORMAT <'column-identifier-1'> <'SAS format-name-1'>...<'column-identifier-n'>
<'SAS format-name-n'>
GETNAMES YES | NO | Y | N
LIST ALL|VIEW |<'column-identifier'>
PATH='path-and-filename.WK1 | .WK3 | .WK4'| <filename>| <fileref>
RANGE <'range-name'>|<'range-address'>
SCANTYPE YES | NO | Y | N |
SKIPROWS <number-of-rows-to-skip>
UPDATE libref.member-name.ACCESS | VIEW
WORKSHEET <worksheet-letter>| worksheet-name
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 the Access procedure. It causes the procedure to terminate. QUIT is used most often in interactive line and batch 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. Omit the statement and the default value GETNAMES=NO is used. The SAS/ACCESS interface generates the SAS variable names VAR0, VAR1, VAR2, and so on. Specify GETNAMES=YES and 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. For more information, see [“Setting Environment Variables for WK \$n\$  Files” on page 277](#).

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. 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. The range address C9..F12 indicates a cell range that begins at Cell C9, ends at Cell F12, inclusive.

The RANGE statement is optional. Omit RANGE and 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 a WK $n$  worksheet to generate the SAS format. SAS variable formats are generated from the Lotus 1-2-3 formats found in the first row of the worksheet or in the specified range of the worksheet.

The SCANTYPE statement is optional, and its default value is NO. Specify YES and the ACCESS procedure scans the Lotus 1-2-3 formats of all the rows in each column of the range. The procedure uses the most common format to generate the default SAS format for each column. Specify a number of rows for the Access procedure to scan only the specified number of rows. The procedure returns the most common format.

Specify the SKIPROWS statement and the ACCESS procedure skips the specified rows. It starts scanning the Lotus 1-2-3 format from the next row. Specify SKIPROWS=3 and the Access procedure begins scanning the formats on the fourth row.

You can change the default value to YES by setting the SS\_SCAN environment variable. For more information, see [“Setting Environment Variables for WK \$n\$  Files” on page 277](#).

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 are used to generate the SAS variable types and formats. You can use the SCANTYPE= statement to scan the formats of specified rows. Then use the most common type and format to generate the 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. The CREATE= option precedes the SKIPROWS= option. Editing statements follow the SKIPROWS= option.

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. Specifying WORKSHEET=B identifies worksheet B from a group of worksheets.

The WORKSHEET statement is an optional 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

Use the Access procedure to define descriptors that identify spreadsheet data and the conversions necessary to use that data in SAS programs.

The Lotus 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. The ACCESS procedure supports most of the WK $n$  number formats and automatically converts them to corresponding SAS formats.

Any WK $n$  data strings longer than 200 characters are truncated while being converted into SAS data sets. Any SAS data set created from WK $n$  files can contain up to 256 variables and 8,192 observations.

The following table shows the default SAS System variable formats that the ACCESS procedure assigns to each type of WK $n$  file. Numeric data include date and time values.

**Table 20.5** Default SAS System Variable Formats for WK $n$  File Data

WK $n$ File Data				SAS Variable Format	
Data Type	Data Format	Column Width	Decimal Number	Type	Format
Char**	*	$w$		Char	$\$w.$
Numeric***	Default	$w$		Num	BEST $w.$
Numeric	AUTOMATIC	$w$	$d$	Num	BEST $w.$
Numeric	FIXED	$w$	$d$	Num	$w.d$
Numeric	SCIENTIFIC	$w$	$d$	Num	$Ew.d$
Numeric	CURRENCY	$w$	$d$	Num	DOLLAR $w.d$
Numeric	PERCENT	$w$	$d$	Num	PERCENT $w.d$
Numeric	COMMA	$w$	$d$	Num	COMMA $w.d$
Numeric	GENERAL	$w$		Num	BEST $w.$
Numeric	DD-MON-YY	$w.$		Num	DATE7.
Numeric	DD-MON	$w$		Num	DATE7.
Numeric	MON-YY	$w$		Num	MONYY5.
Numeric	MM-DD-YY	$w$		Num	MMDDYY8.

WK $n$ File Data			SAS Variable Format		
Data Type	Data Format	Column Width	Decimal Number	Type	Format
Numeric	MM-DD	$w$		Num	MMDDYY8.
Numeric	HH-MM-SS	$w$		Num	TIME8.
Numeric	HH-MM-SS	$w$		Num	TIME5.
Numeric	HH-MM-SS AM/PM	$w$		Num	TIME12.
Numeric	HH-MM AM/PM	$w$		Num	TIME9.

\* Any valid Lotus 1-2-3 data format.

\*\* Label or formula string data.

\*\*\* Number or formula data.

If WK $n$  file fall outside of the valid SAS data ranges, when you try to access the data, an error message is written to the SAS log.

The SAS/ACCESS interface does not fully support the Lotus 1–2–3 hidden and text formats. WK $n$  data in hidden format are displayed in SAS data sets. You can drop the hidden column when creating the access descriptor.

To display the formula in the text format, add a label prefix character. This indicates that the formula entry is a label. Otherwise, the results of the formula are displayed.

---

## DBLOAD Procedure: WK $n$ Specifics

See [Chapter 19, “The DBLOAD Procedure,”](#) on page 249 for general information about this feature. This is WK $n$ -specific syntax for the DBLOAD procedure and description of DBLOAD procedure data conversions. See [“DBLOAD Procedure Data Conversions for WK \$n\$  Files”](#) on page 276.

### 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>
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
PUTNAMES=YES | NO | Y | N
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>
RUN;

```

The QUIT statement is also available in the DBLOAD procedure. It causes the procedure to terminate. QUIT is used most often in interactive line mode and batch mode to exit the procedure without exiting SAS.

WK<sub>n</sub>-specific statements are as follows:

```
PUTNAMES =YES|NO|Y|N
```

writes column names to the first row of the new WK<sub>n</sub> file. The column names are the SAS variable names. You can use the LABEL statement to assign SAS variable labels. You can modify the column names using the RENAME statement.

The PUTNAMES statement is optional. Omit PUTNAMES= and data is read from the data set and written to the WK<sub>n</sub> file. Data is written beginning in the first row of the WK<sub>n</sub> file. No column names 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 the FORMAT statement to change the format, column width, or the number of decimal digits for columns being loaded into the PC file. If you change the SAS variable format 12.1 to DOLLAR15.2, the column format of the loaded data changes. The fixed numeric format with a column width of 12 and one decimal digit changes to a currency format with a column width of 15 and two decimal digits.

### **DBLOAD Procedure Data Conversions for WK<sub>n</sub> Files**

SAS data is read into Lotus 1-2-3 data when a table is loaded. In this conversion, SAS character data type is converted into the Lotus 1-2-3 label type. SAS numeric type is converted into the Lotus 1-2-3 number type.

The SAS/ACCESS interface converts SAS formats to the same or associated Lotus 1-2-3 formats and column widths. You can temporarily assign other formats and column widths to SAS variables with the FORMAT statement.

*Note:* The FORMAT statement in the DBLOAD procedure changes only the format of SAS variables while you are creating and loading the WK<sub>n</sub> files. When the procedure finishes, the formats of the SAS variables return to their original settings.

**Table 20.6** Converting SAS Variable Formats to WKn File Data

SAS Variable Format		WKn File Data			
Type	Data Format	Data Type	Column Format	Column Width	Number
Char	\$w.	LABEL	DEFAULT	w	
Char	\$CHARw.	LABEL	DEFAULT	w	
Num	w.d	NUMBER	FIXED	w	d
Num	Fw.d	NUMBER	FIXED	w	d
Num	Ew.d	NUMBER	SCIENTIFIC	w	d
Num	DOLLARw.d	NUMBER	CURRENCY	w	d
Num	PERCENTw.d	NUMBER	PERCENT	w	d
Num	COMMAw.d	NUMBER	COMMA	w	d
Num	BESTw.	NUMBER	DEFAULT	w	
Num	BESTw.	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	

### Setting Environment Variables for WKn 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, which means their default values are NO.

#### SS\_MISS NULLS

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 are 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 the Access procedure.

#### SS\_NAMES YES | NO

YES in the Access procedure generates SAS variable names from column names in the first row of the worksheet or specified range of the worksheet. Data is then read starting from the second row.

YES in the DBLOAD procedure writes column names using SAS variable names or SAS variable labels to the first row of the new WK $n$  file. Data is then read from the data set and written to the WK $n$  file beginning with the second row.

NO in the Access procedure generates the SAS variable names VAR0, VAR1, VAR2, and so on. It reads data from the first row of the worksheet or a specified range. NO is the default.

NO in the DBLOAD procedure reads the data from the data set and writes it to the WK $n$  file beginning with the first row.

Setting the SS\_NAMES environment variable changes the default value of the GETNAMES= option in the Access procedure, and the PUTNAMES= option in the DBLOAD procedure.

#### SS\_SCAN= YES | NO | *number-of-rows*

YES scans the data type and format of rows in a worksheet or specified range. If SKIPROWS= is specified, the first row is after skipping the number of rows specified. SS\_SCAN finds the most common Lotus 1-2-3 data type and format to generate the SAS data type and format. If a number of rows is specified, the Access procedure scans only the data type and format from the specified rows.

NO uses the type and format of the first row in a worksheet or specified range. If SKIPROWS= is specified, the first row is after skipping the number of rows specified. NO is the default.

*Number-of-rows* scans only the type and format of the specified number of rows. Setting the number of rows is more efficient because data is read from only the specified number of rows rather than the entire file.

Setting the SS\_SCAN environment variable changes the default value of the SCANTYPE= statement option in the Access procedure.

## ACCESS Procedure: DBF Specifics

See [Chapter 18, “The ACCESS Procedure for PC Files,”](#) on page 226 for 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, use the DBMS= DBFMEMO option and the database-description statement PATH= option. This PATH= option 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 required only 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 <option(s)>
CREATE <libref.name.>ACCESS|VIEW
PATH=<path-and-filename.DBF>|<filename>|<fileref>
UPDATE libref.name.ACCESS|VIEW
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 the Access procedure. It causes the procedure to terminate. QUIT is used most often in interactive line mode and batch mode to exit the procedure without exiting SAS.

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

```
OPTIONS LINESIZE=80;
LIBNAME adlib 'SAS data-library';
LIBNAME vlib '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.;
        LOST 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 20.7** Default SAS Variable Formats for DBF File Data Types

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

\* This data type applies to dBASE V and later. Check with other software products' documentation to determine whether this data type applies.

---

## DBLOAD Procedure: DBF Specifics (Windows)

See [Chapter 19, “The DBLOAD Procedure,” on page 249](#) for general information about this feature. This section provides DBF-specific syntax for the DBLOAD procedure and describes DBLOAD procedure data conversions “[DBLOAD Procedure Data Conversions for DBF Files](#)” on page 282.

### DBLOAD Procedure Syntax for DBF Files

To create and load a DBF table, SAS/ACCESS Interface to PC Files uses the following statements:

```

PROC DBLOAD DATA=<libref.SAS data-set>;
DBMS=DBFMEMO
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>;
LISTALL|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. It causes the procedure to terminate. QUIT is used most often in interactive line mode and batch mode to exit the procedure without exiting SAS.

The following list provides detailed information about the DBF-specific statements:

**VERSION**= *dBASE-version-number*

specifies the version of the dBASE product that you are using, such as dBASE IV. The *dBASE-version-number* argument can be one of the following values: II, III, IIIP, IV, V, 2, 3, 3P, 4, and 5. *v* is the default.

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.

This example defines the data types for several database fields. You can specify the length of the data type.

```

PROC DBLOAD DBMS=DBFMEMO DATA=employee;
  PATH='c:\sasdemo\employee.dbf';
  RENAME firstname = fname;
  TYPE  empid      = 'numeric(6)';
        hiredate   = 'date';
        salary     = 'numeric(10,2)';
        jobcode    = 'numeric(5)';
  RUN;

```

This example creates a DBF table, Exchange.Dbf, from the data file DLib.RateOfex. An access descriptor DbFliba.Exchange is created based on the new table. You must be granted the appropriate privileges to create DBF tables.

```

LIBNAME dbfliba 'SAS data-library';
LIBNAME dlib 'SAS data-library';

PROC DBLOAD DBMS=DBF DATA=dlib.rateofex;
  PATH='c:\dbfiles\sasdemo\exchange.dbf';
  ACCDESC=dbfliba.exchange;

```

```

RENAME fgnindol=fgnindolar 4=dolrsinfgn;
TYPE country='char(25)';
LOAD;
RUN;

```

## DBLOAD Procedure Data Conversions for DBF Files

**Table 20.8** DBF File Data Types Assign Corresponding SAS Variable Formats

SAS Variable Formats	DBF File Data Types
\$w.	CHAR( <i>n</i> )
w.	NUMERIC
w.d.	NUMERIC
datetimew.d	DATE
datew.	DATE
ew.	FLOAT
binaryw.	NUMERIC

## Setting Environment Variables for DBF Files

Missing numeric values are replaced with nines by default. The DBFMISCH environment variable is used to change the default. Specify the character that the interface to DBF files represents as missing data in numeric fields. If you write a SAS file with a missing numeric variable to a DBF file, the corresponding field in the DBF file is 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.

Set the DBFMISCH environment variable in the SAS configuration file as follows:

```
-SET DBFMISCH value
```

Valid values are as follows:

*any single character*

to fill missing numeric values with any character (zero in this code), **-SET DBFMISCH 0.**

**NULLS**

to replace missing numeric values with binary zeros, **-SET DBFMISCH NULLS.**

**BLANKS**

replace missing numeric values with blanks, **-SET DBFMISCH BLANKS.**

---

## ACCESS Procedure: DIF Specifics

See “ACCESS Procedure Data Conversions for DIF Files” on page 285.

### ACCESS Procedure Syntax for DIF Files

To create an access descriptor, 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 required only 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.

```
PROC ACCESS <option(s)>;
  CREATE <libref.>member-name.ACCESS | VIEW
```

The SAS/ACCESS interface to DIF uses the following procedure statements:

- ASSIGN= | AN YES | NO
- DIFLABEL
- DROP 'column-identifier-1'... 'column-identifier-n'
- FORMAT 'column-identifier-1' 'SAS format-name-1'... 'SAS variable-name-n' 'column-identifier-n'
- LIST= ALL | VIEW | 'column-identifier'
- PATH='path-and-filename'|.DIF|filename fileref
- RENAME 'column-identifier-1' SAS variable-name-1... 'column-identifier-n' SAS variable-name-n
- RESET= ALL | 'column-identifier-1'... 'column-identifier-1'
- SELECT=ALL|'column-identifier-1'... 'column-identifier-n'
- SKIPROWS='number-of-rows-to-skip'
- SUBSET
- TYPE='column-identifier-1'= C | N ... 'column-identifier-n'= C | N
- UPDATE='path-and-filename'| 'filref'
- UNIQUE=YES | NO
- RUN;

The QUIT statement is also available in the Access procedure. It causes the procedure to terminate. QUIT is used most often in interactive line mode and batch mode 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. Omit this statement and variable names are generated based on the columns' placement in the first row. SAS labels each column as **COL0**, **COL1**, **COL2**, and so on. These labels are the names of SAS variables in the access descriptor.

Specify **DIFLABEL** and the **ACCESS** procedure reads column labels from the first row of the **DIF** file. The labels are also used as the SAS variable names in the access descriptor. You provide the **DIF** file column labels; they are not the letters ( **A**, **B**, and so on) that identify the columns in a worksheet. Specify **DIFLABEL** , and the **SKIPROWS=** statement automatically changes to **1**.

Always specify the **DIFLABEL** statement after the **PATH=** statement and before any editing statements. When you update a descriptor, the **DIFLABEL** statement cannot be changed.

Create an Access Descriptor and a View Descriptor Based on **DIF** file data.

```

OPTIONS LINESIZE=80;
LIBNAME difliba '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.;
    LOST 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 skip when reading data. The default value for **SKIPROWS** is **0**. The skipped rows often contain information such as column labels or names, or underscores rather than input data.

When you specify the **DIFLABEL** statement, the default value of **SKIPROWS** automatically changes to **1**.

When you are creating a description, **SKIPROWS=** should always follow the **PATH=** statement and precede any editing statements. 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. This is true even if the data in the next row is numeric.

Any data value in a column that does not match the type is treated as a missing value.

If you set the **DIFNUMS** environment variable to **YES** in your SAS configuration file, numeric values in a character column are converted to character. The values are not treated as missing values. To set the **DIFNUMS** value, add the following line to your SAS configuration file: **-SET DIFNUMS YES**

The default value for the **DIFNUMS** environment variable is **NO**.

You can change the column type from the default type that the Access procedure defines when you create an access descriptor.

*Note:* Refer to the SAS documentation for your operating environment for more information about environment variables.

### 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.

**Table 20.9** 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 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.

If a SAS datetime format is stored in a column's descriptor, the Lotus value for January 1, 1960 (21,916) is converted to the SAS value for January 1, 1960 (0). Otherwise, the Lotus value of 21,916 is treated as a SAS numeric value of 21,916.

**Table 20.10** 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

See [Chapter 19, “The DBLOAD Procedure,”](#) on page 249 for general information about this feature. This section provides DIF-specific syntax for the DBLOAD procedure and describes DBLOAD procedure datetime conversions. See [“Datetime Conversions in the DBLOAD Procedure”](#) on page 287.

### DBLOAD Procedure Syntax for DIF Files

To create and load a DIF table, 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 the DBLOAD procedure. It causes the procedure to terminate. QUIT is used most often in interactive line mode and batch mode 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. Specify the LABEL statement to use SAS labels. You can modify the column labels using the RENAME statement.

*Note:* If this statement is omitted, data is read from the data set and written to the DIF file beginning in the first row. 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 SAS format to a SAS variable in the input SAS data set. This format temporarily overrides any other format for the variable. The assignment lasts for the duration of the procedure. You can assign formats to as many variables as you want in one FORMAT statement.

Use the FORMAT statement to change the format, column width, or the number of decimal digits for columns being loaded into the PC file. If you change the SAS variable format 12.1 to DOLLAR15.2, the column format of the loaded data

changes. The fixed numeric format with a column width of 12 and one decimal digit changes to a currency format with a column width of 15 and two decimal digits.

This example creates a 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 dlib 'SAS data-library';
LIBNAME adlib '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 value into the equivalent Lotus datetime value. The conversion is written to the DIF file when it is created. 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. It is recommended that you 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.

*Note:* SAS dates are based on January 1, 1960. 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.

Use the DBLOAD FORMAT statement to maintain a SAS variable format in the input data set. This changes the format only while the DBLOAD procedure is in progress. Assigning a temporary format to a SAS variable does not affect how SAS stores the variable.

If the SAS format for the BirthDat variable in the MyData.SasEmps data set is the default 15.2 format, you can specify a FORMAT statement to change the format to DATE7. Use the FORMAT statement while you are creating and loading the DIF file. When you load the DIF file into a Lotus 1-2-3 worksheet, specify an equivalent Lotus date format. Specify the FORMAT statement when you invoke the DBLOAD procedure using any of the methods of processing. When the DBLOAD procedure has completed, the SAS format for the BirthDat variable reverts to its original 15.2 format.

*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 DATETIMEw.d format into a DIF file, Lotus stores the number with both the integer and decimal. When you load the DIF file into a Lotus 1-2-3 worksheet that you can specify a date format for the column. DATE formats only use the integer portion of the data. Alternatively, you can also specify a TIME format that only uses the decimal portion of the data. You cannot specify both at the same time.

### Setting Environment Variables for DIF File Data Types

By default, any data value in a column that does not match the type is treated as a missing value. 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. They 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 that SAS/ACCESS determines when you create an access descriptor.

## Part 6

---

# Appendixes

<i>Appendix 1</i>	
<b>LIBNAME Statement for the JMP Engine</b> .....	291
<i>Appendix 2</i>	
<b>The DBF and DIF Procedures</b> .....	293



## Appendix 1

# LIBNAME Statement for the JMP Engine

---

Dictionary .....	291
LIBNAME Statement for the JMP Engine .....	291

---

## Dictionary

---

### LIBNAME Statement for the JMP Engine

Associates a libref with a JMP data table and enables you to read and write JMP data tables.

<b>Valid in:</b>	Anywhere
<b>Category:</b>	Data Access
<b>See:</b>	Base SAS LIBNAME Statement

---

### Syntax

```
LIBNAME libref JMP 'path' <FMTLIB=libref,format-catalog>;
```

### Arguments

#### *libref*

is a character constant, variable, or expression that specifies the libref that is assigned to a SAS library.

**Range:** 1 to 8 bytes

#### *path*

is the physical name for the SAS library. The physical name is the name that is recognized by the operating environment. Enclose the physical name in single or double quotation marks.

#### FMTLIB=*libref,format-catalog*

specifies where the formats are stored when a JMP data table is read and where the formats come from when a JMP data table is created.

**Requirement:** The library that is specified in the FMTLIB argument must be a SAS data set LIBNAME statement.

**Example:**

```
libname inv jmp "." fmtlib=seform.formats;
libname seform '.';
data work.mine;
set inv.suri2011;
run;
```

## Details

A JMP file is a file format that the JMP software program creates. JMP is an interactive statistics package that is available for Microsoft Windows and Macintosh. For more information, see the JMP documentation that is packaged with your system.

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.

Base SAS supports access to JMP files. You can access JMP files by either of these two methods:

- the IMPORT and EXPORT procedures and the Import and Export Wizard without a license for SAS/ACCESS Interface to PC Files

For more information, see *SAS/ACCESS Interface to PC Files: Reference*.

- the LIBNAME statement for the JMP engine

## Examples

### **Example 1: Using the LIBNAME Statement to Read a JMP Data Table**

This example reads and prints five observations from the **bank** JMP data table.

```
libname b jmp 'c:/temp/national';
proc contents data=b.bank(drop=edlevel id age);
run;
proc print data=b.bank(obs=5 drop=edlevel id age);
run;
```

### **Example 2: Reading and Sorting a JMP Data Table**

This example reads a JMP data table, sorts it, and stores it in a SAS data set. The formats stored on the JMP data set are put in **a.formats**.

```
libname a 'c:/temp/field';
libname b jmp '.' fmtlib=a.formats;

proc sort data=b.cars out=a.sorted;
by category_ic;
run;
```

## Appendix 2

# The DBF and DIF Procedures

---

<b>Overview: DBF and DIF Procedures</b> .....	<b>293</b>
<b>Dictionary</b> .....	<b>293</b>
The DBF Procedure .....	293
PROC DIF Statement Options .....	297

---

## Overview: DBF and DIF Procedures

The DBF and DIF procedures provide an alternate way to access dBase files. You can use these procedures to convert PC file types to SAS data sets, or vice versa.

Under UNIX and Microsoft Windows operating environments, you can use the DBF and DIF procedures to convert a DBF or DIF file to a SAS data set. You can convert a SAS data set to a DBF or DIF file.

See [“Methods for Accessing PC Files Data”](#) on page 3 for additional ways to access data in PC file formats under UNIX, and Microsoft Windows operating environments.

---

## Dictionary

---

### The DBF Procedure

Convert a data interchange format (DBF) files to SAS data sets.

---

#### Syntax

**PROC DBF** *option(s)*

#### Required Argument

**DB2 | DB3 | DB4 | DB5** *=fileref|filename*

specifies the version of the dBase file and 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. Specify the version with the DB *n* option, where *n* is the version number. The values are 2, 3, 4, or 5.

To specify a fileref, the FILENAME statement must specify the filename with a .dbf extension. This example assigns the fileref *myref* to the file `'/my_dir/myfile.dbf'`:

```
filename myref '/my_dir/myfile.dbf';
```

If you specify a filename instead of a fileref, specify the filename without the .dbf extension. The file must be in the current directory, in uppercase.

The following 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;
```

**Restriction:** You *cannot* specify the filename extension .dbf or a full pathname.  
PROC DBF DB5='/my/unix\_directory/emp.dbf'.

**Requirements:**

- You must specify a FILENAME statement.
- The DB *n* = option is required.

## Optional Arguments

### DATA= <libref>.member

specifies the name of the SAS data set you are reading to create a DBF file. If you omit the DATA= option, SAS creates an output SAS data set from the DBF file. The output file is written to the temporary WORK directory. The naming convention is Data1, Data2, DataN. The temporary WORK file is available only during your current SAS session.

### OUT=<libref.>member

specifies the name of the SAS data set you are creating from a DBF file. If you omit the OUT= option, SAS creates an output SAS data set from the DBF file. 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, the temporary data set is named Data1, ..., Data *n*]. Under Microsoft Windows, it is called \_DATA\_). The WORK data set remains available only during your current SAS session.

## Details

### Overview

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

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

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

*Note:* Any DBF file that you plan to import 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 the DBF file contains inconsistent data, such as a row of underscores, hyphens, or blanks, delete these rows before converting the file. It is recommended that you make a backup copy of your DBF table before you make these modifications.

### **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 with data 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.

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.

The DBF procedure 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 supports DBF files that are dBASE (II, III, III PLUS, IV, and 5.0) versions and releases. The DBF procedure supports most DBF files that other software products create.

*Note:* Future versions of dBASE files might not be compatible with the current version of the DBF procedure.

### **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 converted from a DBF file. SAS decimal values must be stored in a decimal format to be converted to a DBF decimal value. Associate the SAS numeric variable with an appropriate decimal format. The corresponding DBF field does not have any value to the right of the decimal point. You can associate a format with SAS variables when you create the data set or with 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 convert 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 convert a SAS data set to a dBASE II file, SAS date variables become dBASE II character fields in the form *YYYYMMDD*.

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 convert 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 convert 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 Select ( Emp.dbf file) and use PROC DBF to convert that file into a SAS data set.

You could also do the reverse: use PROC DBF to convert a SAS data set into a DBF file and then load the DBF file into the Excel spreadsheet.

**Examples****Example 1: Converting a SAS Data Set to a dBASE 5 File**

**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

```
LIBNAME mylib '/my/unix_directory';
FILENAME employee '/sasdemo/employee.dbf';
PROC DBF DB5=employee DATA=mylib.employee;
RUN;
```

**Example 2: Converting a dBASE II File to a SAS Data Set on UNIX**

The dBASE II file named Employee.dbf is converted to a SAS data set. No FILENAME statement is specified so the last level of the filename is .dbf. 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 3: Creating a DBF File from a SAS Data Set in a Microsoft Windows Environment**

In this example, the mylib.employee SAS data set is converted to a dBASE 5 file. The FILENAME statement specifies the name of the DBF file. You must specify the FILENAME statement before the PROC DBF statement.

```
LIBNAME mylib 'c:\my\directory';
FILENAME employee 'c:\sasdemo\employee.dbf';
PROC DBF DB5=employee DATA=mylib.employee;
RUN;
```

**Example 4: Creating a DBF File from a SAS Data Set in a z/OS Environment**

This example is the same as the previous example - for the z/OS environment.

```
LIBNAME mylib 'sasdemo.employee.data';
FILENAME dbfout 'sasdemo.newemp.dbf' RECFM=n;
PROC DBF DB5=dbfout DATA=mylib.employee;
RUN;
```

---

## PROC DIF Statement Options

Convert a DIF file to SAS data set or a SAS data set to a DIF file.

**Restriction:** The DIF procedure is available only under UNIX and Microsoft Windows operating environments.

---

### Syntax

**PROC DIF** *option(s)*;

### Summary of Optional Arguments

#### PROC Statement

*DIF = fileref|filename*

#### Statement Option

*DATA= libref.member*

**LABELS**

**OUT=**

*PREFIX= name*

*SKIP= n*

### Optional Arguments

#### **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.

```
filename myref '/my_dir/myfile.dif'
```

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

```
PROC DIF dif=emp data=mylib.employee;
      RUN;
```

You cannot specify emp.dif or a full pat name

```
PROC DIF dif='/my/unix_directory/emp.dif';
      RUN;
```

#### **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=**

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, ..., DataN). Under Microsoft

Windows, it is called `_DATA_`. If `OUT=` is omitted or if you do not specify a two-level name, the data set remains available during your current SAS session. The data set is not permanently saved.

### LABELS

specifies that PROC DIF writes the names of the SAS variables as the first row of the DIF file. It also writes 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.

**Restriction:** 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`.

**Default:** If you omit the `PREFIX=` option, PROC DIF assigns the names `Col1`, `Col2`, ..., `ColN`.

### 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, 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 begins in row 3. You should specify `SKIP= 2` so that PROC DIF ignores the non-data portion of your DIF file. You could also delete the first two rows of your DIF file before using PROC DIF.

## Details

### Overview

The DIF procedure converts data interchange format (DIF) files to SAS data sets. The data sets are compatible with the current release of SAS software, or SAS data sets convert 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, it is still supported by many software products.

*Note:* Any DIF file that you plan to import 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 the DIF file contains inconsistent data, such as a row of underscores, hyphens, or blanks, delete these rows before converting the file. It is recommended that you make a backup copy of your DIF table before you make these modifications.

When you are converting a DIF file, each row of the file becomes an observation in the SAS data set. Conversely, when you are converting a SAS data set, 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 represented as integers or scientific notation in a DIF file, become SAS numeric variables when 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 transfer SAS data sets to another software product using DIF files, 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 from a software product to a SAS data set, reverse the process. Convert the data to a DIF file, run the DIF procedure to transfer 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 with a value indicator other than V as missing values. The DIF procedure follows this convention. When a DIF file is imported 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 with missing numeric values 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***

A DIF file named Employee.dif is converted to a SAS data set. Because a FILENAME statement is not specified, the last level of the filename is assumed to be .dif. The file is assumed to be in your current directory, 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***

A SAS data set named EMP is converted to a DIF file. A FILENAME statement is used to specify a fileref that names the DIF file. Specify the FILENAME statement before the PROC DIF statement.

```
FILENAME employee 'c:\sasdemo\employee.dif';
PROC DIF DIF=employee DATA=save.employee;
RUN;
```

**Example 3: Convert a SAS Data Set to a DIF File in the UNIX Environment**

This is basically the same codes as the previous example, coded for the UNIX environment. A SAS data set named EMP is converted to a DIF file. A FILENAME statement is used to specify a fileref that names the DIF file. Specify the FILENAME statement before the PROC DIF statement.

```
FILENAME emp '/sasdemo/emp_UNX.dif';  
PROC DIF DIF=emp DATA=save.emp;  
RUN;
```

# Glossary

---

**access descriptor**

a SAS/ACCESS file that describes data that is managed by SAS, by a database management system, or by a PC-based software application such as Microsoft Excel, Lotus 1-2-3, or dBASE. After creating an access descriptor, you can use it as the basis for creating one or more view descriptors.

**client**

an application that requests either resources or services from a server, possibly over a network.

**column**

a vertical component of a table. Each column has a unique name, contains data of a specific type, and has particular attributes. A column is analogous to a variable in SAS terminology.

**commit**

the process that ends a transaction and that makes permanent any changes to the database that the user made during the transaction.

**Data Interchange Format file**

an ASCII text file with a file header section and a data section that is used to exchange data between incompatible systems. Software vendors' files must have a tabular format in order to be translated using DIF. Using the Lotus 1-2-3 Translate Utility, a 1-2-3 worksheet can be translated into a DIF file; DIF files, not 1-2-3 worksheets, are specified in the ACCESS and DBLOAD procedures. Short form: DIF file.

**data set**

See SAS data set.

**DATA step view**

a type of SAS data set that consists of a stored DATA step program. A DATA step view contains a definition of data that is stored elsewhere; the view does not contain the physical data. The view's input data can come from one or more sources, including external files and other SAS data sets. Because a DATA step view only reads (opens for input) other files, you cannot update the view's underlying data.

**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**

a unit of character, numeric, or alphanumeric information. This unit is stored as one item in a data record, such as a person's height being stored as one variable (namely, a column or vertical component) in an observation (row).

**data view**

See SAS data view.

**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 management system**

a software application that enables you to create and manipulate data that is stored in the form of databases. Short form: DBMS.

**DBMS**

See database management system.

**delimiter**

a character that serves as a boundary that separates the elements of a text string.

**DIF file**

See Data Interchange Format file.

**engine**

a component of SAS software that reads from or writes to a file. Various engines enable SAS to access different types of file formats.

**format**

See SAS format.

**index**

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.

**interface view engine**

a type of SAS engine that SAS/ACCESS software uses 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).

**library reference**

See libref.

**libref**

a SAS name that is associated with the location of a SAS library. For example, in the name MYLIB.MYFILE, MYLIB is the libref, and MYFILE is a file in the SAS library.

**member**

an element of a dimension. For example, for a dimension that contains time periods, each time period is a member of the dimension.

**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, AUDIT, DMBD, DATA, CATALOG, FDB, INDEX, ITEMSTOR, MDDB, PROGRAM, UTILITY, and VIEW.

**missing value**

a type of value for a variable that contains no data for a particular row or column. By default, SAS writes a missing numeric value as a single period and 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 either one data value or a missing-value indicator for each variable.

**pass-through facility**

See SQL pass-through facility.

**PC Files Server**

a SAS application that receives client requests to access Microsoft Windows-specific data files, such as Microsoft Excel and Microsoft Access. The application runs on both 32-bit and 64-bit Windows, as either a 32-bit or a 64-bit application.

**PROC SQL view**

a SAS data set 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. The output of a PROC SQL view can be either a subset or a superset of one or more files.

**query**

a set of instructions that requests particular information from one or more data sources.

**RDBMS**

See relational database management system.

**record**

See data record, schema record.

**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, Oracle, Sybase, and Microsoft SQL Server.

**rollback**

a data recovery process that restores a database after a hardware or software failure, or that returns it to a state before changes were made.

**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.

**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. Short form: data view.

**SAS format**

a type of SAS language element that applies a pattern to or executes instructions for a data value to be displayed or written as output. Types of formats correspond to the data's type: numeric, character, date, time, or timestamp. The ability to create user-defined formats is also supported. Examples of SAS formats are BINARY and DATE. Short form: format.

**SAS metadata**

metadata that is created by SAS software. Metadata that is in SAS Open Metadata Architecture format is one example.

**SAS variable**

a column in a SAS data set or in a SAS data view. The data values for each variable describe a single characteristic for all observations (rows).

**server**

software that provides either resources or services to requesting clients, possibly over a network.

**SQL**

See Structured Query Language.

**SQL pass-through facility**

the technology that enables SQL query code to be passed to a particular DBMS for processing. Short form: pass-through facility.

**Structured Query Language**

a standardized, high-level query language that is used in relational database management systems to create and manipulate objects in a database management system. SAS implements SQL through the SQL procedure. Short form: SQL.

**type**

See data type.

**variable**

See SAS variable.

**view**

a definition of a virtual data set that is named and stored for later use. A view contains no data; it merely describes or defines data that is stored elsewhere.

**view descriptor**

a SAS/ACCESS file that defines part or all of the DBMS data that is described by an access descriptor.



# Index

---

## Special Characters

= statement  
 IMPORT and EXPORT procedures 31

## A

ACCDB file format 52  
 ACCDESC= option  
 PROC ACCESS statement 230  
 ACCDESC= statement  
 DBLOAD procedure 252  
 access  
 files, PC Files Server 168  
 PC Files Server 168  
 access descriptors  
 creating 232  
 resetting column settings 227  
 updating 244  
 WK<sub>n</sub> files with 271  
 ACCESS engine  
 committing updates 104, 107, 187, 191  
 access methods 3  
 ACCESS procedure 226  
 component for Excel 35  
 statements 229  
 syntax 226  
 ACCESS procedure, DBF (Windows) 279  
 data conversions 280  
 QUIT statement 279  
 syntax 279  
 ACCESS procedure, DIF 283  
 data conversions 285  
 datetime conversions 285  
 DIFLABEL statement 284  
 QUIT statement 283  
 SKIPROWS statement 284  
 syntax 283  
 ACCESS procedure, WK<sub>n</sub> 271  
 data conversions 274  
 GETNAMES statement 272  
 QUIT statement 272

RANGE statement 272  
 SCANTYPE statement 273  
 SKIPROWS statement 273  
 syntax 271  
 WORKSHEET statement 273  
 ACCESS procedure, XLS 260  
 data conversions 260  
 syntax 260  
 ACCESS= LIBNAME option 181  
 Ace queries 144  
 administration  
 PC Files Server 168  
 ASSIGN statement  
 ACCESS procedure 231  
 authentication  
 PC Files Server 167  
 AUTOCOMMIT= argument  
 CONNECT statement 134, 209  
 AUTOCOMMIT= data set option 104,  
 187  
 AUTOCOMMIT= LIBNAME option  
 121, 181  
 autostart  
 PC Files Server 169

## B

buffers  
 reading rows into 118, 202

## C

cells 33  
 character data 34, 63  
 very long character data type 110, 194  
 column names  
 saved as label names 112, 196  
 variable labels as 109, 193  
 columns  
 date format of 119, 202  
 disallowed characters in 108, 192

- NULL values 111, 195
  - renaming 108, 192
  - comma-separated values files
    - See *CSV files*
  - COMMAND\_TIMEOUT=
    - CONNECT statement 134, 210
  - COMMAND\_TIMEOUT= data set option 104, 187
  - COMMAND\_TIMEOUT= LIBNAME
    - option 121, 181
  - commands
    - Jet commands 146
  - commit statements
    - row processing 105, 188
  - configuration
    - PC Files Server 166
  - CONNECT\_STRING= LIBNAME option 175
  - CONNECT\_STRING= option
    - CONNECT statement 132, 208
  - CONNECT statement
    - arguments 134, 209
    - Pass-Through Facility, UNIX 131, 206
  - CONNECT TO component 139, 214
  - CONNECTION\_GROUP= argument
    - CONNECT statement 135, 210
  - CONNECTION\_GROUP= LIBNAME
    - option 122, 182
  - connection options
    - Access files 153
    - Excel workbook files 148
  - CONNECTION= argument
    - CONNECT statement 134, 210
  - CONNECTION= LIBNAME option 121, 181
  - constraints
    - PC Files Server 170
  - conversion
    - date 54
    - SAS Data Sets to Microsoft Access database 54
    - time 54
  - CREATE statement
    - ACCESS procedure 232
  - CREATE TABLE statement
    - data source-specific syntax 107, 190
  - CSV files 29
  - CSV files
    - exporting data sets to 32
    - importing into SAS 32
    - importing subsets into SAS 32
  - CURSOR\_TYPE= argument
    - CONNECT statement 135, 210
  - CURSOR\_TYPE= data set option 105, 188
  - CURSOR\_TYPE= LIBNAME option 122, 182
  - cursor type 105, 188
- D**
- data access methods 3
  - data conversions
    - ACCESS procedure, DBF 280
    - ACCESS procedure, DIF 285
    - ACCESS procedure, WK<sub>n</sub> 274
    - ACCESS procedure, XLS 260
    - DBLOAD procedure, DBF 282
    - DBLOAD procedure, WK<sub>n</sub> 276
    - DBLOAD procedure, XLS 267
  - data set options
    - PC files on Linux and UNIX 186
  - data sets
    - converting dBASE files to 293
    - converting dBASE II files to 296
    - converting to DIF files 299
    - DBF files and 280, 282
    - DIF files and 285
    - exporting to CSV files 32
    - exporting to Excel files 45
    - importing ranges to 45
    - transferring with DIF files 299
    - WK<sub>n</sub> files and 274, 276
  - data source
    - user-defined 11
  - data source tables
    - data type for 114, 197
  - data sources
    - supported 7
  - data sources, supported 7
  - data type
    - DTA 78
  - data types
    - Ace provider 142
    - conversion for Access files 154
    - conversion for Excel workbook files 150
    - DIF 288
    - Excel 34
    - for data source tables 114, 197
    - Jet provider 142
    - JMP 71
    - Microsoft Access database 53
    - overriding 113, 197
    - SAV 75
    - SPSS 75
    - WK<sub>n</sub> 63
  - DATA=
    - DIF procedure 297
  - DATA= argument
    - PROC DBLOAD statement 251

- PROC EXPORT statement 22
- DATA= option
  - DBF procedure 293
- database fields 66
- DATABASE= option
  - statement for Access 57
- DATAFILE= argument
  - PROC IMPORT statement 14
- DATAROW= statement
  - IMPORT and EXPORT procedures 31
- DATATABLE= argument
  - PROC IMPORT statement 15
- date and time values 63
- date format
  - of data source columns 119, 202
- date values 34
- date/time values
  - SAS versus Access 157
  - SAS versus Excel 153
- datetime conversions
  - ACCESS procedure, DIF 285
  - DBLOAD procedure, DIF 287
- datetime values 34
- DB files, Paradox 74
- dBase DBF files 66
  - ACCESS procedure with 279
  - data conversions 280, 282
  - DBLOAD procedure with 280
  - examples of importing and exporting 69
  - missing values 282
  - setting environment variables 68
  - setting system options 68
  - supported IMPORT/EXPORT procedure statements 68
  - transferring other software files to 296
- dBase DBFMEMO files 70
- dBASE files
  - converting to data sets 293
- dBASE II files
  - converting to data sets 296
- DBCMMIT= data set option 105, 188
- DBCMMIT= LIBNAME option 122, 182
- DBCONDITION= data set option 106, 189
- DBCREATE\_TABLE\_OPTS= data set option 107, 190
- DBDSOPTS= option
  - statement for Access 57
  - statement for Excel 37
- DBENCODING = option
  - statement for DBF files 69
- DBENCODING= data set option 190
- DBF fields
  - converting SAS variables to 296
- DBF files
  - See dBase DBF files
- DBF options
  - DBF procedure 293
- DBF procedure 293
- DBFORCE= data set option 107, 191
- DBGEN\_NAME= argument
  - CONNECT statement 135, 211
- DBGEN\_NAME= data set option 108, 192
- DBGEN\_NAME= LIBNAME option 123, 183
- DBKEY= data set option 109, 193
  - format of WHERE clause 112, 196
- DBKEY= processing
  - missing values and 117, 118, 200, 201
- DBLABEL= data set option 109, 193
- DBLOAD procedure 249, 250
  - component for Excel 35
  - naming conventions 250
  - syntax 250
- DBLOAD procedure, DBF (Windows )
  - VERSION statement 281
- DBLOAD procedure, DBF (Windows) 280
  - data conversions 282
  - environment variables 282
  - QUIT statement 281
  - syntax 280
- DBLOAD procedure, DIF 286
  - datetime conversions 287
  - DIFLABEL statement 286
  - FORMAT statement 286
  - QUIT statement 286
  - syntax 286
- DBLOAD procedure, WKn 275
  - data conversions 276
  - FORMAT statement 276
  - PUTNAMES statement 276
  - QUIT statement 276
  - syntax 275
- DBLOAD procedure, XLS 266
  - data conversions 267
- DBMAX\_TEXT= data set option 110, 194
- DBMAX\_TEXT= LIBNAME option 123, 183
- DBMAX\_TEXT= option
  - CONNECT statement 135, 211
- DBMEMO files 70
- DBMS engines
  - trace information from 91
- DBMS specifications
  - EXPORT procedure 22
  - IMPORT procedure 14
- DBMS= argument

- PROC DBLOAD statement 251
  - DBMS= option
    - PROC ACCESS statement 230
    - PROC EXPORT statement 23
    - PROC IMPORT statement 15
  - DBNULL= data set option 111, 195
  - DBNULLKEYS= LIBNAME option 123, 183
  - DBPASSWORD= LIBNAME option 175
  - DBPASSWORD= option
    - CONNECT statement 133, 209
    - statement for Access 57
  - DBPASSWORD=LIBNAME option 100
  - DBSASLABEL= data set option 112, 196
  - DBSASLABEL= LIBNAME option 124, 183
  - DBSASLABEL= option
    - statement for Access 57
    - statement for Excel 38
  - DBSASTYPE= data set option 113, 197
  - DBSYSFILE= LIBNAME option 176
  - DBSYSFILE= option
    - CONNECT statement 133, 209
    - statement for Access 58
  - DBSYSFILE=LIBNAME option 100
  - DBTYPE= data set option 114, 197
  - DEFER= LIBNAME option 124, 184
  - DEFER= option
    - CONNECT statement 136, 211
  - DELETE statement
    - DBLOAD procedure 252
  - delimited files
    - IMPORT and EXPORT procedure statements for 30
    - text files 29
  - delimiter 29
  - delimiter-separated values (DSV) files 29
  - DELIMITER= statement
    - IMPORT and EXPORT procedures 31
  - descriptor files
    - Access Descriptor examples 234
  - desktop application
    - PC Files Server 164
  - desktop applications
    - running PC Files Server as 164
  - DIF
    - procedure 297
  - DIF (Lotus) files
    - ACCESS procedure with 283
    - data conversions 285
    - DBLOAD procedure with 286
  - DIF file data types 288
  - DIF files
    - converting to data sets 297
    - missing values and 299
    - transferring data sets with 299
  - DIF variables
    - converting DIF variables to SAS variables 298
    - converting to SAS variables 298
  - DIFLABEL statement
    - ACCESS procedure 284
    - DBLOAD procedure 286
  - DIRECT\_SQL= LIBNAME option 124, 184
  - DISCONNECT statement
    - Pass-Through Facility, Windows 137, 212
  - DNS= LIBNAME option 176
  - DROP statement
    - ACCESS procedure 234
  - DSN= option
    - CONNECT statement 132, 207
  - DSV files
    - DLM files 29
  - DTA data types 78
  - DTA files 78
- E**
- EFI (External File Interface) 11
  - encoding
    - character set for Access databases 190
    - character set for Excel workbook files 190
  - encryption
    - PC Files Server 167
  - ENDCOL= option
    - statement for Excel 49
  - ENDNAMEROW= option
    - statement for Excel 49
  - ENDROW= option
    - statement for Excel 49
  - environment variables
    - dBase DBF files 68
    - Lotus WKn files 277
    - XLS files 270
  - ERRLIMIT= data set option 115, 198
  - ERRLIMIT= statement
    - DBLOAD procedure 253
  - error limit, before rollback 115, 198
  - Excel 2007
    - XLS files 33
  - Excel files 33
  - Excel workbooks 33
  - EXECUTE statement
    - Pass-Through Facility, Windows 137, 213
  - export
    - SPSS files 77
    - Stata data files 80
  - EXPORT procedure 21

- Access table specification compatibility
  - 14, 22
- DBMS specifications 22
- Excel spreadsheet specification
  - compatibility 22
  - features of 7
  - statements for delimited files 30
  - supported data sources and platforms 7
  - syntax 22
- export utilities 35
- Export Wizard 11
  - features of 7
  - start 11
  - supported data sources and platforms 7
- exporting
  - Access files 54
  - data sets to Excel files 45
  - DB files 75
  - DBF files 68, 69
  - DTA files 80
  - Excel files with LIBNAME statement 35
  - JMP files 73, 74
  - WKn files 63, 65
- External File Interface (EFI) 11
- extracting data 228
  - view descriptors for 228

**F**

- file formats
  - ACCDB 52
  - Access database files 52
  - CSV 29
  - DBF 66
  - DBFMEMO 70
  - DTA 78
  - JMP 71
  - MDB 52, 54
  - Paradox DB 74
  - SAV 75
  - TAB 29
  - WKn 62
  - XLS 33, 35
  - XLSB 33, 45
  - XLSM 33
  - XLSX 33, 35
- FILELOCK= LIBNAME option 125
- FMTLIB= option
  - statement for exporting JMP files 73
  - statement for exporting SPSS files 77
  - statement for importing JMP files 73
  - statement for importing SPSS files 77
- FMTLIB= statement
  - DBLOAD procedure 253
- FORMAT statement

- ACCESS procedure 235
- DBLOAD procedure 276, 286
- functions 86, 174

**G**

- GETDELETED= option
  - statement for DBF files 69
- GETNAMES statement
  - ACCESS procedure 272
- GETNAMES= option
  - statement for Excel 38, 47, 50
  - statement for Lotus 1–2–3 64
- GETNAMES= statement
  - IMPORT and EXPORT procedures 31
- GUESSINGROWS= statement
  - IMPORT and EXPORT procedures 31

**I**

- import
  - SPSS files 77
  - Stata data files 80
- IMPORT procedure 13
  - DBMS specifications 14
  - Excel spreadsheet specification
    - compatibility 14
  - Excel spreadsheet specifications 14
  - features of 7
  - statements for delimited files 30
  - supported data sources and platforms 7
- import utilities 35
- Import Wizard 11
  - start 11
  - supported data sources and platforms 7
- importing
  - Access files 54
  - DB files 75
  - DBF files 68, 69
  - DTA files 80
  - Excel files with LIBNAME statement 35
  - JMP files 73, 74
  - ranges to data sets 45
  - WKn files 63, 65
- INIT= LIBNAME option 99
- INSERT\_SQL= data set option 116, 199
- INSERT\_SQL= LIBNAME option 125
- insert processing
  - missing values and 117, 118, 200, 201
- INSERTBUFF= data set option 116, 199
- INSERTBUFF= LIBNAME option 125, 184
- installing PC Files Server 163

**J**

- Jet commands 145
- Jet queries 144
- JMP data types 71
- JMP engine LIBNAME statement 291
- JMP files 71
  - examples of importing and exporting 73, 74
  - importing and exporting data in 73
  - missing values 71
- joins
  - performance improvement 109, 193

**L**

- LABEL option
  - PROC EXPORT statement 26
- LABEL statement
  - DBLOAD procedure 254
- LABELS
  - DIF procedure 298
- LIBNAME connection options
  - Access files 153
  - Excel workbook files 148
- LIBNAME engines 89
  - macro variables 90
  - software requirements 90
  - trace system options 91
- LIBNAME options
  - PC files on Linux and UNIX 181
- LIBNAME options, PC files 121
  - syntax 121
- LIBNAME statement, for JMP Engine 291
- LIBNAME statement, PC files 35, 97
  - assigning librefs 101
  - assigning LIBREFS 86
  - connection options 99
  - for Access files 153
  - for Excel workbook files 147
  - functions with PC files data 86
  - importing and exporting Excel files 35
  - on 64-bit Microsoft Windows 174
  - on Linux 174
  - on UNIX 174
  - sorting data 85
  - syntax 97
  - syntax, on Linux 174
  - syntax, on UNIX 174
- libraries
  - disassociating librefs from 178
  - writing attributes to log 101, 178
- librefs
  - assigning 101
  - assigning to Access databases 178
  - assigning to Excel workbooks 179

- assigning to Oracle databases 180
- assigning to SQL Server databases 180
- disassociating 101, 178

**LIBREFS**

- assigning 86

**LIMIT= statement**

- DBLOAD procedure 254

**Linux**

- data set options 186
- LIBNAME options 181
- LIBNAME statement 174

**LIST statement**

- ACCESS procedure 236
- DBLOAD procedure 254

**LOAD statement**

- DBLOAD procedure 251, 255

**log**

- writing library attributes to 101, 178

**Lotus DIF files**

- ACCESS procedure with 283
- data conversions 285
- DBLOAD procedure with 286

**Lotus WKn data types 63****Lotus WKn files 62**

- ACCESS procedure with 271
- data conversions 274, 276
- DBLOAD procedure 275
- environment variables 277
- examples of exporting and importing 65

- import and export components and statements 63

**M****macro variables 90****MDB file format 52**

- IMPORT and EXPORT procedure statements 54

**MEMOSIZE= option**

- statement for Access 58

**META= option**

- statement for exporting JMP files 73
- statement for importing JMP files 73

**Microsoft Access**

- assigning librefs to databases 178
- encoding character set for database 190
- table specification compatibility 14, 22

**Microsoft Access Database**

- conversion 54

**Microsoft Access database data types 53****Microsoft Access database files 52**

- IMPORT and EXPORT procedure statements for 54

- with IMPORT and EXPORT procedures 54

- Microsoft Access files
    - data types conversion for 154
    - file-specific reference 153
    - LIBNAME connection options for 153
    - LIBNAME statements for 153
  - Microsoft Ace provider data types 142
  - Microsoft Excel
    - assigning librefs to workbooks 179
    - data types 34
    - DBMS spreadsheet specifications 14
    - import and export components 35
    - LIBNAME statement, PC files on
      - UNIX 177
    - spreadsheet specification compatibility
      - 14, 22
    - workbooks 33
  - Microsoft Excel 2007 33
    - Excel 2007 45
  - Microsoft Excel files 33
    - exporting data sets to 45
    - importing ranges to data sets 45
    - importing/exporting with LIBNAME
      - statement 35
  - Microsoft Excel workbook files 33
    - data types conversion for 150
    - encoding character set for 190
    - file-specific reference 147
    - LIBNAME connection options for 148
    - LIBNAME statements for 147
  - Microsoft Jet provider data types 142
  - missing values
    - DBF files 282
    - DBKEY= processing and 117, 118, 200, 201
    - DIF file conversions 299
    - insert processing and 117, 118, 200, 201
    - JMP files 71
    - update processing and 117, 118, 200, 201
  - MIXED statement
    - ACCESS procedure 237
  - MIXED= option
    - statement for Excel 38
  - MSENGINE= LIBNAME option 99, 125, 176
  - MSENGINE= option
    - CONNECT statement 133, 209
- N**
- NAMEROW= option
    - statement for Excel 50
  - naming conventions
    - DBLOAD procedure 250
  - NEWFILE= option
    - statement for Excel 41, 50
  - NULL values
    - in columns 111, 195
  - NULLCHAR= data set option 117, 200
  - numeric data 34, 63
- O**
- Oracle
    - assigning librefs to databases 180
  - ordering PC files data 106, 189
  - OUT=
    - DIF procedure 297
  - OUT= argument
    - PROC IMPORT statement 19
  - OUT= option
    - DBF procedure 293
    - PROC ACCESS statement 230
  - OUTFILE= argument
    - PROC EXPORT statement 22
  - OUTTABLE= argument
    - PROC EXPORT statement 23
  - overriding data types 113, 197
- P**
- Paradox DB files 74
    - examples of importing and exporting 75
  - Pass-Through Facility 129
  - Pass-Through Facility, UNIX 205
    - return codes 206
    - syntax 206
  - Pass-Through Facility, Windows
    - DISCONNECT statement 137, 212
    - syntax 130
  - PASSWORD= LIBNAME option 176
  - PASSWORD= option
    - CONNECT statement 133, 209
  - PASSWORD=LIBNAME option 100
  - PATH= argument
    - CONNECT statement 139, 214
  - PATH= LIBNAME option 99, 176
  - PATH= option
    - CONNECT statement 132, 208
  - PATH= statement
    - ACCESS procedure 237
    - DBLOAD procedure 251, 256
  - PC files
    - ACCESS procedure 226
    - assigning librefs 101
    - assigning LIBREFS 86
    - data access methods 3
    - DBLOAD procedure 249
    - functions with PC files data 86, 174
    - Pass-Through Facility on UNIX 205

- sorting data 85, 106, 174, 189
  - subsetting data 106, 189
- PC Files Server 162
  - access to 168
  - administration 168
  - application modes (32-bit versus 64-bit) 163
  - authentication 167
  - autostart 169
  - configuration 166
  - constraints 170
  - data encryption 167
  - desktop application 164
  - installation 163
  - maximum connections 167
  - operating modes 163
  - port number 163, 167
  - running as desktop application 164
  - security 167
  - service name 167
  - shared information 171
  - user accounts 170
  - Windows service 163
- PCFFSCL.SAS file 262
- PCFILES
  - queries 219
- performance
  - joins 109, 193
- platforms, supported 7
- port number
  - PC Files Server 163, 167
- PORT= LIBNAME option 176
- PORT= option
  - CONNECT statement 132, 208
  - statement for Access 58
  - statement for Excel 41
- PREFIX=
  - DIF procedure 298
- printing
  - SASTRACE information 94
- PROC DBLOAD statement 251
- PROC EXPORT statement 22
- PROC IMPORT statement 13
- PROC SQL 129
- procedure 226, 297
- PROMPT= argument
  - CONNECT statement 139, 214
- PROMPT= LIBNAME option 99
- PUTNAMES statement
  - DLOAD procedure 276
- PUTNAMES= option
  - statement for Excel 50

**Q**

- queries 219

- Jet and Ace 144
- QUIT statement
  - ACCESS procedure 238, 272, 279, 283
  - DBLOAD procedure 256, 276, 281, 286

**R**

- RANGE statement
  - ACCESS procedure 272
- RANGE= option
  - statement for Excel 41, 47, 50
  - statement for Lotus 1–2–3 64
- ranges 33
  - importing to data sets 45
- READBUFF= data set option 118, 202
- READBUFF= LIBNAME option 126, 185
- READBUFF= option
  - CONNECT statement 136, 211
- registry editor
  - TypeGuessRows 39
- RENAME statement
  - ACCESS procedure 239
  - DBLOAD procedure 256
- REPLACE option
  - PROC EXPORT statement 26
  - PROC IMPORT statement 19
- RESET statement
  - ACCESS procedure 240
  - DBLOAD procedure 257
- return codes
  - Pass-Through Facility, UNIX 206
- rollback, error limit for 115, 198
- row processing
  - commit statement for 105, 188
- rows
  - insertion method 116, 199
  - number for single insert 116, 199
  - number to read into buffer 118, 202

**S**

- sample data 4
- SASDATEFMT= data set option 119, 202
- SASTRACE= system option 91
  - printing trace information 94
- SAV data types 75
- SAV files 75
- SCAN\_TEXTSIZE= LIBNAME option 185
- SCAN\_TIMETYPE= LIBNAME option 126, 185
- SCANMEMO= option
  - statement for Access 58

- SCANSCANTIME= option
    - statement for Access 59
  - SCANTEXT= option
    - statement for Excel 42
  - SCANTIME= option
    - statement for Excel 43
  - SCANTYPE statement
    - ACCESS procedure 273
  - security
    - model, PC Files Server 168
    - PC Files Server 167
    - policy, PC Files Server 169
    - user accounts, PC Files Server 170
  - SELECT statement
    - ACCESS procedure 241
  - SERVER= LIBNAME option 176
  - SERVER= option
    - CONNECT statement 132, 208
    - statement for Access 59
    - statement for Excel 43
  - SERVERPASS= option
    - CONNECT statement 133, 208
    - statement for Access 59
    - statement for Excel 43
  - SERVERUSER= option
    - CONNECT statement 133, 208
    - statement for Access 59
    - statement for Excel 43
  - service name
    - PC Files Server 167
  - SERVICE= option
    - statement for Access 60
    - statement for Excel 44
  - shared information
    - PC Files Server 171
  - SHEET= option
    - statement for Excel 44, 48, 51
    - statement for Lotus 1–2–3 65
  - SKIP=
    - DIF procedure 298
  - SKIPROWS statement
    - ACCESS procedure 273, 284
  - software requirements, for LIBNAME engine 90
  - sorting PC files data 85, 106, 174, 189
  - space-separated data values
    - importing into SAS 32
  - special PCFILES queries 219
  - SPOOL= LIBNAME option 126, 185
  - SPSS SAV data types 75
  - SPSS SAV files 75
  - SQL Procedure 130
  - SQL Server
    - assigning librefs to databases 180
  - SQLXMSG macro variable 206
  - SQLXRC macro variable 206
  - SS\_MIXED environment variable 278
  - SS\_NAMES environment variable 278
  - SS\_SCAN environment variable 278
  - SSPI= option
    - CONNECT statement 133, 208
    - statement for Access 60
    - statement for Excel 44
  - STARTCOL= option
    - statement for Excel 51
  - STARTROW= option
    - statement for Excel 51
  - Stata DTA data types 78
  - Stata DTA files 78
    - examples of importing and exporting 80
  - STRINGDATES= LIBNAME option 127, 186
  - STRINGDATES= option
    - CONNECT statement 136, 211
  - SUBSET statement
    - ACCESS procedure 226
  - subsetting
    - PC files data 106, 189
  - sub—setting
    - importing CSV file subsets 32
  - support
    - JMP files 3
  - supported data sources 3, 7, 11
  - supported platforms 7
  - system options
    - dBase DBF files 68
    - for tracing 91
- T**
- TAB files 29
  - TEXTSIZE= option
    - statement for Excel 45
  - time values 34
  - trace information
    - from DBMS engine 91
  - trace system options 91
  - TYPE statement
    - ACCESS procedure 226
- U**
- UDL= LIBNAME option 99
  - UNICODE= LIBNAME option 127
  - UNIQUE statement
    - ACCESS procedure 242
  - UNIX
    - data set options 186
    - LIBNAME options 181
    - LIBNAME statement 174
    - Pass-Through Facility 205

- update processing
    - missing values and 117, 118, 200, 201
  - UPDATE statement
    - ACCESS procedure 243
  - updating data
    - missing values and 282
  - USE\_DATATYPE= LIBNAME option
    - 127, 186
  - USEDATE= option
    - CONNECT statement 136, 212
    - statement for Access 60
    - statement for Excel 45
  - USER= LIBNAME option 177
  - USER= option
    - CONNECT statement 134, 209
  - USER=LIBNAME option 100
  - utilities
    - import and export 35
- V**
- variables
    - converting SAS variables to DBF fields
      - 293
    - DBF files and 280, 282
    - DIF files and 285
    - WKn files and 274, 276
  - VERSION statement
    - DBLOAD procedure 281
  - VERSION= LIBNAME option 134, 177, 209
  - VERSION= option
    - statement for Access 60
    - statement for Excel 45
  - VERSION=LIBNAME option 100
  - very long character data type 110, 194
  - view descriptors 233
    - creating 233
    - extracting data with 228
    - re-setting column settings 228
  - updating 244
  - VIEWDESC= option
    - PROC ACCESS statement 230
- W**
- WHERE clause
    - format of, with DBKEY= data set
      - option 112, 196
  - WHERE statement
    - DBLOAD procedure 258
  - Windows service
    - PC Files Server 163
  - Wizard
    - Export 11
    - Import 11
  - WKn data types 63
  - WKn files
    - See also* Lotus WKn files
    - ACCESS procedure 271
    - data conversions 274, 276
    - DBLOAD procedure 275
    - environment variables 277
  - WORKSHEET statement
    - ACCESS procedure 273
  - worksheets 33
- X**
- XLS file formats 35
  - XLS files
    - ACCESS procedure 260
    - data conversions 260, 267
    - DBLOAD procedure 266
    - environment variables 270
  - XLSB files 33, 45
  - XLSM files 33
  - XLSX file formats 35
  - XLSX files 33