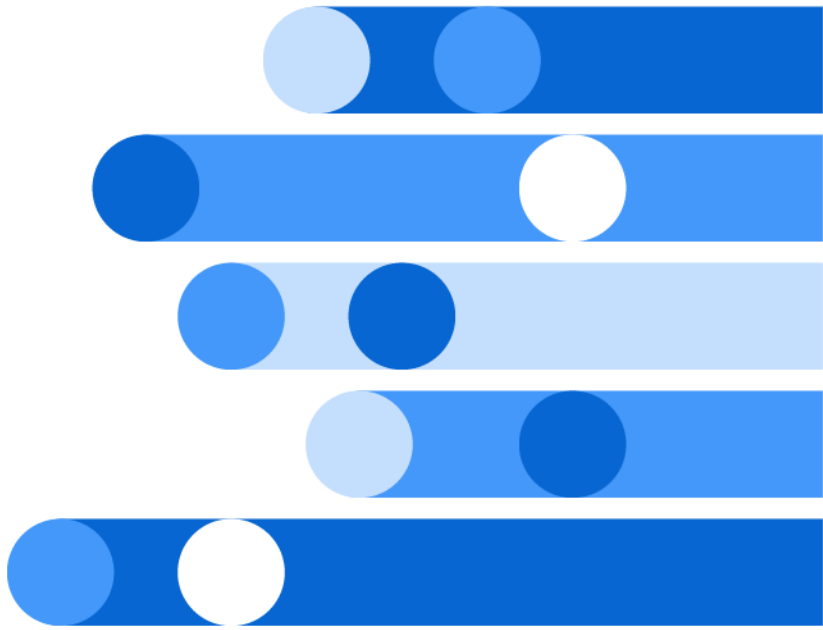
SAS/ACCESS® 9.4 Interface to PC Files: Reference, Fourth Edition



SAS® Documentation
October 30, 2024

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2016. *SAS/ACCESS® 9.4 Interface to PC Files: Reference, Fourth Edition*. Cary, NC: SAS Institute Inc.

SAS/ACCESS® 9.4 Interface to PC Files: Reference, Fourth Edition

Copyright © 2016, SAS Institute Inc., Cary, NC, USA

All Rights Reserved. Produced in the United States of America.

For a hard copy 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 License Rights; Restricted Rights: The Software and its documentation is commercial computer software developed at private expense and is provided with RESTRICTED RIGHTS to the United States Government. Use, duplication, or disclosure of the Software by the United States Government is subject to the license terms of this Agreement pursuant to, as applicable, FAR 12.212, DFAR 227.7202-1(a), DFAR 227.7202-3(a), and DFAR 227.7202-4, and, to the extent required under U.S. federal law, the minimum restricted rights as set out in FAR 52.227-19 (DEC 2007). If FAR 52.227-19 is applicable, this provision serves as notice under clause (c) thereof and no other notice is required to be affixed to the Software or documentation. The Government's rights in Software and documentation shall be only those set forth in this Agreement.

SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

November 2024

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 trademarks of their respective companies.

9.4-P9:acpcref

Contents

What's New in SAS/ACCESS 9.4 Interface to PC Files . . . . . *vii*

PART 1 Introduction to SAS/ACCESS 9.4 Interface to PC Files 1

Chapter 1 / Working with SAS/ACCESS 9.4 Interface to PC Files **3**
Methods for Accessing PC Files Data 3
Import and Export Wizards 4
IMPORT and EXPORT Procedures 4
LIBNAME Engines 4
ACCESS, DBLOAD, DBF, and DIF Procedures 5
Using This Document 5
Sample Data in This Document 6

PART 2 Import and Export Wizards and Procedures 7

Chapter 2 / Supported Data Sources and Environments **9**
Reading and Writing Data between SAS Data Sets and PC Files 9
Reading and Writing Data between CAS Tables and PC Files 10
Supported Data Sources and Environments 10
When is the PC Server Required? 14

Chapter 3 / Using SAS Import and Export Wizards **15**
Importing and Exporting Data with the Wizards 15
Using the Wizards and the PC File Server 16
Using the External File Interface (EFI) 29

Chapter 4 / IMPORT Procedure **31**
Overview: IMPORT Procedure 31
Syntax: IMPORT Procedure 32

Chapter 5 / EXPORT Procedure **41**
Overview: EXPORT Procedure 41
Syntax: EXPORT Procedure 42

Chapter 6 / File Format-Specific Reference for the IMPORT and EXPORT Procedures **51**
Delimited Files 53
Microsoft Workbook Files 62
Microsoft Access Database Files 93

Lotus 1-2-3 WKn Files 107
dBase DBF Files 112
dBase DBF MEMO Files 118
JMP Files 119
Paradox DB File Formats 126
SPSS SAV Files 127
Stata DTA Files 134

PART 3 SAS/ACCESS Interface to PC Files LIBNAME Engines 141

Chapter 7 / SAS/ACCESS LIBNAME Engines **143**
Overview of SAS LIBNAME Engines for Microsoft Excel and Microsoft Access 143
Software Requirements 144
Comparing SAS LIBNAME Engines for PC Files Data 145
Assigning a Libref Interactively 147
Sorting PC Files Data 149
Using SAS Functions with PC Files Data 149
Macro Variables 149
System Options 151
Dictionary 151

PART 4 LIBNAME Statements for Excel and Access Engines on Microsoft Windows 157

Chapter 8 / LIBNAME Statement for Excel and Access Engines on Microsoft Windows **159**
Dictionary 159

Chapter 9 / LIBNAME Statement Options for Excel and Access Engines **171**
Dictionary 171

Chapter 10 / Data Set Options for Excel and Access LIBNAME Statement **179**
Overview of SAS Data Set Options for the Excel and Access LIBNAME Statement 180
Dictionary 180

Chapter 11 / LIBNAME Statement: File-Specific Reference for Excel and Access Engines **201**
Microsoft Excel Workbook Files 201
Microsoft Access Database Files 209

Chapter 12 / SQL Pass-Through Facility for Excel and Access Engines **217**
Overview: SAS SQL Pass-Through Facility for Excel and Access Engines 217
Dictionary 218

PART 5 LIBNAME Statement for XLSX and JMP Engines under UNIX and Microsoft Windows 239

Chapter 13 / LIBNAME Statement for the XLSX Engine under UNIX and Microsoft Windows **241**
Dictionary 241

Chapter 14 / LIBNAME Statement for the JMP Engine under UNIX and Microsoft Windows **247**
Dictionary 247

PART 6 LIBNAME Statement for the PC Files Engine with the PC File Server 251

Chapter 15 / LIBNAME Statement: PC Files Engine with the PC File Server **253**
LIBNAME Statement for PC Files Engine on UNIX and Microsoft Windows 254
Dictionary 255

Chapter 16 / SQL Pass-Through Facility for the PC Files Engine **291**
Overview: SQL Pass-Through Facility for the PC Files Engine 291
Dictionary 292

Chapter 17 / Special PC Files Query Support on Microsoft Windows and UNIX **307**
Special PC Files Queries on Microsoft Windows and UNIX 307
Macro Variable to Pass Content to PC Files Queries 309

PART 7 ACCESS and DBLOAD Procedures 311

Chapter 18 / ACCESS Procedure **313**
Overview: ACCESS Procedure 314
Concepts: ACCESS Procedure 315
Syntax: ACCESS Procedure 318
Usage: ACCESS Procedure 338
Results: ACCESS Procedure 339

Chapter 19 / DBLOAD Procedure **341**
Overview: DBLOAD Procedure 341
Syntax: DBLOAD Procedure 343

Chapter 20 / File-Specific Reference for the ACCESS and DBLOAD Procedures **355**
Overview: ACCESS Procedure and DBLOAD Procedure 356
ACCESS Procedure: XLS Files 356
DBLOAD Procedure: XLS Specifics 363
ACCESS Procedure: WKn Specifics 369
DBLOAD Procedure: WKn Specifics 373
ACCESS Procedure: DBF Specifics 377
DBLOAD Procedure: DBF Specifics (Windows) 379

ACCESS Procedure: DIF Specifics 382
DBLOAD Procedure: DIF Specifics 385

PART 8 Appendixes 389

What's New in SAS/ACCESS 9.4 Interface to PC Files

Overview

SAS/ACCESS Interface to PC Files enables you to import and export supported PC file formats between the original source format and SAS data sets or CAS files. Files are moved between the native PC format and SAS using IMPORT and EXPORT procedures, wizards, or through LIBNAME statements.

General Enhancements

General enhancements found in this or earlier releases include the following.

- In SAS 9.4M8, support was added for the PRESERVE_COMMENTS= LIBNAME option and the PCFILES_SQL_COMMENT macro variable.
- In SAS Viya 3.5, PC Files added PLX (Power UNIX) to the host builds.
- In SAS 9.4 and SAS Viya 3.5, the XLSX engine supports the DBLABEL= data set option.
- In SAS 9.4M623, a new section was added named .XLSB Files on Linux on page 66.
- Beginning in SAS Viya 3.4, Stata 15 is supported for >32,767 variables, >2B rows (on import only), and UTF8.
- In SAS Viya 3.4 and earlier, the SAS PC Files Server is not included. You can connect to the SAS PC Files Server from a SAS 9 client.
- Beginning in SAS 9.4M5, for the IMPORT procedure, the specification DBMS=XLSX now supports storing currency values using the NLMNY format, which formats the value according to the locale.

- In SAS 9.4M5, the documentation has been merged with SAS Viya documentation. Functionality that is available in SAS 9.4M5 but not in SAS Viya is annotated.
- Beginning in SAS 9.4M5, you can import data to CAS tables with PROC IMPORT and the CAS LIBNAME engine. PROC EXPORT can store VARCHARs in SPSS, Stata, JMP, and Excel .xlsx files
- In SAS Viya 3.3 and earlier, the following functionality is not supported in SAS Viya: Microsoft Windows, SAS PC File Server, IMPORT and EXPORT Wizards and EFI, Microsoft Access database files, Lotus files, EXCEL, and PC Files LIBNAME engines. These components are supported in SAS 9.4M5 client for SAS Viya.
- Starting in SAS 9.4M4, SAS/ACCESS Interface to PC Files supports creating new .xlsx output files when using the EXPORT procedure with DBMS=EXCELCS or when using the PC Files LIBNAME engine. Creating files with .xlsm extension is no longer supported with PROC EXPORT DBMS=EXCELCS or the PC Files LIBNAME engine.
- Starting in SAS 9.4M2, SAS/ACCESS, Interface to PC Files has added a new SAS LIBNAME engine named XLSX. It enables you to directly read and write data between Microsoft Excel .xlsx files and SAS on UNIX and Microsoft Windows operating environments. For more information, see LIBNAME Statement: XLSX Engine on page 241.
- SAS/ACCESS 9.4 Interface to PC Files has a number of enhancements to support Microsoft Excel .xlsx files. For example, you can add a new Microsoft Excel .xlsx worksheet to an existing workbook, and you can replace an existing worksheet in a workbook. For more information, see "Import and Export Microsoft Excel Files Using the XLS and XLSX Identifiers" on page 83.

  Note: SAS recommends that you move to the .xlsx file format for enhancements and the expanding support that this format is receiving.

- Starting in SAS 9.4, SAS/ACCESS can export Excel .xlsx files directly to UNIX. In this release, you can specify DBMS=XLSX to read and write to Excel workbooks under UNIX and Microsoft Windows directly without going through the SAS PC Files Server.
- Beginning in SAS 9.4 SAS/ACCESS imports data from JMP files saved in JMP 7 or later formats, and it exports data to files in JMP 7 or later formats. File formats in JMP 3 through JMP 6 are no longer supported. Support for these newer file formats enables you to access JMP files such as those from the JMP Graph Builder iPad app. For more information, see "JMP Files" on page 119.

  The META data type for JMP files has been replaced by support for extended attributes. META is ignored in programs, and a NOTE is generated about it in the SAS log.

  JMP variable names can be up to 255 characters in length.

  Because you can use the Base SAS IMPORT and EXPORT procedures on JMP files without a SAS/ACCESS license, the above information applies to SAS 9.4 as well.

- In SAS 9.4M2, the documentation has revisions and certain enhancements. For example, *Reading Delimited Text Files Into SAS*, has been incorporated into "Delimited Files" on page 53.
- In SAS 9.4M1, the documentation has revisions and certain enhancements. For example, information has been added and updated from two SAS Usage Notes (45340 and 44197). These new topics give display-by-display descriptions of how to use the Import and Export Wizards and the SAS PC Files Server on Windows X64 with 32-bit Microsoft Office applications. For more information, see "Importing and Exporting Data with the Wizards" on page 15.
- The SAS LIBNAME options FILELOCK=, CONNECTION=, and CONNECTION_GROUP are no longer supported. See the SAS 9.3 documentation for descriptions of these options.

Import and Export Procedures and Wizards

- Certain statements used with the IMPORT and EXPORT procedures have been updated: RANGE=, SHEET=, GETNAMES=, GUESSINGROWS=. How RANGE= and SHEET= handle special characters in names has changed in Microsoft Excel .xlsx files.
- In SAS 9.4M1, when GETNAMES=YES is specified, columns that begin with a special character are prefixed with an underscore (_) when importing both .xlsx and .xls files. When GETNAMES=NO, PROC IMPORT generates columns named A, B, C, and so on. For more information, see "Import and Export Microsoft Excel Files Using the XLS and XLSX Identifiers" on page 83.
- In SAS 9.4M1, when importing and exporting .csv files, a SAS data set name can now contain a single quotation mark when the VALIDMEMNAME=EXTEND system option is specified. Using VALIDMEMNAME= expands the rules for SAS data set names. For more information, see "External File Interface (EFI)" on page 55.
- The SAS/ACCESS 9.4 Interface to PC Files (and SAS 9.4) support importing and exporting JMP files with more than 32,767 variables.
- The META statement is used only for importing and exporting JMP files. META is no longer supported and is ignored. Instead, extended attributes are automatically used. If a file has extended attributes, they are automatically transferred to the new file when that file is imported or exported. For example, when importing a JMP file with extended attributes, the attributes are automatically attached to the new SAS data set.
- The ROWSTATE data type is generated by JMP and is used to store several row-level characteristics. If the JMP file contains row state information, PROC IMPORT stores this information as a new variable with the name **`_rowstate_`**. If

PROC EXPORT sees a column named **_rowstate_**, it converts it back into row state information in the output JMP file.

- Starting in SAS 9.4, when you import a Stata .dta file, SAS can read multiple Stata missing values and map them to multiple SAS special missing values, `.a` –`.z` or as a single dot (`.`), for up to 27 missing values.

Note: The Import and Export Wizards are not included in SAS Viya. You can use them from the SAS 9 client.

SAS PC Files Server

In SAS Viya 3.5, the DBINDEX= LIBNAME option and data set option were added so that the SAS PC Files Server can use UTF-8 and Latin9 encoding.

Note: The SAS PC Files Server is not included in SAS Viya 3.5 and earlier. However, you can use the SAS 9 client to connect to the server and access data that is stored on the server. For download instructions, see *SAS PC Files Server: Installation and Configuration Guide*.

The SAS PC Files Server is not included in SAS Viya 3.4 and earlier. However, you can use the SAS 9 client to connect to the server and access data stored on the server.

Starting in SAS 9.4M5, you can now specify a log file location. For more information, see *SAS PC Files Server: Installation and Configuration Guide*.

Starting in SAS 9.4M2, information about the installation, configuration, security, and administration of the SAS PC Files Server has been moved to a separate document. For more information, see *SAS PC Files Server: Installation and Configuration Guide*.

For an introduction to the SAS PC Files Server and its role in the PC Files LIBNAME engine, see " PC Files LIBNAME Engine and the PC Files Server" on page 254.

The SAS PC Files Server can be operated as a Microsoft Windows Service or as a Windows application on the 64-bit Windows operating system. Starting in SAS/ACCESS 9.4, the 64-bit version of SAS PC Files Server is the default if the machine does not have Microsoft Office or a Microsoft ACE driver already installed on it. For more information, see *SAS PC Files Server: Installation and Configuration Guide*.

PART 1

Introduction to SAS/ACCESS 9.4 Interface to PC Files

Chapter 1
Working with SAS/ACCESS 9.4 Interface to PC Files 3

1

Working with SAS/ACCESS 9.4 Interface to PC Files

Methods for Accessing PC Files Data . . . 3
Import and Export Wizards . . . 4
IMPORT and EXPORT Procedures . . . 4
LIBNAME Engines . . . 4
ACCESS, DBLOAD, DBF, and DIF Procedures . . . 5
Using This Document . . . 5
Sample Data in This Document . . . 6

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 Microsoft Excel workbook files as well as in other PC file formats.

There are several methods for accessing PC files data:

- Import and Export Wizards
- IMPORT and EXPORT procedures
- LIBNAME engines
- ACCESS, DBLOAD, DBF, and DIF procedures

With these utilities, you can read and write data to or from a SAS data set for the following formats:

- Microsoft Access
- Microsoft Excel

- Lotus 1–2–3
- dBase
- JMP
- SPSS
- Stata
- Paradox

Not every PC file format is available for all methods or under every operating environment. For information about supported file formats and operating environments, see Table 2.1 on page 11.

Import and Export Wizards

The import and export wizards provide a user interface to guide you through the steps to access your data. See "Importing and Exporting Data with the Wizards" on page 15.

IMPORT and EXPORT Procedures

The IMPORT and EXPORT procedures are part of SAS Foundation software. You can use these procedures without a license for SAS/ACCESS Interface to PC files. Without a SAS/ACCESS Interface to PC Files license, access is available only to JMP 7 or later files and delimited files such as .csv and .tab. You can also use other SAS methods to import delimited data, such as the SAS INFILE statement.

These procedures use a database identifier as part of the syntax to identify the PC file format to read or write. For example, DBMS=SAV specifies that the files that you want to read or write are SPSS .sav files. DBMS=XLSX specifies Microsoft Excel .xlsx files. In this book, this specification is called the DBMS identifier. For more information, see Table 2.1 on page 11.

LIBNAME Engines

LIBNAME engines are provided for Microsoft Excel workbooks and Microsoft Access databases. Some LIBNAME engines support the SQL pass-through of queries directly to the SQL procedure.

To use the Access LIBNAME engine or the Excel LIBNAME engine, you must have Microsoft ACE (Access Database Engine) software installed. The XLSX LIBNAME statement does not use the ACE software. For more information, see LIBNAME Statement: PC Files on page 159 or LIBNAME Statement: XLSX Engine on page 241.

To use the PC Files LIBNAME statement, you must have the SAS PC Files Server installed on Microsoft Windows. The SAS PC Files Server is not included with SAS Viya 3.4 and earlier but you can connect to it using a SAS 9 client.

To use the SQL pass-through facility, see "Overview: SAS SQL Pass-Through Facility for Excel and Access Engines" on page 217.

ACCESS, DBLOAD, DBF, and DIF Procedures

These are specialized procedures that were first introduced in SAS Version 6.

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, "ACCESS Procedure," on page 313 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, "DBLOAD Procedure," on page 341 for additional information.

On UNIX and Windows, you can use the DBF procedure to convert formatted data between dBase (.dbf) files and SAS data sets. The DIF procedure enables you to convert between files in the data interchange format (DIF) and SAS data sets. For more information, see Appendix 1, "DBF Procedure," on page 391 and Appendix 2, "DIF Procedure," on page 397.

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 and formats.
- 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. Samples can be found at the following:

- for UNIX installations, samples are located in **`SASHome/SASFoundation/9.4/samples/`*`engine`*** or **`SASHome/SASFoundation/9.4/samples/dbi`**
- for Windows installations, samples are located in **`SASHOME\SASFoundation\9.4\access\sample`**

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

Chapter 2
Supported Data Sources and Environments 9

Chapter 3
Using SAS Import and Export Wizards 15

Chapter 4
IMPORT Procedure 31

Chapter 5
EXPORT Procedure 41

Chapter 6
File Format-Specific Reference for the IMPORT and EXPORT Procedures 51

2

Supported Data Sources and Environments

Reading and Writing Data between SAS Data Sets and PC Files 9
Reading and Writing Data between CAS Tables and PC Files 10
Supported Data Sources and Environments 10
When is the PC Server Required? 14

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, "IMPORT Procedure," on page 31, Chapter 5, "EXPORT Procedure," on page 41, and Chapter 3, "Using SAS Import and Export Wizards," on page 15.

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

Beginning in SAS 9.4M5, certain PC file formats can access CAS tables that are in memory on a CAS server by using the CAS LIBNAME engine. File formats include Excel, JMP, SPSS, and Stata file.

Note: The Wizards do not support CAS tables.

Reading and Writing Data between CAS Tables and PC Files

To read and write data between CAS tables and external PC files, you must establish a connection to the CAS server using the CAS engine and specify an appropriate server port and a server session reference. For information about connecting to the CAS server, see *SAS Cloud Analytic Services: User's Guide*. For examples of importing to a CAS table or exporting from a CAS table to a supported PC file format, see Chapter 6, "File Format-Specific Reference for the IMPORT and EXPORT Procedures".

Supported Data Sources and Environments

The IMPORT and EXPORT procedures work within the limited range of available PC file formats when SAS is running on UNIX. The procedures work with a wider range of PC file formats when SAS is running on Windows.

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 available only to JMP files, and delimited files such as .csv and .tab delimited files.

In 9.4M5, you can access CAS data sets that are in memory on the CAS server by using the CAS LIBNAME engine.

Note: The Wizards do not support CAS tables.

Alternate methods to access data are described in "ACCESS, DBLOAD, DBF, and DIF Procedures" on page 5.

See the SAS support site for Supported Operating Systems for more detailed information.

Note: SAS Viya runs only on Linux operating system and supports only UTF-8 encoding.

Table 2.1 *Data Source and Environment Support Summary*

Data Source		Import		Export		Supported Platforms
Description	DBMS= Identifier	Wizard	PROC	Wizard	PROC	
Excel files using the Excel LIBNAME engine	EXCEL[1] EXCEL97[2]	Yes	Yes	Yes	Yes	Microsoft Windows
Excel files using the PC Files LIBNAME engine and the SAS PC Files Server	EXCELCS[5]	Yes	Yes	Yes	Yes	Microsoft Windows, UNIX, (Power Linux Viya 3.5)
Excel .xlsx files	XLSX	No	Yes	No	Yes	Microsoft Windows, UNIX, (Power Linux Viya 3.5)
Excel .xls files	XLS	No	Yes	No	Yes	Microsoft Windows, UNIX
Excel 4 using PROC ACCESS and PROC DBLOAD	EXCEL4	Yes	Yes	Yes	Yes	Microsoft Windows
Excel 5 using PROC ACCESS and PROC DBLOAD	EXCEL5	Yes	Yes	Yes	Yes	Microsoft Windows
Access files using Access LIBNAME engine	ACCESS[3] ACCESS2000[4]	Yes	Yes	Yes	Yes	Microsoft Windows
Access using the PC Files LIBNAME engine and the SAS PC Files Server	ACCESSCS[5]	Yes	Yes	Yes	Yes	Microsoft Windows, (Power Linux Viya 3.5)
dBase .dbf files	DBF	Yes	Yes	Yes	Yes	Microsoft Windows, UNIX, (Power

Data Source		Import		Export		Supported Platforms
Description	DBMS= Identifier	Wizard	PROC	Wizard	PROC	
						Linux Viya 3.5)
dBase files with MEMO support	DBFMEMO	No	Yes	No	Yes	Microsoft Windows, UNIX, (Power Linux Viya 3.5)
JMP .jmp files using JMP Version 7 or later	JMP	Yes	Yes	Yes	Yes	Microsoft Windows, UNIX, (Power Linux Viya 3.5)
Paradox .db files	DB	Yes	Yes	Yes	Yes	Microsoft Windows, UNIX, (Power Linux Viya 3.5)
SPSS .sav files	SAV	Yes	Yes	Yes	Yes	Microsoft Windows, UNIX, (Power Linux Viya 3.5)
Stata .dta files	DTA	Yes	Yes	Yes	Yes	Microsoft Windows, UNIX, (Power Linux Viya 3.5)
Lotus 1-2-3 files .wk1	WK1	Yes	Yes	Yes	Yes	Microsoft Windows
Lotus 1-2-3 files .wk3	WK3	Yes	Yes	Yes	Yes	Microsoft Windows
Lotus 1-2-3 files .wk5	WK4	Yes	Yes	Yes	Yes	Microsoft Windows

Data Source		Import		Export		Supported Platforms
Description	**DBMS= Identifier**	**Wizard**	**PROC**	**Wizard**	**PROC**	
Comma-separated file	CSV	Yes	Yes	Yes	Yes	Microsoft Windows, UNIX, (Power Linux Viya 3.5) Open VMS
Tab-separated file	TAB	Yes	Yes	Yes	Yes	Microsoft Windows, UNIX, (Power Linux Viya 3.5) Open VMS
Delimiter-separated file	DLM	Yes	Yes	Yes	Yes	Microsoft Windows, UNIX, (Power Linux Viya 3.5), Open VMS
File formats .jmp, .sav, and .dta using the SAS PC Files Server	PCFS[5]	No	Yes	No	Yes	Microsoft Windows, UNIX

[1]Data Source Identifier EXCEL has alias names EXCEL2007, EXCEL2010, or later.

[2]Data Source Identifier EXCEL97 has alias names EXCEL2000, EXCEL2002, and EXCEL2003.

[3]Data Source Identifier ACCESS has alias names ACCESS2007, ACCESS2010, or later.

[4]Data Source Identifier ACCESS2000 has alias names ACCESS2002 and ACCESS2003.

[5]SAS PC File Server is not available in SAS Viya 3.4 and earlier, but you can connect to it using a SAS 9 client.

DBMS= specifications ACCESSCS, EXCELCS, and PCFS all require the SAS PC File Server to be installed. The PC File Server is not included in SAS Viya 3.4 and earlier. You can connect to the server through the SAS 9 client. For more information about the PC File Server, see SAS PC Files Server: Installation and Configuration Guide.

When is the PC Server Required?

The IMPORT and EXPORT procedures and Import and Export Wizards support both 64-bit and 32-bit Microsoft Office applications. To use the procedures or the wizards with SAS for Windows, check your version of Microsoft Office to see whether it matches the bitness of your SAS installation.

This table shows whether the PC File Server is required.

Table 2.2 *Is the PC File Server Required?*

If you have SAS running on this operating system	And you have Office 32-bit	And you have Office 64-bit
SAS on Windows 32–bit	No	Yes
SAS on Windows 64–bit	Yes	No
SAS on UNIX	Yes	Yes

For more information about installing the SAS PC Files Server, see *SAS PC Files Server: Installation and Configuration Guide*.

To check your version of Microsoft Office, open one of the products, such as Excel, and follow these steps:

1. On the left side of the application, click **File** ⇨ **Help | Account**. (Whether **Help** or **Account** selection is available depends on your version of Office.)
2. The version of Microsoft Excel is shown under **About Excel** (for example, `Microsoft® Excel® 2013 (15.0.4911.1000) MSO (15.0.4911.1000) 32-bit`).
3. If you click the **About Excel** icon and its **System Info** button, you can also see which version of Microsoft Windows you have (for example, `System Type x64-based PC`).

If you have SAS for Windows x64 and 32–bit Excel, install the SAS PC Files Server.

3

Using SAS Import and Export Wizards

Importing and Exporting Data with the Wizards 15

Using the Wizards and the PC File Server 16
Overview 16
Using the Wizards when the Bitness of SAS on Windows and Microsoft Office Match 16
Handling User-Defined Formats 17
Using the Export Wizard with 64–Bit SAS on Windows and 32-Bit Microsoft Office 17
Using the Import Wizard with the 64–Bit SAS on Windows and 32-Bit Microsoft Office 21

Using the External File Interface (EFI) 29

Importing and Exporting Data with the Wizards

The Import and Export wizards enable you to transfer data between external data sources and SAS data sets. Each wizard presents a series of pages with simple choices to guide you through the import or export process.

Because there is no coding required, using the wizards makes it easy to complete a complex or infrequently performed task. The wizards generate DATA step code that you can save to a file for subsequent use.

Restriction: The Import and Export wizards and the External File Interface are not supported in SAS Viya or with CAS tables. They must be used in the DMS environment, which is available in the SAS 9 client.

Using the Wizards and the PC File Server

Overview

The Import and Export wizards support both 64-bit and 32-bit Microsoft Office applications. To use the wizards on SAS for Windows, check your version of Microsoft Office to see whether it matches the bitness of your SAS installation.

This table shows whether the PC File Server is required.

If you have SAS running on this version of Windows	And you have Office 32-bit	And you have Office 64-bit
SAS on Windows 32–bit	No	Yes
SAS on Windows 64–bit	Yes	No
SAS on UNIX	Yes	Yes

For information about when the PC Files Server is required, see "When is the PC Server Required?" on page 14.

If the bitness of SAS for Windows and Microsoft Excel do not match, install the SAS PC Files Server and proceed to the following sections:

- "Using the Export Wizard with 64–Bit SAS on Windows and 32-Bit Microsoft Office" on page 17
- "Using the Import Wizard with the 64–Bit SAS on Windows and 32-Bit Microsoft Office" on page 21

If the bitness of SAS for Windows and Microsoft Excel match you do not need the SAS PC Files Server.

For more information on installing the SAS PC Files Server, see *SAS PC Files Server: Installation and Configuration Guide*.

Using the Wizards when the Bitness of SAS on Windows and Microsoft Office Match

The Import wizard guides you through the steps to read data from an external data source and write it to a SAS data set. If the bitness of SAS for Windows and

Microsoft Office match, you do not need the SAS PC File Server. Start the Import wizard by selecting **File** ⇨ **Imported Data**. Follow the instructions on the screens.

The Export wizard guides you through the steps to read data from a SAS data set and write it to an external data source. If the bitness of SAS for Windows and Microsoft Office match, you do not need the SAS PC File Server. Start the Export wizard by selecting **File** ⇨ **Export Data**. Follow the instructions on the screens.

For more information on matching bitness, see "When is the PC Server Required?" on page 14.

Handling User-Defined Formats

If you indicate on the source selection page that the source data type (for import or for export) is a user-defined format, the wizard opens the External File Interface (EFI) to enable you to specify the format of your data. For more information, see "Using the External File Interface (EFI) " on page 29.

Using the Export Wizard with 64–Bit SAS on Windows and 32-Bit Microsoft Office

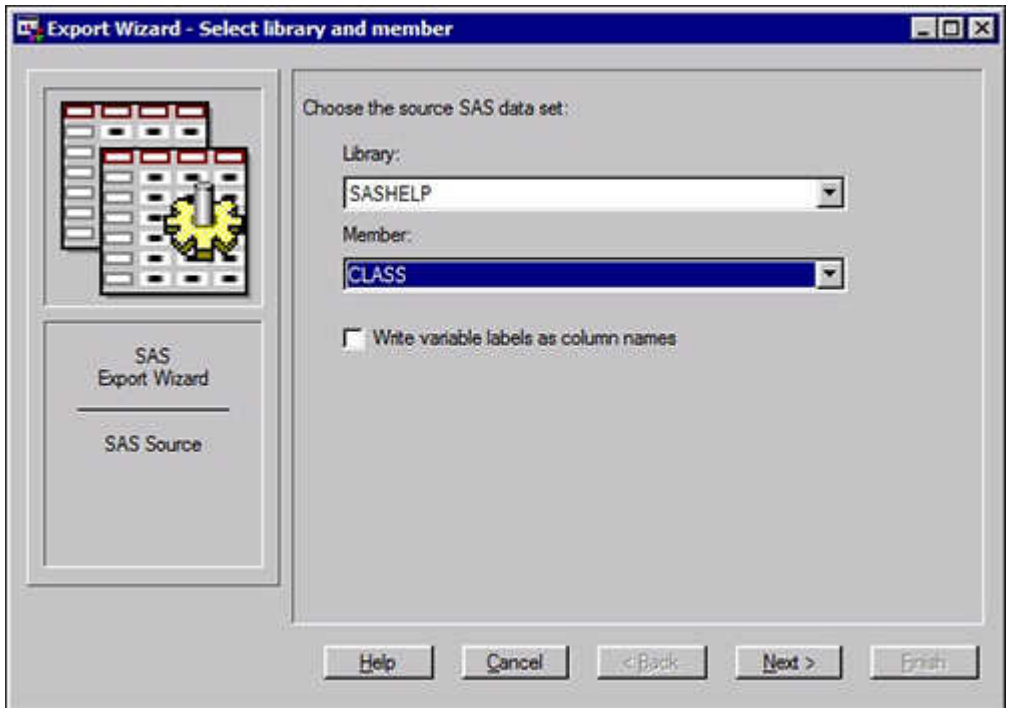
To use the Export wizard on Windows X64 systems with 32-bit Microsoft Office, follow these steps.

Click **File** ⇨ **Export Data**.

The Export wizard opens the Select library and member page.

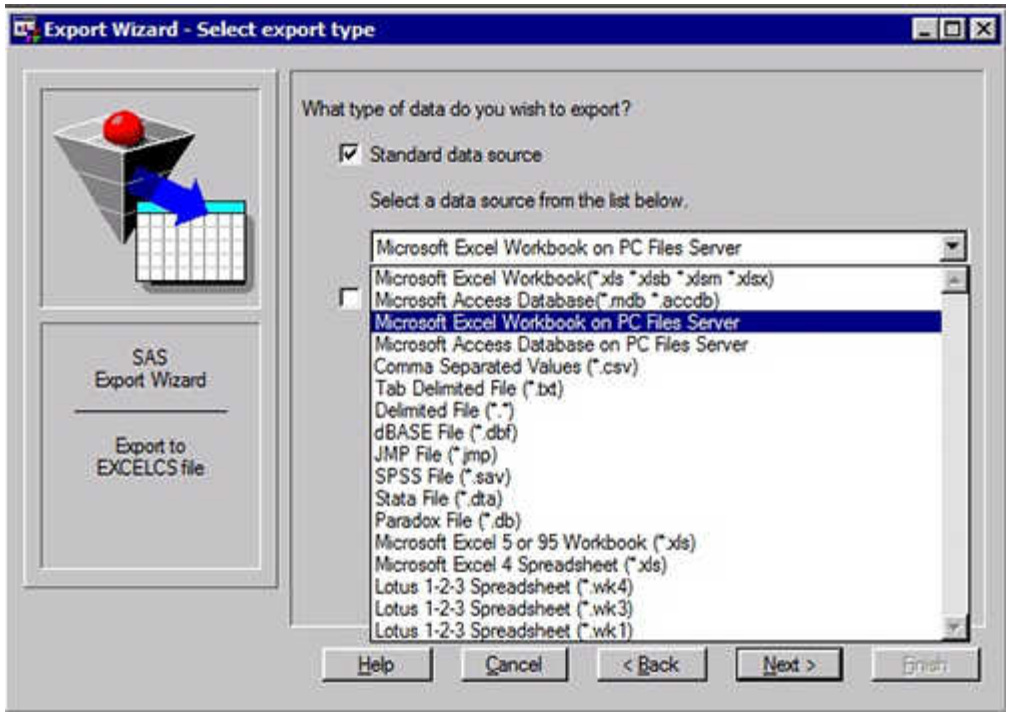
Select the libref for your **SAS Library** and then select the **Member** of the library (always a SAS data set). Always select the libref first because the SAS data set is associated with the libref.

Figure 3.1 Select Library and Member Page



The Select export type page lists possible file formats.

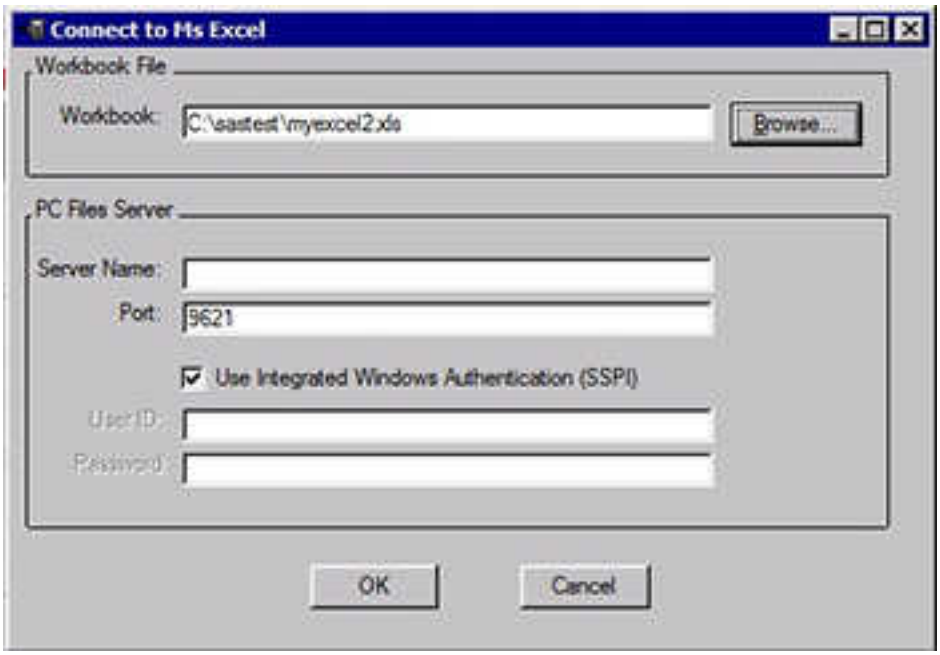
Figure 3.2 Listing of Possible File Format Types



- Do not select **`Microsoft Excel Workbook`**. This is for 64-bit Office, not 32-bit.
- Select **`Microsoft Excel Workbook on PC Files Server`**.

The Connect to MS Excel page enables you to specify the location for your Excel file.

Figure 3.3 *Location for Your Exported Excel File*



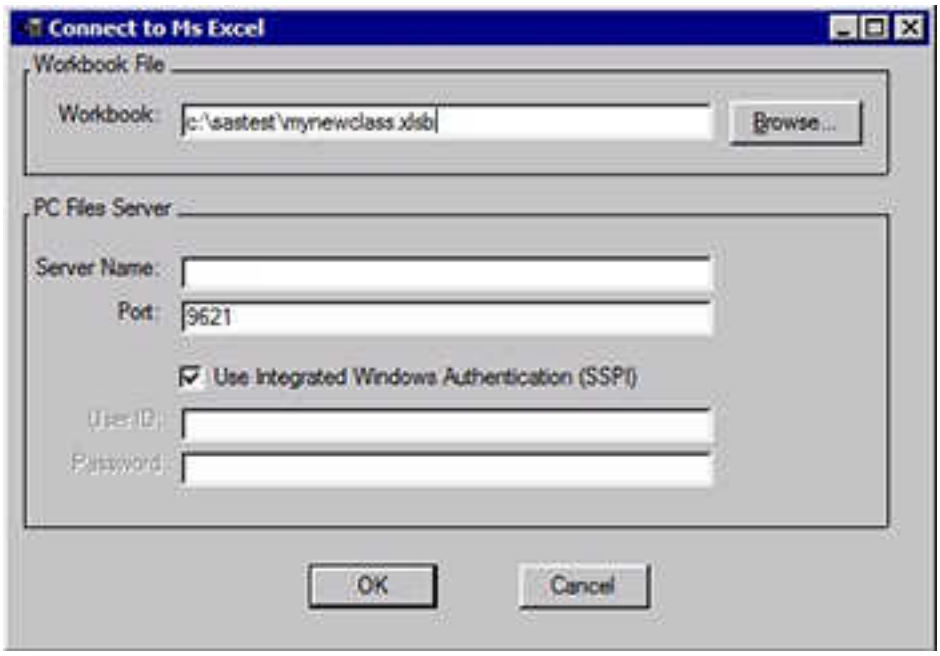
Your entry for the **Workbook** location depends on whether the Excel file already exists.

- If the Excel file exists, you can navigate to its location using the Browse dialog box. The Excel file can be in any of these formats: .xls, .xlsx, or .xlsb.
- If the Excel file does not exist, enter the full directory path including the Excel file name that you want to create. Do not use the **Browse** button. However, you can copy the directory path using Windows Explorer.

> **Note:** Any new Excel file must have an .xls, .xlsx or .xlsb extension on it. It cannot have any other extension type as required by the Microsoft Excel ODBC driver when creating new Excel files. New .xlsx files will have a "_SAS_empty_" worksheet in addition to the worksheet containing the data that was added by SAS. This is due to a limitation of the Microsoft Excel ODBC driver.

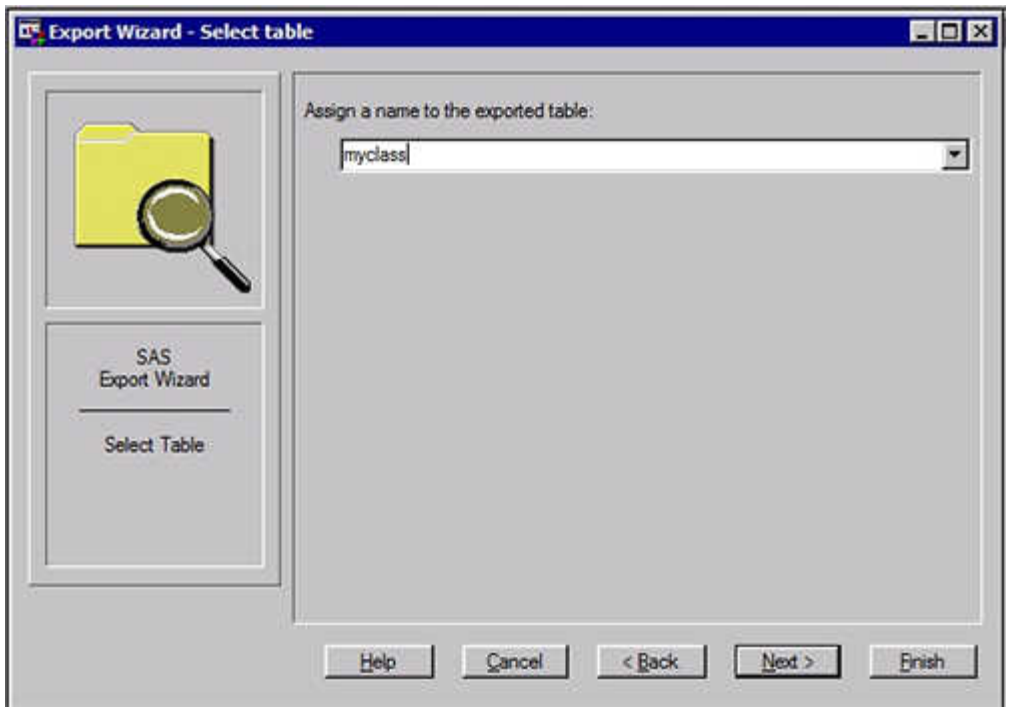
In this example, you specify a new .xlsb file. (The **Port** listed is the default value for SAS/ACCESS 9.4 to PC Files and for the SAS PC Files Server.)

Figure 3.4 Display Shows a New .xlsb File



The Select Table page enables you to enter the name of the worksheet or table that you want to add to your Excel file. If the worksheet already exists, you can use the down arrow to select the sheet. If the sheet or the file does not exist, enter the name of the worksheet.

Figure 3.5 Select Table Page

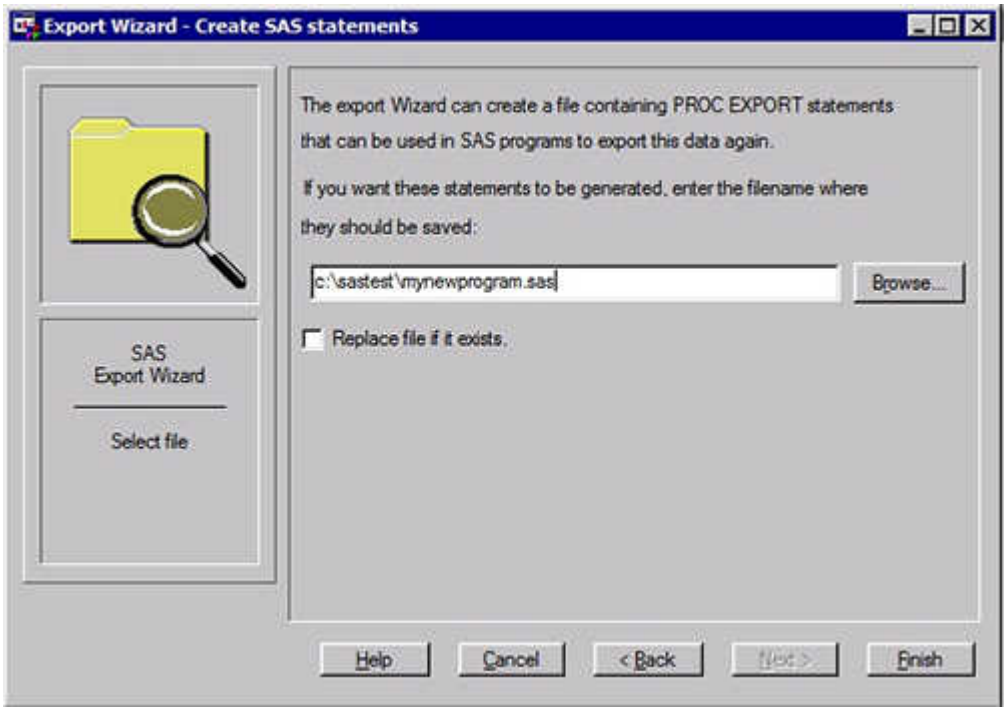


Click either **Next** or **Finish**. If you click **Next**, the wizard creates a SAS program for you, generating the EXPORT procedure code. If you click **Finish**, you skip the next page and you are done. This example proceeds to the next page.

The Create SAS Statements page enables you to store the PROC EXPORT program that the wizard creates. Enter the name of the SAS program, including its full directory path. SAS programs use an `.sas` extension, as shown on the page.

If you want to replace an existing PROC EXPORT program, you can click the check box to replace it and navigate to it. You are prompted to verify that you want to replace it.

Figure 3.6 *Create SAS Statements Page*



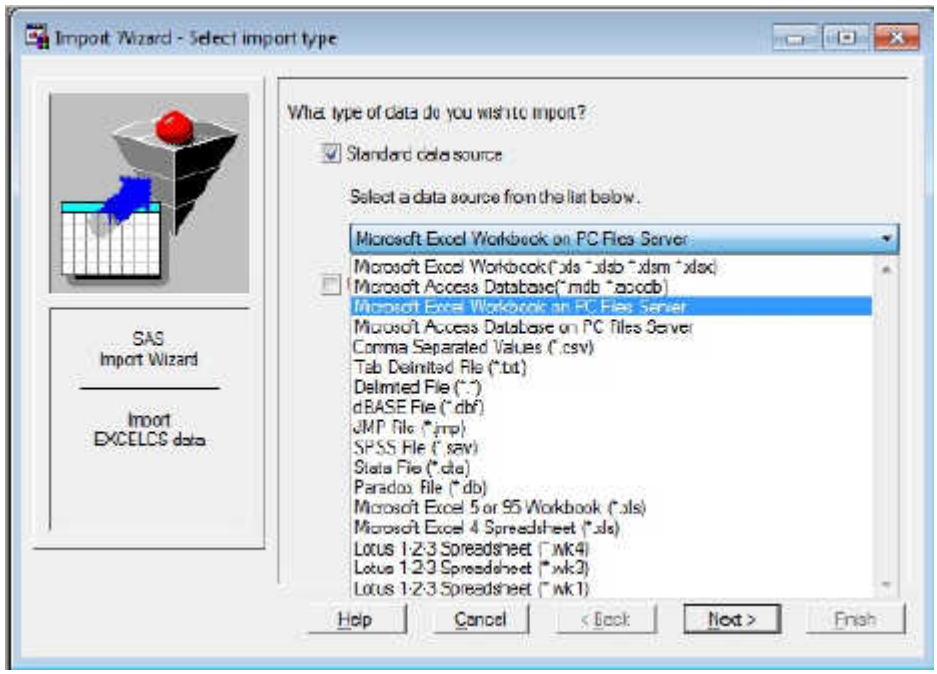
Click **Finish** to complete the process.

Using the Import Wizard with the 64–Bit SAS on Windows and 32-Bit Microsoft Office

To use the Import wizard, click **File** ⇨ **Import Data**.

The Import wizard opens to a list of possible file format types.

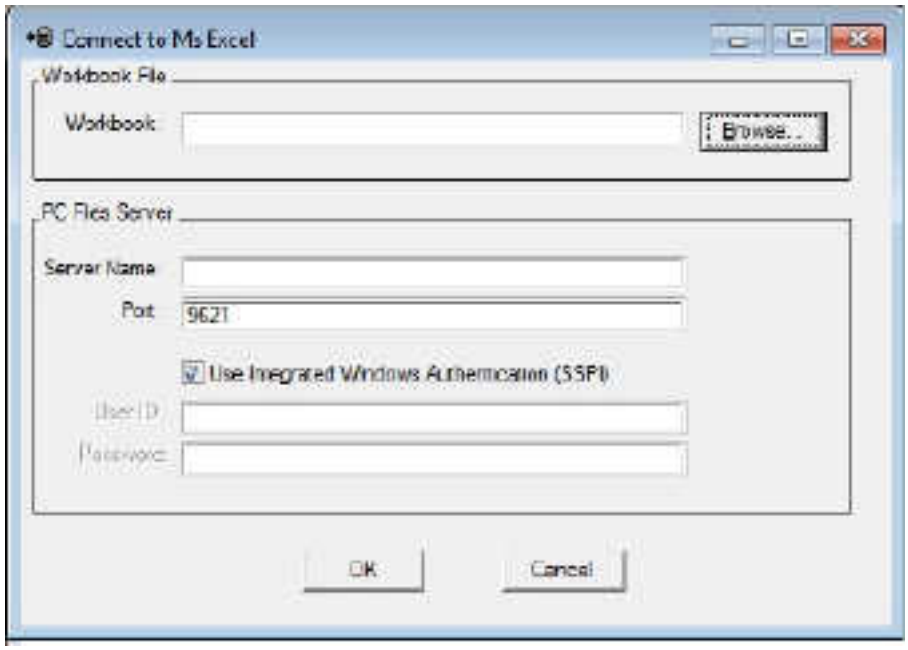
Figure 3.7 *Import Wizard - Select Import Type Page*



- Do not select **`Microsoft Excel Workbook`**. This is for 64-bit Office, not 32–bit Office.
- Select **`Microsoft Excel Workbook on PC Files Server`**.

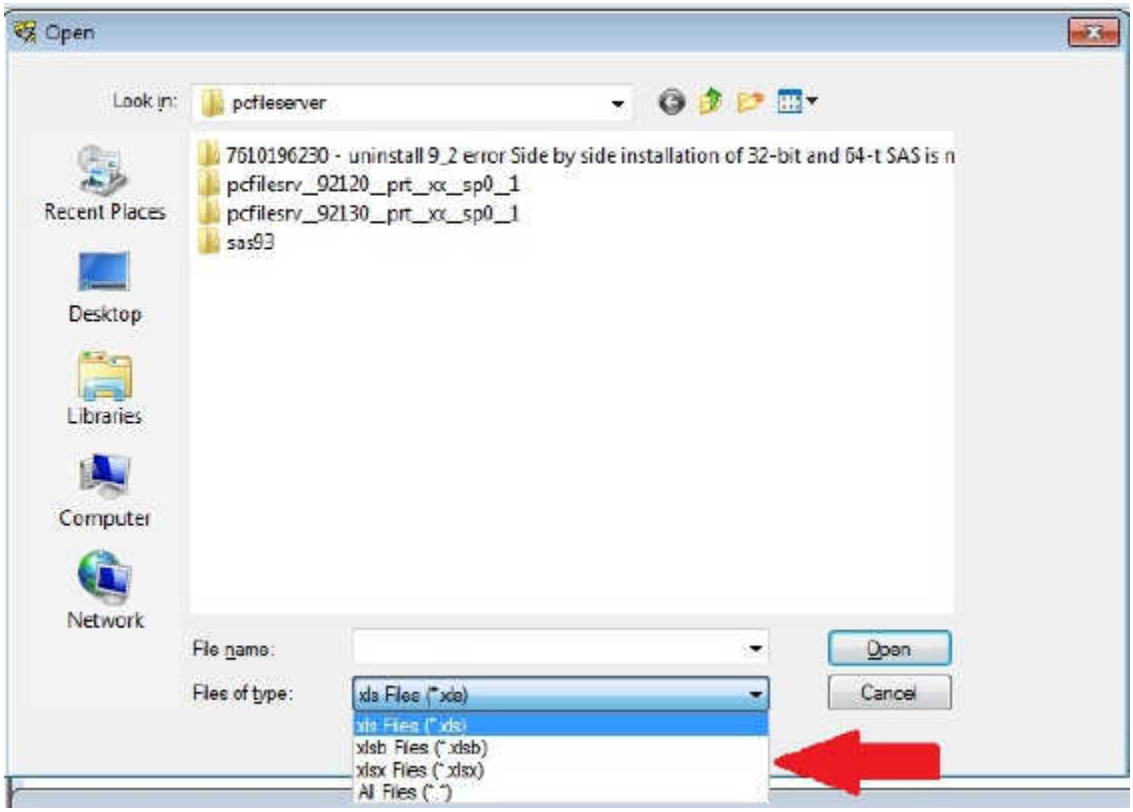
The Connect to MS Excel page appears so that you can identify the Excel file's location. (The **Port** listed is the default value for SAS/ACCESS 9.4 to PC Files and for the SAS PC Files Server.)

Figure 3.8 *Import Wizard - Connect to MS Excel*



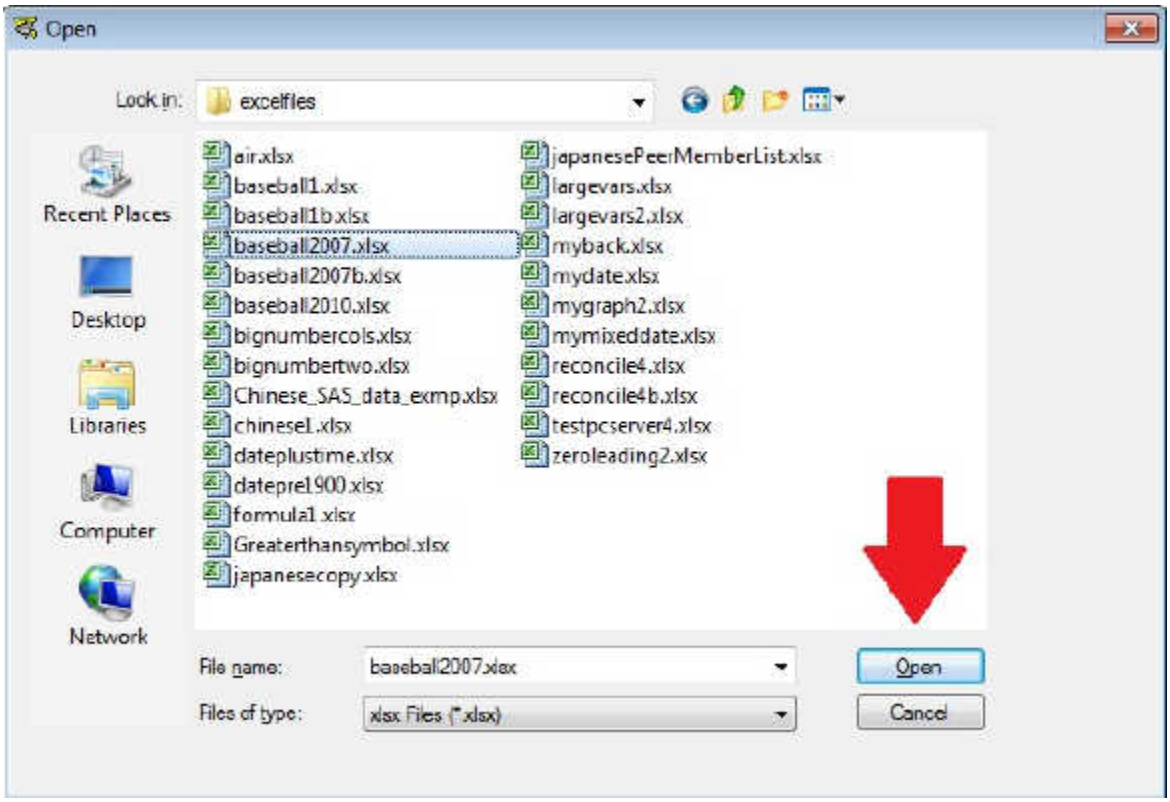
Click **Browse** to open the **Files of Type** drop-down list. Select the file extension of your Excel file. If you do not know the extension, select **`All Files`**:

Figure 3.9 *Dialog Box That Shows "File of types" Drop-down List*



Navigate to the location of your Excel file and click **Open**.

Figure 3.10 Dialog Box that Shows .xlsx File Selected to Open



The Select table page gives you a drop-down list of worksheets or tables to read. From the drop-down list, select the table that you want to read (that is, import into SAS).

Figure 3.11 Page to Select the Table to Import

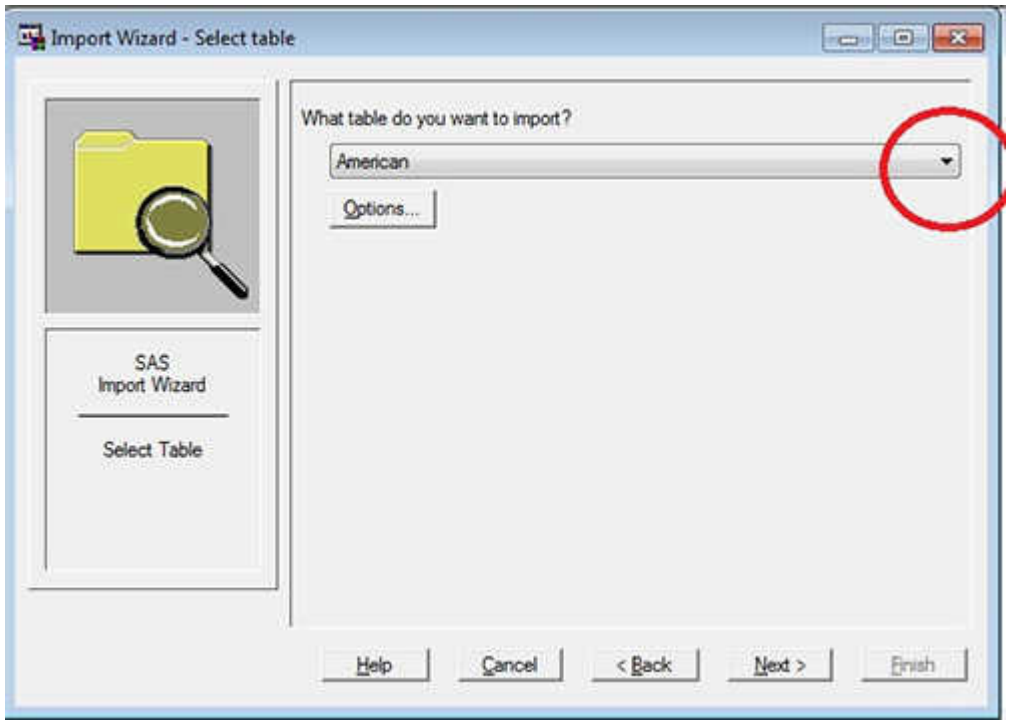
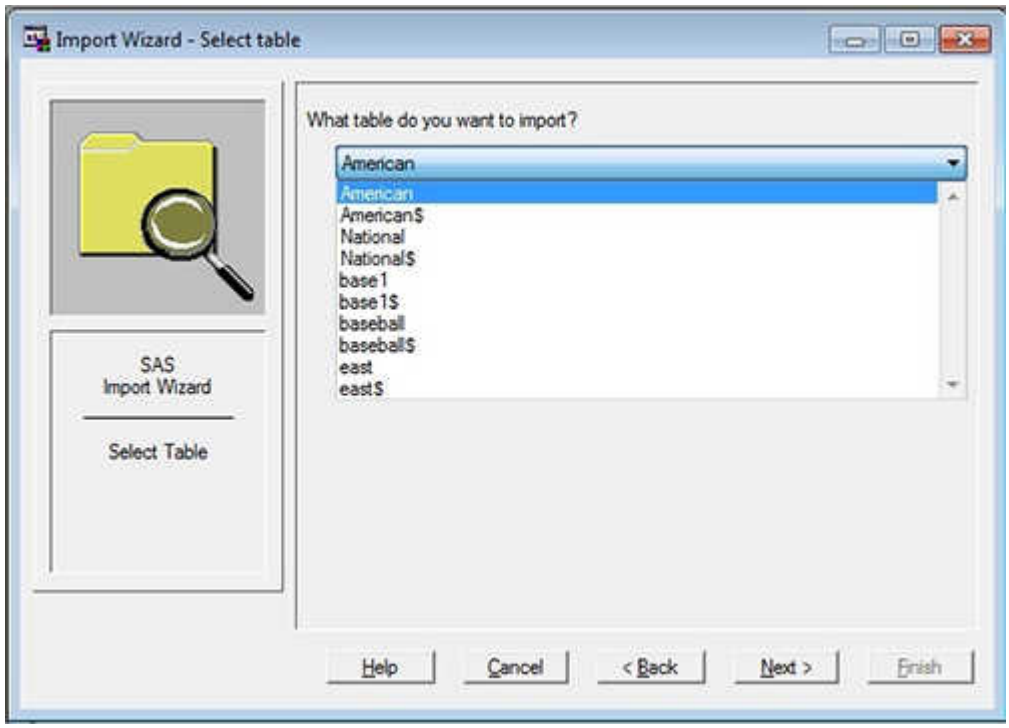


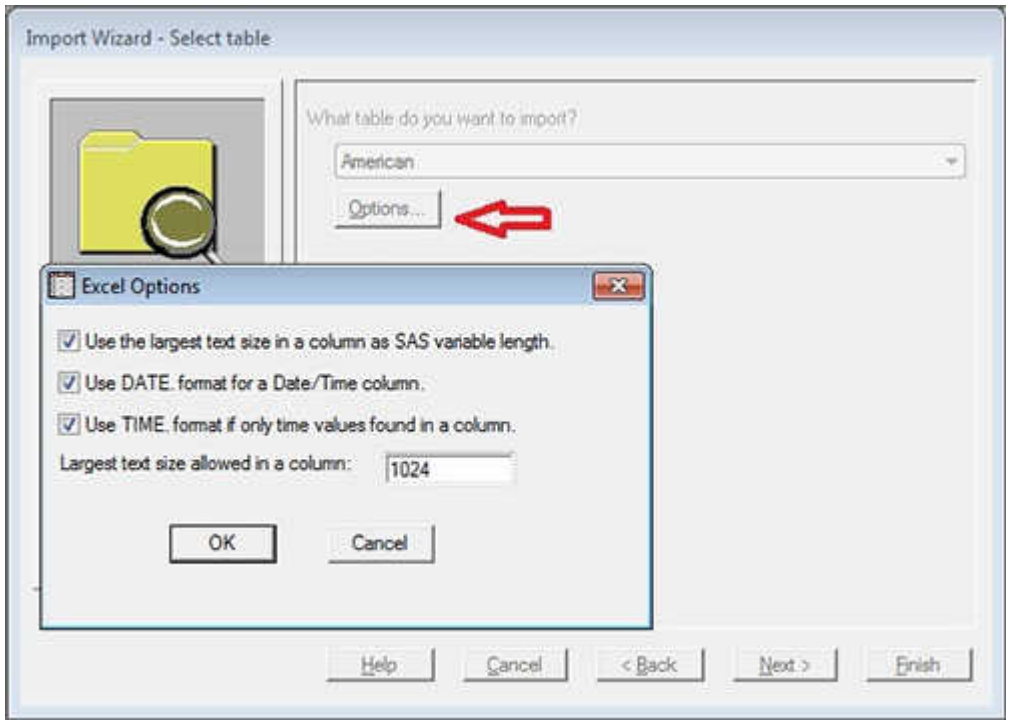
Figure 3.12 *Drop-down List of Tables from Which to Select*



Click the **Options** button to open the **Excel Options** dialog box. These options are set by default.

- If you have DATETIME data in a column of your worksheet (such as `01jan2013:08:13:22`), deselect the check box for **Use DATE.format for a Date/Time column**. Otherwise, you get only the date.
- You can adjust the `1024` setting to capture column text lengths in your worksheet that are longer than 1024 characters. The maximum value is 32,767 characters.

Figure 3.13 Excel Options for the Table Selected



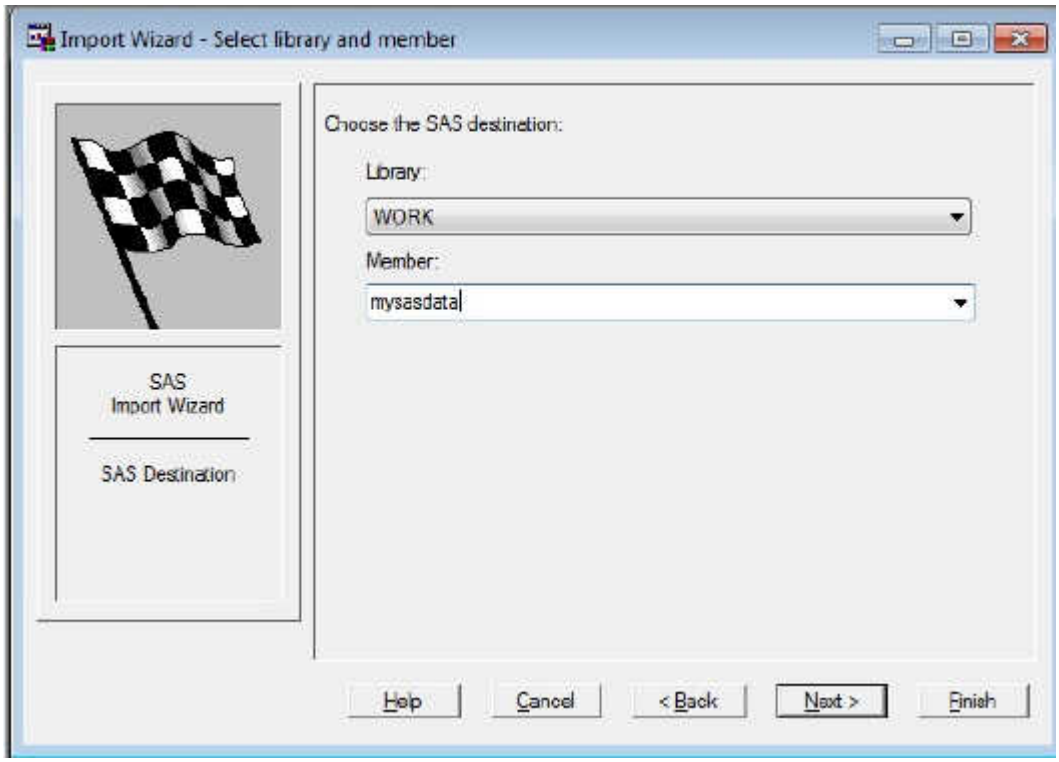
Click **OK** to close the **Excel Options** and click **Next** from the Select Table page.

In the Select library and member page, select the location to store the Excel worksheet in a SAS data set. Choose the SAS libref in which you want to store the SAS data set.

- If you want the data set to be removed after you exit SAS, select the default WORK library.
- If you want the data set to remain after you exit SAS, select another SAS library, such as SASUSER.

If the SAS data set does not exist, enter a name in the **Member** field to create the data set. If the SAS data set already exists, choose the member to replace it, and click **Next**.

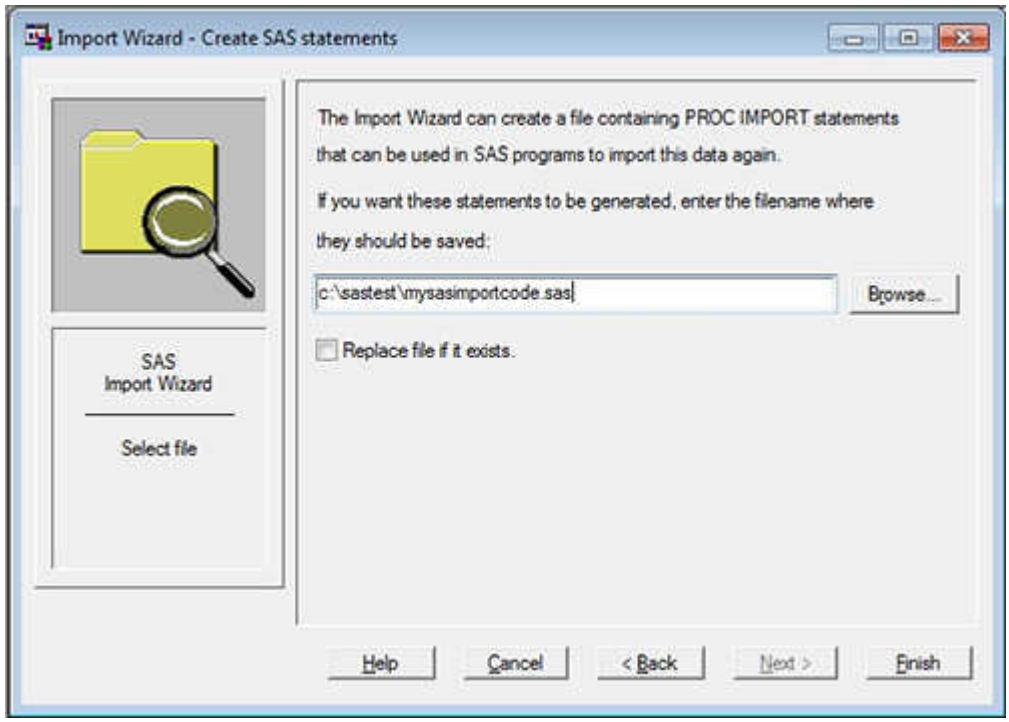
Figure 3.14 *SAS Libref and Data Set Name for Storing the Excel File*



The Create SAS statements page enables you to store the PROC IMPORT code that the wizard generates. The code is saved as a SAS program that you can use without opening the Import wizard.

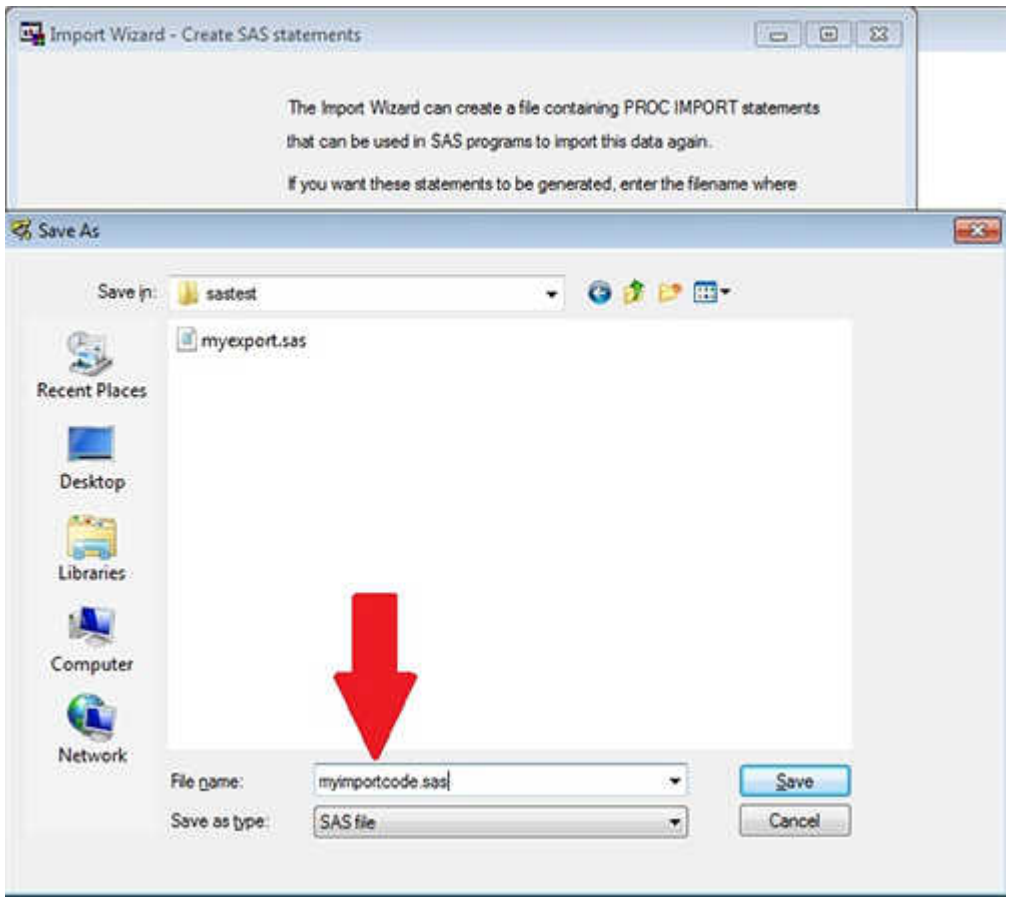
You can either create a new SAS program or replace an existing SAS program. To create a new SAS program, enter its name including its full directory path; SAS programs use the extension `.sas`, as shown in the figure.

Figure 3.15 *Name of the SAS Program Generated by the Import Wizard*



To replace an existing PROC IMPORT program, you can click the check box to replace it and then navigate to the program (using the **Browse** button). You are prompted to verify that you want to replace it.

Figure 3.16 *Drop-down List of SAS Programs That You Could Select from to Replace with This Program*



Click **Finish** to complete the process.

The SAS log contains a message similar to the following:

```
NOTE: WORK.MYSASDATA data set was successfully created.
NOTE: The data set WORK.MYSASDATA has 175 observations and 22 variables.
```

Using the External File Interface (EFI)

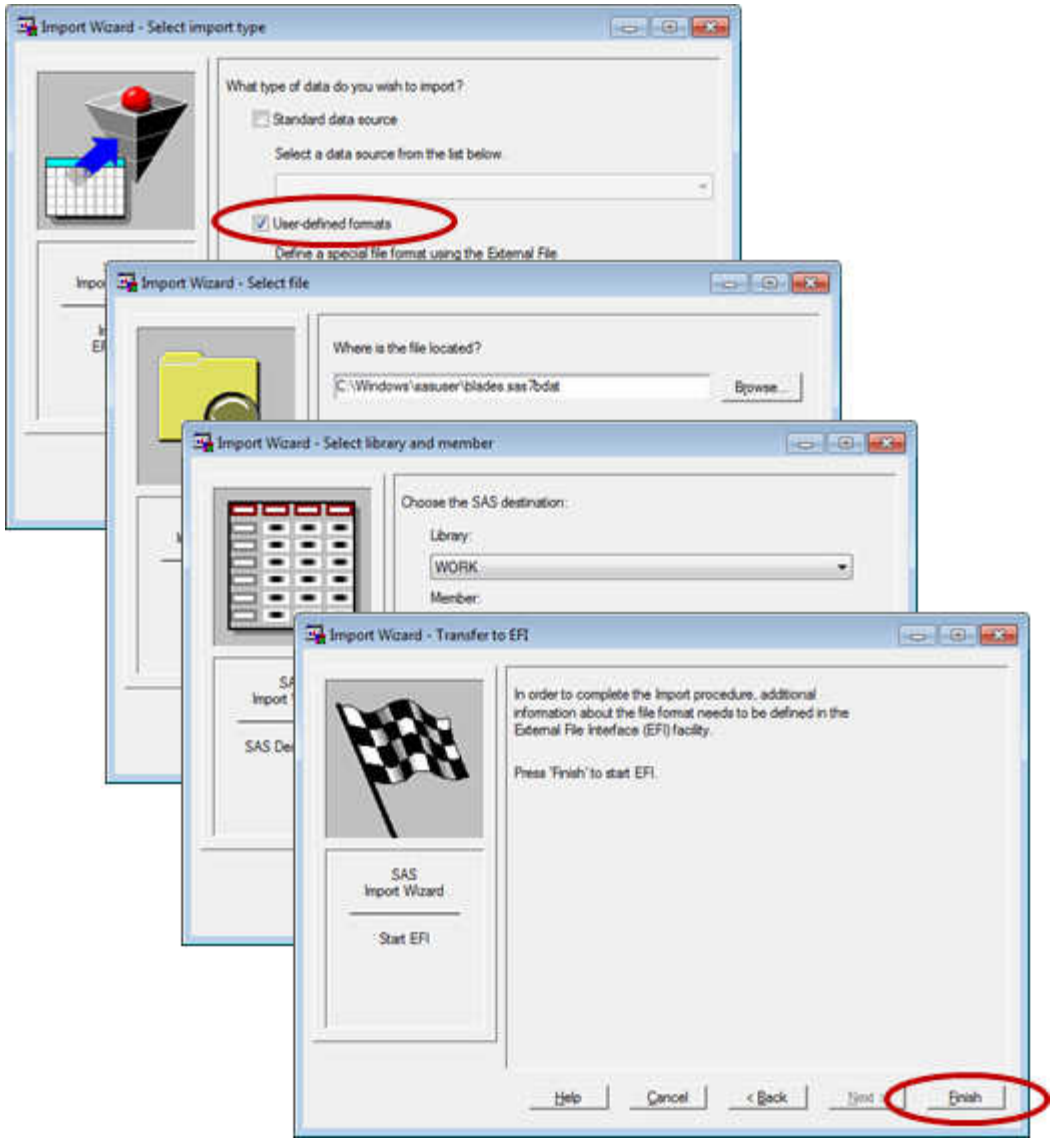
The External File Interface (EFI) is a graphical user interface that enables you to read and write data that is not in the SAS format. That is, the source data is in a user-defined format.

Start the Import wizard or Export wizard. When you get to the data source selection page, you can select the **User-defined format** check box. After three more pages, the Import or Export Wizard prompts you to select **Finish** in order to invoke the EFI.

Note: EFI does not recognize non-ASCII characters when SAS Registry contains "UseDateStyle"="Yes".

Note: The EFI is not supported on SAS Viya. Use the SAS 9 client, which supports SAS Display Manager. SAS Display Manager is required to use the EFI.

CAUTION
Sequential access is not allowed when you use EFI.



Note: To return to the Import or Export wizard without invoking the EFI, click **Back**. To close the Import or Export wizard without invoking the EFI, click **Cancel**. Assistance in using EFI is available from the **Help** button.

4

IMPORT Procedure

Overview: IMPORT Procedure *31*
Overview: PROC IMPORT 31
Syntax: IMPORT Procedure *32*
PROC IMPORT Statement 32

Overview: IMPORT Procedure

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

In addition, JMP, Excel (.xlsx and .xls), SPSS, and Stata files are supported by the CAS engine for importing to a CAS table in memory on the CAS server and

exporting CAS tables to the PC file format. A connection to the CAS server must be made in order to use the CAS engine to import or export CAS data. When importing into CAS tables, all CHAR data types or strings in the PC file formats are converted to the VARCHAR data type in a CAS table. For more information, see "Support for the VARCHAR Data Type" in *Base SAS Procedures Guide*. For more information about the CAS engine and the CAS LIBNAME statement, see *SAS Cloud Analytic Services: User's Guide*.

The SAS column (or variable) definitions are based on the input rows (or records). PROC IMPORT imports the data using one of the following methods:

- generated DATA step code
- generated SAS/ACCESS code

You can customize the results with statements and options that are specific to the input data source. PROC IMPORT generates a SAS data set or CAS table and writes information about the import to the SAS log. The DATA step or the SAS/ACCESS code that PROC IMPORT generates is written to the log.

When you create delimited files on one host operating environment and then transfer them to another host, the end-of-line delimiters in the files might not match. To remedy the situation, see "Sharing Delimited Files across Hosts" on page 56.

To import data, you can also use the Import Wizard or the External File Interface (EFI) to guide you through the steps to import an external data source. You can use the Import Wizard to generate PROC IMPORT statements, which you can save to a file for subsequent use.

To open the Import Wizard, select **File** ⇨ **Import Data**. For more information, see Chapter 3, "Using SAS Import and Export Wizards," on page 15.

Note: The Import and Export wizards are not supported in SAS Viya or with CAS tables. You can read data into a SAS data set using the SAS 9 client.

Syntax: IMPORT Procedure

PROC IMPORT

DATAFILE="*file-name*" | DATATABLE="*table-name*"

<*options*>;

PROC IMPORT Statement

The IMPORT procedure reads external data and writes the data to a SAS data set or CAS table if the CAS LIBNAME engine is used. You must have a connection to the CAS server.

Restriction: The path name (which includes the file name) must be a total of 201 characters or less.

Notes: SAS Viya supports only the UTF-8 encoding and runs only on Linux.

Invalid UTF-8 characters are replaced with question marks so that Excel can open the file.

Tip: Beginning with SAS Viya 3.5, PROC IMPORT supports all access types that are available in the FILENAME statement.

See: Chapter 6, "File Format-Specific Reference for the IMPORT and EXPORT Procedures"

Syntax

PROC IMPORT

DATAFILE="*file-name*" | DATATABLE="*table-name*"

< DBMS=*data-source-identifier* >

<OUT=*libref.SAS-data-set-name* <*SAS-data-set-options*>>

< REPLACE >

< *file-format-specific-statements* >;

Required Arguments

DATAFILE="*file-name*" | "*fileref*"

specifies the complete path and file name 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 *file name* does not include certain special characters, such as a backslash or spaces.

Alias: FILE

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

The IMPORT procedure can import data, and in most cases, the data is converted to a supported SAS or CAS data type. SAS supports NUMERIC (double) and CHARACTER data types. The CAS engine supports NUMERIC, CHAR, and VARCHAR data types. When a CHAR or string value in a PC file is imported to a CAS table, the CAS engine converts the CHAR to VARCHAR in the CAS table. In some cases, If the data that you want to import is a type that SAS does not support, the IMPORT procedure might not import it correctly.

The path name (which includes the file name) must be a total of 201 characters or less.

Interactions By default, the IMPORT procedure reads delimited files as varying record-length files. If your external file has a fixed-length format, use a SAS DATA step with an INFILE statement that includes the RECFM=F and LRECL= options. For more information, see the INFILE statement.

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 the CHAR data type and any subsequent numeric data for that column is set to missing. (You can change the default from 8 to 0 in the Windows registry; 0 causes all the rows in the column to be scanned to determine the type.) The CAS engine defaults CHAR to VARCHAR for CAS tables in memory on the CAS server. Other file formats might have other default values.

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 DATA Step 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. DATATABLE= is generally used for Microsoft Access database tables, but not for Microsoft Excel sheets.

Alias TABLE

Requirement When importing Microsoft Access database tables,SAS/ACCESS converts the table name to a SAS member name. Note that SAS does not support member names longer than 32 bytes. When you import a DBMS table, you must specify the DBMS= identifier option.

Optional Arguments

SAS-data-set-options
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 4.3 on page 35. For example, DBMS=ACCESS specifies to import a Microsoft Access 2000, 2002, 2003, 2007, 2010, and later database. This table shows the available DBMS= identifiers. All DBMS= specifications refer to local access, except where noted in this table.

Table 4.1 *DBMS Specifications*

DBMS= Identifier	Output Data Source	File Extension
ACCESS	Microsoft Access 2000, 2002, 2003, 2007, 2010, and later table.	.mdb .accdb
ACCESSCS	Microsoft Access table connecting remotely through SAS 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) **Note:** the delimiter must be specified using the DELIMITER= statement.	.*
DTA	Stata file	.dta
EXCEL	Microsoft Excel 97, 2000, 2002, 2003, 2007, 2010, and later workbooks.	.xls .xlsb .xlsm .xlsx
EXCEL4 EXCEL5	Microsoft Excel 4.0, Excel 5.0 or 7.0 (95) workbook, using PROC ACCESS.	.xls
EXCELCS	Microsoft Excel workbook connecting remotely through the SAS PC Files Server.	.xls,.xlsx, .xlsb, xlsm
JMP	JMP files in Version 7 and later format.	.jmp
PARADOX	Paradox DB files	.db

DBMS= Identifier	Output Data Source	File Extension
PCFS	JMP files, SPSS files, and Stata files connecting remotely through the SAS PC Files Server. SAS can be running on UNIX or another Windows system. **Note:** Microsoft .xls and .xlsx files are not supported. Use DBMS=EXCELCS.	.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 **Note:** Transcoding is not supported for DBMS=XLS. The output yields unpredictable results. Use DBMS=EXCEL or (DBMS=EXCELCS with the SAS PC Files Server) as an alternative. Or, if your file has more than 255 columns, save the .xls file as .xlsx to support transcoding. If you use UNIX use DBMS=XLSX for transcoding.	.xls
XLSX	Microsoft Excel 2007 and later workbook using file formats	.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 the SAS PC Files Server on Microsoft Windows.

Note: The SAS PC File Server is not included in SAS Viya 3.4 and earlier. You can connect to the server using the SAS 9 client.

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 SAS 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;
```

Note: When you specify DBMS=XLS with PROC IMPORT, if there are two or more characters that are invalid in the regular naming convention, then the code switches the variable names to **Var#**.

When you specify DBMS=EXCEL, the IMPORT procedure can read any files that are saved in EXCEL97, EXCEL2000, EXCEL2002, EXCEL2003, EXCEL2007, EXCEL2010, and later versions. If you specify DBMS=XLSX, the IMPORT procedure can read only EXCEL2007 or later versions of the files that are saved in Microsoft Excel workbooks. DBMS=EXCEL uses the Excel LIBNAME engine transparently.

When you specify DBMS=EXCELCS, the IMPORT procedure uses the PC Files LIBNAME engine transparently to import Excel files on a remote Windows PC. The PC must be running the SAS PC Files Server. You can connect from UNIX or another Windows PC.

This table lists the versions of Microsoft Excel files that are supported by each DBMS= identifier.

Table 4.2 *Microsoft Excel Workbook Specifications*

DBMS= Identifier	Excel 2007 and later	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
EXCELCS	Yes	Yes	Yes	Yes
EXCEL5	No	No	Yes	Yes
EXCEL4	No	No	Yes	Yes

This table lists the Excel-specific DBMS= identifier, the software that it uses, any required software, and the operating platform that can use the DBMS= specification.

Table 4.3 *DBMS Specifications for Excel*

DBMS= Identifier	Uses	Requires	Operating Platform
EXCEL	Excel LIBNAME engine	Microsoft ACE or Jet Provider	Microsoft Windows
XLS	File formats technology		Microsoft Windows, UNIX
XLSX	File formats technology		Microsoft Windows, UNIX
EXCELCS	PC Files LIBNAME engine	SAS PC Files Server	Microsoft Windows, UNIX

This table is a quick reference for which DBMS= data source identifier to use for Microsoft Excel files, based on the location of your data.

Table 4.4 *DBMS Data Source Identifiers for Microsoft Excel and Access Files*

DBMS= Identifier	Microsoft Windows	UNIX	Linux
ACCESS	yes	no	no
ACCESSCS	yes (source file on remote PC File Server on Windows)	yes (source file on remote PC File Server on Windows, target data set on in SAS on UNIX).	yes (source file on remote PC File Server Windows, target data set in SAS on Linux).
EXCEL	yes	no	no
XLS	yes	yes	yes
XLSX	yes	yes	yes
EXCELCS	yes (source file on remote PC File Server on Windows)	yes (source file on remote PC File Server on Windows, target	yes (source file on remote PC File Server on Windows, target data set on SAS on Linux).

DBMS= Identifier	Microsoft Windows	UNIX	Linux
		data set on SAS on UNIX).	

Note Transcoding and multi-byte characters are not supported for DBMS=XLS. The output yields unpredictable results. If you use Microsoft Windows and your file has 255 columns or fewer, use DBMS=EXCEL for transcoding. Or save the .xls file as .xlsx. If you use UNIX, use DBMS=XLSX for transcoding.

PCFS

Specify DBMS=PCFS to connect to the SAS PC Files Server to import JMP, SPSS, and Stata files that are on a remote Windows PC. DBMS=PCFS enables you to access data on Microsoft Windows from UNIX or another Microsoft Windows operating environment. The remote Windows PC must be running the SAS PC Files Server. The SAS PC Files Server is not included with SAS Viya 3.4 and earlier. You can connect to the server through the SAS 9 client.

This option does not support Microsoft Excel workbook files.

For more information, see "Importing and Exporting SAS JMP Files Data" on page 121, "Importing and Exporting SPSS Files" on page 129, and "Stata DTA Files" on page 134.

ACCESS and ACCESSCS

Microsoft Access versions 2000, 2002, and 2003 share the same internal file formats. When you specify DBMS=ACCESS, the Access LIBNAME engine is used transparently. The Access LIBNAME engine recognizes ACCESS2000, ACCESS2002, ACCESS2003, ACCESS2007, ACCESS2010, and later versions 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.

DBMS=ACCESSCS uses the PC Files LIBNAME engine connecting through the PC Files Server.

When the DATABASE= connection option is specified for an Access database .mdb file that does not exist, a database is created using the format specified with the DBMS= identifier. If you specify DBMS=ACCESS to create a file, the result is an .mdb file that Access 2000–2013 can read. Access 97 cannot read this file.

For more information about the DATABASE= connection option, see "Microsoft Access Database Files" on page 93.

CAUTION
Only Access 2007 and later can open .accdb files.

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.

See Chapter 2, "Supported Data Sources and Environments," on page 9

OUT=*<libref.>SAS data-set*
identifies the output SAS data set with either a one- or two-level SAS name (library and data set names). 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 it uses the WORK library if SASUSER not assigned.

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

CAUTION
Using the IMPORT procedure with the REPLACE option to replace an existing SAS generation data set causes the most recent (base) generation data set or group of generation data sets to be deleted.

If you write to an existing generation data set using the IMPORT procedure with the REPLACE option and you

- specify the GENMAX= data set option to increase or decrease the number of generations, all existing generations are deleted and replaced with a single new base generation data set.
- omit the GENMAX= data set option, all existing generations are deleted and replaced with a single new data set by the same name. This resulting data set is not a generation data set.

Instead, use a SAS DATA step with the REPLACE= data set option to replace a permanent SAS data set and to maintain the generation group for that SAS data set.

<file-format-specific-statements>
See Chapter 6, "File Format-Specific Reference for the IMPORT and EXPORT Procedures" for the supported syntax for your data source.

5

EXPORT Procedure

Overview: EXPORT Procedure *41*
Overview: PROC EXPORT 41
Syntax: EXPORT Procedure *42*
PROC EXPORT Statement 42

Overview: EXPORT Procedure

Overview: PROC EXPORT

The EXPORT procedure reads data from a SAS data set or a CAS table that is in memory on the CAS server, and writes it to an external data source. Here is what 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

In addition, JMP, Excel (.xlsx and .xls), SPSS, and Stata files are supported by the CAS engine for exporting a CAS table in memory on the CAS server to the PC file format. A connection to the CAS server must be made in order to use the CAS engine to import or export CAS data. When exporting CAS tables, all VARCHAR data types are converted to the CHAR data type in the PC file format. For more information about the CAS engine and the CAS LIBNAME statement, see *SAS Cloud Analytic Services: User's Guide*.

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

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.

You can also use the Export Wizard or the External File Interface (EFI) to guide you through the steps to export a SAS data set. The Export Wizard can generate EXPORT procedure statements, which you can save to a file for subsequent use.

To open the Export Wizard from the SAS windowing environment, select **File** ⇨ **Export Data**. For more information and an example, see Chapter 3, "Using SAS Import and Export Wizards," on page 15.

Note: The Import and Export wizards are not supported in SAS Viya or with CAS tables. You can use them from the SAS 9 client.

Syntax: EXPORT Procedure

PROC EXPORT

DATA=libref.SAS-data-set | caslib.table-name

OUTFILE="*file-name*" | OUTTABLE="*table-name*"

< *options* >;

PROC EXPORT Statement

The EXPORT procedure reads a SAS data set (or CAS table if you have a connection to CAS) and writes the data to an external data file.

See: "EXPORT Procedure" in *Base SAS Procedures Guide*

Syntax

PROC EXPORT

DATA=<*libref.*>*SAS-data-set* | *caslib.table-name* < (*SAS-data-set-options*) >

OUTFILE="*file-name*" | OUTTABLE="*table-name*"

< DBMS=*data-source-identifier* >

< LABEL > < REPLACE > < *file-format-specific-statements* >;

Required Arguments

DATA=<*libref.*> *SAS data set* | *caslib.tablename*

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).

caslib.tablename specifies the input CAS table. You must use the DATA= two-level name (caslib and table) because you cannot specify just a one-level name for a CAS table.

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

Restrictions PROC EXPORT does not support the DROP | KEEP data set options with Name Range Lists for DBMS=CSV | TAB | DLM.

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.

The path name (which includes the file name) must be a total of 201 chars or less.

OUTFILE="*file name*" | "*fileref*"

specifies the complete path and file name, 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 For client/server applications: Specify the full path and file name of the export 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.

PROC EXPORT does not support the DROP | KEEP data set options with Name Range Lists for DBMS=CSV | TAB | DLM.

Notes The SAS PC Files Server is not included with SAS Viya 3.4 and earlier. Use the SAS 9 client.

For information about how SAS converts data types, see the specific information for the data source file format to which you are exporting.

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 Microsoft Access database files.

Optional Arguments

<*SAS-data-set-options*>
specify 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. You can add the ENCRYPTKEY= data set option to specify the key value that is required for exporting an AES-encrypted SAS data set. For information, see *SAS Data Set Options: Reference*.

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, 2007, 2010, and later database.

The following table lists the values for the DBMS= identifier when exporting files.

Table 5.1 *Data Source Identifier Summary*

DBMS= Identifier	Output Data Source	File Extension
ACCESS	Microsoft Access 2000, 2002, 2003, 2007, 2010, and later table (uses the Access LIBNAME engine)	.mdb .accdb
ACCESSCS	Microsoft Access table connecting remotely through the SAS PC Files Server (uses the PC Files LIBNAME engine transparently.)	.mdb .accdb

DBMS= Identifier	Output Data Source	File Extension
	Note: The PC Files Server is not included in SAS Viya 3.4 and earlier. Connect to it using the SAS 9 client.	
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 .dbt .fpt
DLM	delimited file (default delimiter is a blank) **Note:** delimiters are specified using the DELIMITER= statement.	.*
DTA	Stata file	.dta
EXCEL	Excel 97, 2000, 2002, 2003, 2007, 2010, and later workbook (uses the Excel LIBNAME engine transparently.)	.xls .xlsb .xlsx
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 the SAS PC Files Server (uses the PC Files LIBNAME engine underneath). **Note:** The PC Files Server is not included in SAS Viya 3.4 and earlier. Connect to it using the SAS 9 client.	.xls .xlsb xlsx
JMP	JMP files, Version 7, and later format	.jmp
PARADOX	Paradox DB files	.db
PCFS	JMP files, Stata files, and SPSS files connecting remotely through the SAS PC Files Server. The PC Files Server is not included with SAS Viya 3.4 and earlier. **Note:** Microsoft Excel files are not supported. Use DBMS=EXCELCS.	.jmp, .dta, .sav

DBMS= Identifier	Output Data Source	File Extension
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) **Note:** Transcoding is not supported for DBMS=XLS. If your file has fewer than 255 columns, use DBMS=EXCEL as an alternative. Otherwise, save your .xls file to a .xlsx file and use DBMS=XLSX to support transcoding.	.xls
XLSX	Excel 2007, 2010, and later spreadsheet (using .xlsx file format.)	.xlsx

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

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

These specifications require a connection to the SAS PC Files Server running on Windows. The server is not included in SAS Viya 3.4 and earlier. Connect to the server using the SAS 9 client.

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

- EXCEL
  - Support for exporting Excel .xlsx files directly to UNIX is new in SAS/ACCESS 9.4. In this release, you can specify DBMS=XLSX as well as DBMS=XLS in order to read and write to Excel workbooks under UNIX and Microsoft Windows directly. You do not need to use the SAS PC Files Server. SAS recommends using .xlsx for its support and enhancements.
  - Microsoft Excel 97, 2000, 2002, and 2003 share the same internal file formats. The Excel LIBNAME engine recognizes EXCEL97, 2000, 2002, 2003, EXCEL2007, EXCEL2010, and later versions as aliases for the

identifier EXCEL. By specifying DBMS=EXCEL, the exported SAS file can be read by any of these versions Microsoft Excel.

- When exporting to an existing Excel workbook .xlsb file, use DBMS=EXCEL or DBMS=EXCELCS.

  When exporting to existing Excel workbook .xls or .xlsx file, you can use DBMS=EXCEL (or DBMS=EXCELCS) or DBMS=XLS (or DBMS=XLSX). However, a .bak file is created only when DBMS=XLS or DBMS=XLSX is used.

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

Table 5.2 *Exported Data: Microsoft Excel Workbook Readability*

File-Extension	Excel 2007 and later	Excel 97, 2000, 2002, 2003
.xlsx, .xlsm, .xlsb	Yes	No
.xls	Yes	Yes

If you use the EXPORT procedure with DBMS=XLSX and the page has formulas that reference other pages, an error message is written to the log. The error states that the page cannot be replaced because it has formulas that reference other pages.

The following table is a quick reference for which DBMS= data source identifier to use for Excel files.

Table 5.3 *DBMS Data Source Identifiers*

DBMS=	Microsoft Windows	UNIX	Linux
EXCEL	yes	no	no
XLS	yes	yes	yes
XLSX	yes	yes	yes
EXCELCS	yes	yes (source data set is on UNIX; target file is on remote Windows)	yes (source data set is on Linux; target file is on remote Windows

Restriction Only Excel 2007 and later can use .xlsx, .xlsm, and .xlsb formats.

Note Transcoding or multi-byte characters are not supported for DBMS=XLS. The output yields unpredictable results. If you use Microsoft Windows and your file does not contain more than 255 columns, use

DBMS=EXCEL for transcoding. If you use UNIX, or if your file has more than 255 columns, save your file as .xlsx and use DBMS=XLSX for transcoding.

See Chapter 6, "File Format-Specific Reference for the IMPORT and EXPORT Procedures".

- PCFS
  - Specify DBMS=PCFS for JMP, SPSS, and Stata files to use the client/server model. This enables you to export data from UNIX or Microsoft Windows operating environments to JMP, SPSS, or Stata files on a remote Microsoft Windows system. You must connect to a SAS PC Files Server on Microsoft Windows. The SAS PC Files Server is not included in SAS Viya 3.4 and earlier. Use the SAS 9 client to access the PC Files Server.
  - For more information, see "Importing and Exporting SAS JMP Files Data" on page 121, "Importing and Exporting SPSS Files" on page 129, and "Stata DTA Files" on page 134.
- ACCESS
  - Microsoft Access versions 2000, 2002, and 2003 share the same internal file formats. The following as aliases are recognized for DBMS=ACCESS: ACCESS2000, ACCESS2002, ACCESS2003, ACCESS2007, ACCESS2010, and later versions. By specifying DBMS=ACCESS, the exported SAS file can be read by any of these versions of Microsoft Access.
  - 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, Access 2003, Access 2007, Access 2010, and later 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= connection option is specified for a Microsoft Access .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 a database (.mdb file) that Access 2000–2013 can read. Access 97 cannot read this file.

    For more information about the DATABASE= connection option, see "Microsoft Access Database Files" on page 93.

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

Table 5.4 *Exported Data: Microsoft Access Readability*

File-Extension	Access 2007 and later	Access 2000, 2002, 2003
.mdb	Yes	Yes
.accdb **Note:** Only Access 2007 and later can open .accdb file formats.	Yes	No

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.

You can either replace an .xls or .xlsx worksheet in an existing workbook, or you can add a new .xlsx worksheet in an existing workbook. Adding a worksheet applies to an .xlsx file format but not to the .xls format. For more information and examples, see "Example 1: Import a Microsoft Access 2007 Database Table to a SAS Data Set" on page 105.

You can replace a worksheet in an Excel workbook using DBMS=EXCEL or DBMS=EXCELCS. You can also add a new worksheet to an existing workbook.

See also the NEWFILE= option on page 73, which specifies whether to delete the Excel file and load the data to a sheet in a new Excel file, when exporting a SAS data set to an existing Excel file (.xls or .xlsx file).

<file-format-specific-statements>
See Chapter 6, "File Format-Specific Reference for the IMPORT and EXPORT Procedures" for the supported syntax for your DBMS.

6

File Format-Specific Reference for the IMPORT and EXPORT Procedures

Delimited Files ... **53**
- Overview ... 53
- CSV Files ... 54
- Tab-Delimited Files ... 54
- Other Delimiters ... 54
- External File Interface (EFI) ... 55
- Sharing Delimited Files across Hosts ... 56
- Processing Delimited Files in SAS ... 56
- IMPORT and EXPORT Procedure Statements for Delimited Files ... 57
- Example 1: Import a Tab-Delimited File into SAS ... 59
- Example 2: Import a Space-Delimited File into SAS ... 60
- Example 3: Import a CSV File with a Single Quotation Mark in the File Name ... 60
- Example 4: Export a SAS Data Set to a CSV File ... 60
- Example 5: Import a Subset of a CSV File into SAS ... 61
- Example 6: Export a SAS Data Set That Has a Single Quotation-Mark Name ... 61

Microsoft Workbook Files ... **62**
- Microsoft Excel Files Essentials ... 62
- Excel Data Types ... 64
- Excel Numeric Date and Time Values ... 64
- Excel File Formats ... 65
- Using the DBMS= Identifiers for Microsoft Excel Files ... 65
- Importing and Exporting Microsoft Excel 4 and Excel 5 Files ... 81
- Example 1: Import a SAS Data Set to an Excel 5 File ... 83
- Example 2: Export a SAS Data Set to an Excel 5 File ... 83
- Import and Export Microsoft Excel Files Using the XLS and XLSX Identifiers ... 83
- Example 1: Import an XLSX File to a CAS Table ... 88
- Example 2: Export a CAS Table to an XLSX File ... 90
- Example 3: Export SAS Data Sets to Excel 2010 Workbook and Replace Sheets ... 91
- Example 4: Export SAS Data Sets to Excel 2010 Workbook and Add a New Sheet ... 91
- Example 5: Import Data Using a Range Name ... 91
- Example 6: Import Data Using an Absolute Range Address ... 92
- Exporting and Importing Excel Files with the SAS PC Files Server ... 92

Example 1: Export a SAS Data Set on UNIX to an Excel File on Microsoft Windows ... 93
Example 2: Import an Excel File on Microsoft Windows to a SAS Data Set on UNIX ... 93

Microsoft Access Database Files ... **93**
Microsoft Access File Essentials ... 93
Microsoft Access Data Types ... 94
The Conversion of Date and Time Values between SAS Data Sets and Microsoft Access Database ... 96
Using the DBMS= Identifiers for Microsoft Access Database Files ... 97

Lotus 1-2-3 WKn Files ... **107**
WKn Files Essentials ... 107
WKn Data Types ... 108
Supported Import and Export Methods and Statements for WKn Files ... 109
Example 1: Export a SAS Data Set to a Lotus WK4 File ... 111
Example 2: Import a Lotus WK4 File Data to a SAS Data Set ... 111
Example 3: Export Data to a Lotus WK1 File from a SAS Data Set ... 111
Example 4: Import Data from a Lotus WK1 File to a SAS Data Set ... 112

dBase DBF Files ... **112**
dBase DBF Files Essentials ... 112
DBF Data Types ... 113
Setting Environment Variables and System Options ... 114
Supported SAS IMPORT and EXPORT Procedure Statements ... 115
Example 1: Export Data to a dBase DBF File from a SAS Data Set ... 116
Example 2: Import Data from a dBase DBF File into a SAS Data Set ... 117
Example 3: Export Data to a dBase DBF File from a SAS Data Set Using Encoding ... 117
Example 4: Import and Translate Data from a dBase DBF File ... 117

dBase DBF MEMO Files ... **118**
Overview ... 118
Import Data from a dBase DBF File with Memo Field into a SAS Data Set ... 118

JMP Files ... **119**
JMP File Essentials ... 119
JMP Missing Values ... 119
JMP Data Types ... 119
Importing and Exporting SAS JMP Files Data ... 121
IMPORT Procedure and EXPORT Procedure Supported Syntax ... 122
Example 1: Import a JMP File to a CAS Table ... 122
Example 2: Export a CAS Table to a JMP File ... 123
Example 3: Export a SAS Data Set to a JMP File ... 124
Example 4: Export a SAS Data Set on UNIX to a JMP File Using DBMS-=PCFS ... 125
Example 5: Import a JMP File to a SAS Data Set ... 125
Example 6: Export a SAS Data Set on UNIX to a JMP File on Microsoft Windows 125
Example 7: Import Data from a JMP File on Microsoft Windows to a SAS Data Set on UNIX ... 126

Paradox DB File Formats ... **126**
Paradox File Essentials ... 126
Export a SAS Data Set to a PARADOX DB File ... 126
Import a SAS Data Set from a Paradox DB File ... 127

SPSS SAV Files ... **127**
SAV File Essentials ... 127

SPSS Data Types 127
Importing and Exporting SPSS Files 129
Import Procedure and the Export Procedure Supported Syntax 129
Example 1: Import an SPSS SAV File to a CAS Table 130
Example 2: Export a CAS Table to an SPSS SAV File 131
Example 3: Export a SAS Data Set to an SPSS SAV File 132
Example 4: Import a SAS Data Set from an SPSS SAV File 133
Example 5: Import Data from an SPSS SAV File and Apply FMTLIB= Statement 133
Example 6: Export a SAS Data Set on UNIX to an SPSS File on Microsoft Windows 133
Example 7: Import Data from an SPSS File on Microsoft Windows to a SAS Data Set on UNIX 134

Stata DTA Files **134**
Stata DTA Files Essentials 134
Stata Data Types 136
Importing and Exporting Stata Data Files 136
IMPORT and EXPORT Procedures Supported Syntax 137
Example 1: Import a Stata DTA File to a CAS Table 137
Example 2: Export a CAS Table to a Stata DTA File 138
Example 3: Export a SAS Data Set to a Stata DTA File on a Local System 139
Example 4: Import a Stata DTA File on a Local System to a SAS Data Set 140
Example 5: Export a SAS Data Set on UNIX to a Stata File on Microsoft Windows 140
Example 6: Import Data from a Stata DTA File on Microsoft Windows to a SAS Data Set on UNIX 140

Delimited Files

Overview

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. Delimited files can be specified using DBMS=CSV, DBMS=TAB, and DBMS=DLM.

Delimited files have these characteristics:

- 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 newline character indicates a new row.

Note: Support of delimited files is included in Base SAS. A license for SAS/ACCESS Interface to PC Files is not required to use this list of features.

By default, the IMPORT procedure reads delimited files as varying record-length files. If your external file has a fixed-length format, use a SAS DATA step with an INFILE statement that includes the RECFM=F and LRECL= options. For more information, see the INFILE statement, RECFM= option.

CSV Files

A comma-separated values (CSV) file is a delimited file specified with DBMS=CSV. The data values are separated by commas. In a .csv 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 specified with DBMS=TAB and have a .txt extension. 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 delimited files. Such files can be specified with DBMS=DLM and can have a .txt extension, .fil extension, or other delimited file extension. Specifying DBMS=DLM requires you to specify the character that is used as the delimiter in that file, for example, DELIMITER="~".

External File Interface (EFI)

The SAS Import and Export Wizards use the SAS External File Interface 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.

Note: The wizards and the EFI are not included in SAS Viya but you can use them in the SAS 9 client.

- 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 on page 15.
- The delimiter can be in hexadecimal. For example, the tab delimiter can be specified as `delimiter='09'X`
- As the IMPORT procedure reaches the end of the current input data row, variables without any values are set to missing.
- For the IMPORT and EXPORT procedures, a SAS data set name can contain a single quotation mark when the VALIDMEMNAME=EXTEND system option is also specified. Using VALIDMEMNAME= expands the rules for the names of certain SAS members, such as a SAS data set name.

  See also these examples:

  - “Example 3: Import a CSV File with a Single Quotation Mark in the File Name” on page 60
  - “Example 6: Export a SAS Data Set That Has a Single Quotation-Mark Name” on page 61

See Also

- “Importing and Exporting Data with the Wizards” on page 15
- Chapter 4, “IMPORT Procedure,” on page 31

Sharing Delimited Files across Hosts

When a delimited file is read into SAS using the IMPORT procedure, each row must end with an end-of-line character or a host-specific delimiter; each host operating environment has a default end-of-line delimiter. If you share delimited files that were created on one host with another host, the default end-of-line delimiters might not match. When this occurs, you must specify the new host's end-of-line delimiter for your files.

Microsoft Windows
: The default newline delimiter is a two-byte hexadecimal sequence for Carriage Return and Linefeed. To read a file that is native to UNIX use a FILENAME statement with the TERMSTR=LF option.

UNIX
: The default end-of-row delimiter is the hexadecimal character for Linefeed. To read a file that is native to Windows, use a FILENAME statement with the TERMSTR=CRLF option.

Processing Delimited Files in SAS

When you use PROC IMPORT to read a comma-separated file, a tab-separated file, or other delimited file, the procedure performs these actions by default:

- scans the first 20 rows
- collects the column names from the first row
- scans the remaining 19 rows and determines the column type
- assigns an informat and a format to each column
- creates a DATA step with an INPUT statement
- submits all of the code to the DATA step compiler, which, in turn, executes the code.

You can change the default actions using PROC IMPORT statements.

This example shows an IMPORT procedure reading a file that has a tilde (~) as the character delimiter:

```
PROC IMPORT DATAFILE='c:\mydata\test.fil'
    DBMS=DLM OUT=work.test REPLACE;
    DELIMITER="~";
   RUN;
```

If you need to revise your code after the procedure runs, issue the RECALL command (or press F4) to the generated DATA step. At this point, you can add or remove options from the INFILE statement and customize the INFORMAT, FORMAT, and INPUT statements to your data.

If you use this method and you modify an informat, also modify the format for that same column. The informat and format for a given column also must be of the same type (either character or numeric). In addition, if the type is character, the assigned format should be as long as the column to avoid truncation when the data is displayed. For example, if a character column is 400 characters long but has a format of $char50, only the first 50 characters are shown when the data is displayed.

To your PROC IMPORT code, issue a second RECALL command (or press F4 again).

By default, PROC IMPORT expects the column names to appear in the first row. The procedure scans the first 20 rows to count the columns, and it attempts to determine the correct informat and format for each column. (You can modify this default number of rows.) You can use these procedure statements on page 57 to accomplish these tasks.

- indicate how many rows SAS scans for columns to determine the type and length (GUESSINGROWS=)
- indicate at which row SAS begins to read the data (DATAROW=)
- modify whether SAS extracts the column names (GETNAMES=).

IMPORT and EXPORT Procedure Statements for Delimited Files

The supported delimited file types include comma-separated values (.csv files), tab-separated values (usually .txt) or other delimiter separated file for which you specify the delimiting character with the DELIMITER= statement.

See: "Example 1: Import a Tab-Delimited File into SAS" on page 59 and "Example 4: Export a SAS Data Set to a CSV File" on page 60

Table 6.1 *IMPORT and EXPORT Procedure Statements*

DBMS= identifier	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 | MAX	20
	PUTNAMES	No	Yes	Yes | No	Yes
DLM	DATAROW	Yes	No	1 to 2147483647	2

DBMS= identifier	Statement Options	PROC IMPORT	PROC EXPORT	Valid Value	Default Value
	DELIMITER	Yes	Yes	'character' | 'nn'x	' '
	GETNAMES	Yes	No	Yes | No	Yes
	GUESSINGROWS	Yes	No	1 to 2147483647 | MAX	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=*'character' | 'nn'x*
specifies the delimiter (either a single character or hexadecimal value) that separates the columns of data for the IMPORT and EXPORT procedures. Delimiters are case-sensitive and enclosed in quotation marks, such as `'&'`.

Default A blank character

Restriction If you specify DBMS=DLM, you must also specify the DELIMITER= statement.

Examples

"Exporting to a Delimited External Data Source" in *Base SAS Procedures Guide*

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 row contains special characters that are not valid in a SAS name, SAS converts the character to an underscore (_). For example, the variable `EMP ID` becomes `EMP_ID`.

Starting in SAS 9.4, GETNAMES=YES prefixes an underscore to the data value rather than replacing the value's first special character. For example, `2013.Changes` becomes `_2013_Changes`.

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

NO specifies that the IMPORT procedure generate SAS variable names as VAR1, VAR2, and so on.

Default Yes

Restrictions Valid only for the IMPORT procedure.

If VALIDVARNAME=ANY is used, GETNAMES= might not prefix an underscore to the data value.

Example "Importing a Delimited File" in *Base SAS Procedures Guide*

GUESSINGROWS=*n* | MAX
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. MAX can be specified instead of the maximum number, 2147483647. (Specifying the maximal value could adversely affect performance.)

Default 20

Range 1 to 2147483647, where MAX= 2147483647

PUTNAMES=YES | NO
determines whether to write SAS variable names as column headings to the first row of the exported data file.

YES specifies that PROC EXPORT is to write SAS variable names as column names (or headings) to the first row of the exported file. It writes the first row of SAS data to the second row of the exported data file.

NO specifies that PROC EXPORT is to write the first row of SAS data to the first row of 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 "Exporting to a Tab Delimited File with the PUTNAMES= Statement" in *Base SAS Procedures Guide*

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 character value for a tab (`'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. The delimiter `'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: Import a CSV File with a Single Quotation Mark in the File Name

You can import a .csv file into SAS, even if the file name contains a single quotation mark (such as “game’s_invoice.csv”) or special characters. Using VALIDMEMNAME=EXTEND expands the rules for the names of certain SAS members, as indicated by the use of the n-literal after the output SAS data set name.

```
OPTIONS VALIDMEMNAME=EXTEND;
PROC IMPORT OUT=WORK."The Night Before Christmas Game"n
     DATAFILE="c:\temp\game's_invoice.csv"
   DBMS=CSV REPLACE;
   GETNAMES=YES;
   DATAROW=2;
RUN;
```

Example 4: 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. Because PUTNAMES=NO, the first row of the SAS data set is written to the first row of the .csv file. This means that the columns of data are unnamed.

```
LIBNAME SDF "&sasdir";
```

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

Example 5: 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;
     GETNAMES=YES;
     DATAROW=6;
RUN;
OPTIONS OBS=MAX;
```

Example 6: Export a SAS Data Set That Has a Single Quotation-Mark Name

In this example, a temporary SAS data set is created and contains single quotation marks in its name. Then it shows two ways that you could use to export the data set to a .csv file. Notice the different values for the VALIDMEMNAME= option; each value has different rules for using special characters in SAS data set names.

```
OPTIONS VALIDMEMNAME=EXTEND;
DATA WORK.'Region Four''s YTD Results'n;
    SET SASHELP.CLASS;
RUN;

OPTIONS VALIDMEMNAME=COMPAT;
PROC EXPORT DATA=WORK."Region Four's YTD Results"n
   OUTFILE="c:\temp\Region_Four_Results.csv"
   DBMS=CSV REPLACE;
RUN;

OPTIONS VALIDMEMNAME=EXTEND;
PROC EXPORT DATA=WORK.'Region Four''s YTD Results'n
   OUTFILE="c:\temp\Region_Four_Results.csv"
   DBMS=CSV REPLACE;
RUN;
```

Microsoft Workbook Files

Microsoft Excel Files Essentials

SAS/ACCESS Interface to PC Files works with Microsoft Excel workbook 5, 95, 97, 2000, 2002, 2003, 2007, 2010, and later. These files are referred to in this document by the file extension names .xls, .xlsx, .xlsb, or .xlsm files. Not all files are supported in all cases. For example, you cannot create an .xlsm file using PROC EXPORT with DBMS=EXCELCS, or from the PC Files LIBNAME engine.

An Excel file represents an Excel workbook. An Excel workbook is a collection of worksheets (also called, spreadsheets).

A cell is the intersection of a column and a row. It is referenced by a column number and a row number (for example, B5). A cell is the basic unit that saves data in the worksheet. A cell can contain a numeric value or a text value of up to 32,767 characters.

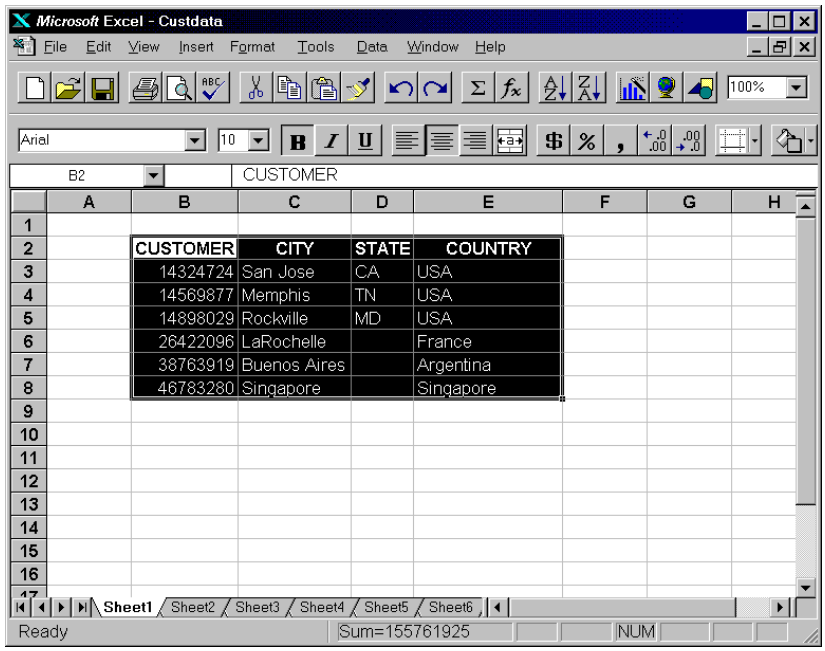
A range is a subset of cells in a worksheet. Its address identifies it. It begins with the name of the top left cell and ends with the name of the bottom right cell, separated by a colon. For example, the range B2:E8 is the range address for a rectangular block of 28 cells, where the top left cell is B2 and the bottom right cell is E8 (shown as shaded, in Figure 6.1). A range name identifies a range. A range specification can also define the rectangle directly by specifying `range="sheetname$upper_left:lower_right"` (for example, `range="inventory$B10:R100"`). A range option also works, while not using the sheet name for example, range="*$upper_left*:*lower_right*". In this case the first sheet of the Excel file is considered the sheet name. A range is equivalent to a SAS table or SAS data set.

A worksheet in an .xls file can save up to 256 columns and 65,536 rows. Excel 2007 and later files (.xlsx files) have been enhanced to support 16,384 columns and 1,048,576 rows in a worksheet.

Files that are created with Excel 2007 and later can have an .xlsx, .xlsb, or .xlsm extension. Note that if you create an .xlsx file with PROC EXPORT using DBMS=EXCELCS, it contains an extra sheet named _SAS_EMPTY_ that cannot be deleted by the Microsoft ACE driver. You can ignore this sheet.

Due to how the Microsoft ACE driver and the Microsoft Jet driver work, DBMS=EXCEL and DBMS=EXCELCS (which use the Excel or PC Files LIBNAME engines transparently) are limited to 255 columns in an Excel file.

Figure 6.1 *A Range of Data in an Excel Worksheet*



SAS/ACCESS treats an Excel workbook as a database and a worksheet or 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 workbook. 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, specify `SHEET="'Sheet1$'n"`, where the sheet name is enclosed in double quotation marks. The first row of data in a range is usually treated as a column heading and used to create a SAS variables name.

Remember these points as you work with Microsoft Excel files.

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

Excel Data Types

Microsoft Excel software has two data types: character and numeric, where numeric includes certain values for currency, dates, time, and so on.

- 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 bytes. When character data is imported using the SAS V9 engine, it is stored in the data set as a CHAR. When character data is imported into a CAS table using the CAS engine the data is stored as a VARCHAR in the CAS table in memory. Note that CAS supports only UTF-8 encoding.
- Numeric data can be numbers, currency values, formulas, or error values, such as, #NULL!, #N/A, #VALUE!.

  When SAS scans a column of data and if all the values are numbers, it is defined as numeric data.

  When reading an Excel numeric value with a CURRENCY format, SAS automatically applies the NLMNYw.d format so that the data is displayed in the currency that matches the locale.

Excel Numeric Date 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). A number can have both a date portion 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 these Excel formats.

- Excel 2007 and later Workbook creates an Excel .xlsx file.
- Excel 97–2003 Workbook creates an Excel .xls file.
- Excel Binary Workbook creates an Excel .xlsb file.
- Excel Macro-Enabled Workbook creates an Excel .xlsm file.

Using the DBMS= Identifiers for Microsoft Excel Files

Overview

DBMS= identifiers are used with PROC IMPORT and PROC EXPORT to specify the Microsoft Excel file format to import or export. Specifications for accessing Microsoft Excel files include DBMS=EXCEL, DBMS=EXCELCS, DBMS=XLS, and DBMS=XLSX.

Several DBMS= identifiers use a LIBNAME engine transparently (for example, DBMS=EXCEL uses the Excel LIBNAME engine and DBMS=EXCELCS uses the PC Files LIBNAME engine). The PC Files engine requires the PC Files Server, which is not included in SAS Viya 3.4 and earlier. You can connect to it using the SAS 9 client.

DBMS= identifiers for Microsoft Excel Files are described below.

DBMS=EXCEL
: enables you to read and write Microsoft Excel 97, 2000, 2002, 2003, 2007, 2010, and later workbooks. DBMS=EXCEL uses a LIBNAME statement transparently. Supports .xls .xlsb .xlsm .xlsx files. For more information about .xlsb files and Linux, see “.XLSB Files on Linux” on page 66.

DBMS=XLS and DBMS=XLSX
: enables you to read data from or write data to an .xls or .xlsx file. DBMS=XLS supports Excel versions 5/95, 97, 2000, 2002, and 2003. DBMS=XLSX supports 2007, 2010, and later. SAS recommends you use the .xlsx file format for its enhanced support. This method does not support Excel .xlsb files.

  Because an .xlsx file has a limit of 16,384 columns and 1,048,576 rows, when you write (export) data from a SAS data set, it cannot have more than 16,384 columns or 1,048,576 rows. Otherwise, an error message is displayed.

DBMS=XLSX supports double-byte character sets (DBCS). To get DBCS for an .xls file, save it to a .xlsx format.

DBMS=EXCELCS
connects through the SAS PC Files Server to read and write Excel workbooks on the server. EXCELCS uses the PC Files engine transparently. The PC Files Server is not included in SAS Viya 3.4 and earlier. Connect using the SAS 9 client. The PC Files Server runs only on Windows.

Be aware that the character set (encoding) that you use must match the set for your data source, and make sure that your character set does not use any invalid characters. For example, although UTF-8 is compatible with WLATIN1 (7-bit ASCII), it is not compatible with WLATIN2 (8-bit ASCII). Therefore, you cannot store 8-bit ASCII characters in a UTF-8. These 8-bit ASCII characters, such as an accented letter in French, must be properly enclosed using UTF-8, which generates multi-byte or double-byte characters.

Use the CAS engine to import a file into a CAS table. This requires connecting to the SAS CAS server. For more information and an example of connecting to the CAS server, see *SAS Cloud Analytic Services: User's Guide*

.XLSB Files on Linux

The SAS PC Files Server is the recommended way of importing .xlsb files into SAS.

These specifics are for importing an .xlsb file.

- The .xlsb files can only be processed using the Microsoft ACE driver, which requires either the Excel or PC Files engine. When you use PROC IMPORT or PROC EXPORT, use `DBMS=EXCEL` or `DBMS=EXCELCS`.
- The Excel engine or PC Files engine can read any of the .xls* formats, but can only write to .xls, .xlsx, and .xlsb files and cannot write to .xlsm files.
- The Excel engine (`DBMS=EXCEL`) can only be used on Windows. Also, the SAS bitness (32 or 64) must match the bitness of the installed ACE driver. ACE is often installed via Microsoft Office and the bitness is often fixed at 32 bits.
- If the bitness is different on Windows or SAS on UNIX or Linux, you have to use the PC Files engine `DBMS=EXCELCS` or `DBMS=ACCESSCS` for Microsoft Access files. The PC Files Server is used to either bridge the bitness gap or the gap between UNIX and Windows.
- The PC Files Server can access any Excel or Microsoft Access file that is accessible via a file path name on the machine that has the PC Files Server. This includes local files, such as **`C:\mydir\myfile.xlsb`** or networked files using a UNC (Universal Naming Convention) path name, such as **`\\server\mydir\myfile.xlsb`**. It is irrelevant whether the networked file is physically located on another Windows machine, a UNIX machine, or a disk server, as long as it can be accessed via a UNC path name from the machine that is running the PC Files Server. It also doesn't matter which particular file extension (of the four allowable extensions for Excel files) the file has. However, note the restriction in the second bullet above.

PROC IMPORT and EXPORT Statement Options for Microsoft Excel Files

The DBMS=EXCEL and DBMS=EXCELCS specifications for importing and exporting Microsoft Excel workbook files generate SAS LIBNAME statement code transparently. The DBMS=EXCEL identifier uses the Excel LIBNAME engine. The DBMS=EXCELCS identifier uses the PC Files LIBNAME engine. The DBDSOPTS= option enables you to specify data set options in the PROC statement that the LIBNAME engine underneath supports. The Excel and PC Files LIBNAME statements use the Microsoft ACE driver or Microsoft Jet driver to access data in Microsoft Excel workbook files.

This table shows the IMPORT and EXPORT procedure statement options that are valid for Microsoft Excel data residing locally on your PC.

Table 6.2 *Statement Options to Import or Export Excel Data That Resides Locally*

DBMS= Identifier	Option	Valid Value	Default Value	PROC IMPORT	PROC EXPORT
EXCEL	DBDSOPTS	Excel engine 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 PROC statement options that are valid when you use DBMS=EXCELCS to import or export Excel data files on a remote Microsoft Windows system from UNIX or another Windows machine. PC Files LIBNAME engine statements are generated, which enables you to specify statement options that the PC Files engine supports. Specifying DBMS=EXCELCS requires that the

SAS PC Files Server is running on a Microsoft Windows operating system where the Excel files reside. The PC Files Server is not included in SAS Viya 3.4 and earlier. Connect to it using the SAS 9 client.

Table 6.3 *Statement Options to Import or Export Excel Data Files on the PC Files Server*

DBMS= Identifier	Option	Valid Value	Default Value	PROC IMPORT	PROC EXPORT
EXCELCS	DBDSOPTS	PC Files engine 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 | 2010 | and later formats	'97'	Yes	Yes

DBDSOPTS=*'valid data set options for the Excel and PC Files LIBNAME engines'*
enables you to specify data set options that are valid for the Excel or PC Files LIBNAME engines when you specify DBMS=EXCEL or DBMS=EXCELCS in the PROC statement. These options are for advanced users who are familiar with the Excel LIBNAME engine or the PC Files LIBNAME engine. The PC Files engine is not included in SAS Viya 3.4 and earlier because it requires the SAS PC

Files Server. Use the SAS 9 client to import or export data using the PC Files Server.

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 "SAS Data Set Options for the LIBNAME Statement for PC Files Engine Data Set Option" on page 271 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 how the Microsoft Jet driver and Microsoft ACE driver work, no more than 64 characters of column names are written to SAS variable labels.

Due to how the Microsoft Jet Excel driver and Microsoft ACE driver work, 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 (_). For example, the data value `Occupancy Code` becomes the SAS variable name `Occupancy_Code`.

Starting in SAS/ACCESS 9.4, GETNAMES= prefixes an underscore to the data value rather than replacing the value's first character. For example, `2013.Changes` becomes `_2013_Changes`.

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.

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. The .xlsx and .xls formats assume MIXED=YES.

Note: Due to how the Microsoft ACE driver and the Microsoft Jet Excel driver work, 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.

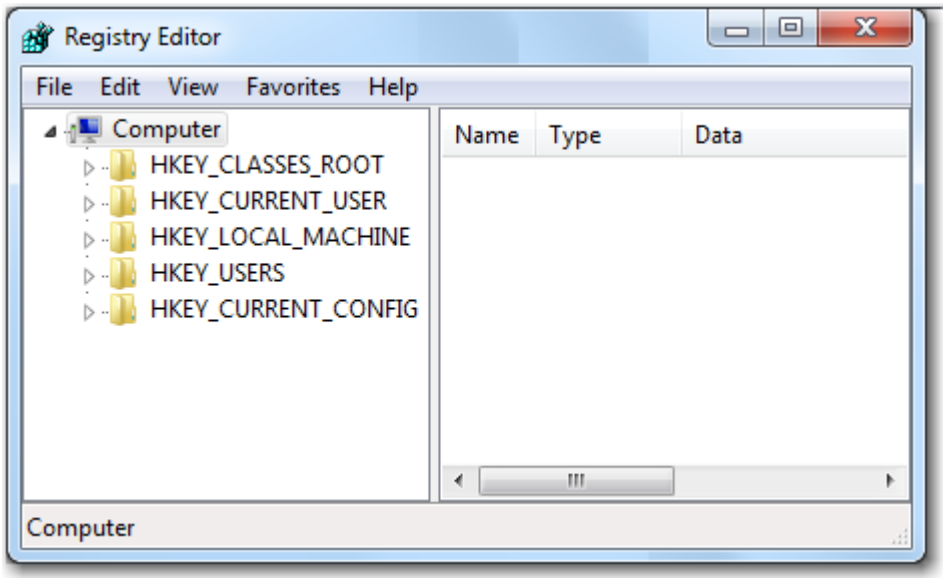
Default NO

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 the Windows registry follow these steps:

1 Open the Registry Editor by selecting **Start** ⇨ **Run** from your desktop and enter `regedit`.



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

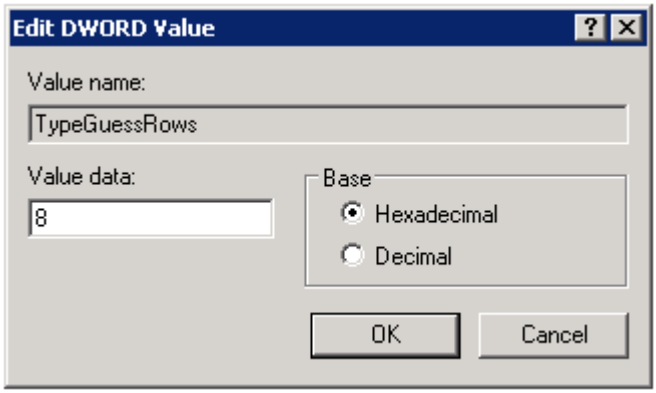
Table 6.4 *Registry Key for TypeGuessRow Based on Microsoft Office Version or Microsoft Engine*

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 and 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 and later 32–bit	**HKEY_LOCAL_MACHINE ⇨ Software ⇨ Wow6432Node ⇨ Microsoft ⇨ Office ⇨ 14.0 ⇨ Access Connectivity Engine ⇨ Engines ⇨ Excel**
Windows 7 or X64 system	Office 2013 32-bit	**HKEY_LOCAL_MACHINE ⇨ Software ⇨ Wow6432Node ⇨ Microsoft ⇨ Office ⇨ 15.0 ⇨ Access Connectivity Engine ⇨ Engines ⇨ Excel**
Windows 7 or X64 system	Office 2010 and later 64-bit	**HKEY_LOCAL_MACHINE ⇨ Software ⇨ Microsoft ⇨ Office ⇨ 14.0 ⇨ Access Connectivity Engine ⇨ Engines ⇨ Excel**

Environment	Office Version or Engine	Registry Key
Windows 7 or x64 system	Office 2013	**HKEY_LOCAL_MACHINE ⇨ Software ⇨ Microsoft ⇨ Office ⇨ 15.0 ⇨ Access Connectivity Engine ⇨ Engines ⇨ Excel**
Windows	Microsoft ACE Engine	**HKEY_LOCAL_MACHINE ⇨ Software ⇨ Microsoft ⇨ Office ⇨ 12.0 ⇨ Access Connectivity Engine ⇨ Engines ⇨ Excel**
Windows	Microsoft Jet Engine (when using MSENGINE=JET only)	**HKEY_LOCAL_MACHINE ⇨ Software ⇨ Microsoft ⇨ Jet ⇨ 4.0 ⇨ Engines ⇨ Excel**

3 In the right pane, double-click **TypeGuessRows**.

4 Change the **Value data** entry from 8 to 0 and select **OK**.



5 Exit from the Registry Editor window.

This table describes the registry settings for the MIXED= option.

Table 6.5 *Registry Settings for the MIXED= Option*

TypeGuessRows	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. 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 driver 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.
ImportMixedTypes	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. 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 driver and the Microsoft Jet driver always assign character type for columns with data of mixed data types. The numeric data is converted to character data.

NEWFILE=YES | NO

when exporting a SAS data set to an existing Excel file (.xls or .xlsx 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.

> **TIP** Because of how the Microsoft ACE driver works, Excel .xls files allow only 65,536 rows per file. If you require more than 65,536 rows per file, it is recommended that you save your data to an .xlsx file format.

Restriction Available only when DBMS=EXCEL

PORT=*port-number*

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

Alias PORT_NUMBER

Default 9621

Restrictions Available only when DBMS=EXCELCS. Requires SAS PC Files Server which is not included in SAS Viya 3.4 and earlier. Connect to the server using the SAS 9 client.

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

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

subsets a worksheet by identifying the rectangular set of cells to import.

If you omit the RANGE= statement, the IMPORT procedure tries to select an existing worksheet and import the entire worksheet. You might get more columns or missing data than you want. The worksheet selected also might not be the one that you wanted. Therefore, it is strongly recommended that you use the RANGE= statement.

The *range-name* is a user-defined table name that represents a range of cells within the worksheet in the Excel file. The *range-name* is not case sensitive and does not allow any special character except for 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. 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.

An absolute range identifies the top left cell that begins the range and the bottom right cell that ends the range. The absolute range is not necessarily the entire worksheet but it can be. You can name the range with a worksheet name and append a `$`. Hence, the range `"Invoice$"` refers to data in the whole worksheet.

You can use RANGE= to specify the row number where PROC IMPORT starts to read data. Set the end point to `"0"`, and the code then determines the last row and last column. Specify RANGE="*Sheetname*$A#:0"; where # is the first data row. Thus, `RANGE="sheet1$A3:0";` starts to read the data at row 3. If you use RANGE= for this purpose, do not specify the DATAROW= statement.

If GETNAMES=YES is set, the first row of data in the range is used for the column names, and the data starts from the second row in the range. If GETNAMES=NO is set, the data starts from the first row and column names are generated by the IMPORT procedure.

You can use the DATASETS procedure to list the SAS data set names that are mapped to the *range-names*. If the displayed *range-name* contains single quotation marks, keep the single quotation marks as part of the *range-name* to access the sheet, and enclose the entire name in double quotation marks.

This examples demonstrate the use of RANGE=.

- To retrieve data from the worksheet for two separate sheet names, `'My#Test $'` and `'CustomerOrders'`, use one RANGE= statement in each PROC IMPORT step. Only one RANGE= statement is used in a PROC IMPORT step. Note that the name must be enclosed in quotation marks:

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

Note: If you want to read data from two ranges in the same Excel workbook file, two PROC IMPORT steps must be submitted.

- 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 worksheet 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$a4:b20"` or `RANGE='summary$a4:b20'n`
- 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 quotation marks. For example, specify `RANGE='sheet_name$A1:C7'n;`. See also the "SHEET=*sheet-name*" on page 78.

Restriction Supported for only 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 driver 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 71.

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 driver 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 76 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 not to scan the time column. The DATETIME format is assigned if USEDATE=NO. The TIME. format is assigned if USEDATE=YES.

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

specifies the name of the SAS PC Files Server, where *SAS-PC-Files-Server-name* can be either the computer name or the associated IP address. You must bring up the listener on the SAS 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 SAS PC Files Server.

Alias SERVER_NAME

Restrictions For use with the client server model (DBMS=EXCELCS).

Available only from SAS 9 when DBMS=EXCELCS. Requires SAS PC Files Server which is not included in SAS Viya 3.4 and earlier.

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

See *SAS PC Files Server: Installation and Configuration Guide*.

SERVERPASS=*'server-user-password'*
specifies the password for the user ID specified with SERVERUSER= option. If the account has no password, omit this option. Always enclose the value in quotation marks, as this preserves the case of the password.

Alias SERVERPASSWORD | SERVERPW | SERVERPWD

Restriction For use with the client server model (DBMS=EXCELCS)

Requirement Available only from SAS 9. Requires SAS PC Files Server which is not included in SAS Viya 3.4 and earlier.

Note Passwords are generally case sensitive.

Example PROC EXPORT using explicit user name and password for the SAS PC Files Server:

```
PROC EXPORT DATA=SDF.INVOICE
              OUTFILE="&tmpdir.customer.xlsx"
        DBMS=EXCELCS  REPLACE
            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 the SAS PC Files Server. Always enclose the value in quotation marks. Otherwise, the backslash can be misinterpreted by the SAS parser.

Alias SERVERUID

Restriction For use with the client server model (DBMS=EXCELCS)

Requirement Available only from SAS 9. Requires the SAS PC Files Server which is not included in SAS Viya 3.4 and earlier.

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

Example See the example for SERVERPASS.

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 SAS PC Files Server control panel screen when it is started on the PC. The service name needs to be defined on your UNIX or local Windows machine and on your SAS PC Files Server running on a remote Windows machine.

Alias SERVICE_NAME

Restrictions For use with the client/server model (DBMS=EXCELCS).

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

Requirement Available only from SAS 9. Requires SAS PC Files Server which is not included in SAS Viya 3.4 and earlier.

SHEET=*sheet-name*

identifies a particular worksheet in an Excel workbook. Use SHEET= when you want to import the entire worksheet.

If both the sheet name and range name are omitted, the IMPORT procedure tries to select an existing worksheet and import the entire worksheet. You might get more columns or missing data than you want. The sheet selected might not be the one that you wanted. It is strongly recommended that you specify the RANGE= statement to import that table that you want.

A worksheet is not a table but rather, a container of tables. A worksheet name with `$`, such as `"Invoice$"`, is a range covering all data in the Invoice worksheet. When you delete the range `"Invoice$"`, you delete all the data in the worksheet. The worksheet, as the container, remains. Therefore, the SAS log indicates that the data is deleted but that the worksheet name still exists.

The *sheet-name* can contain up to 31 characters. These examples demonstrate how SAS converts non-compliant sheet names.

- If the sheet name in the EXPORT procedure contains a special character (such as a forward slash), SAS converts it to an underscore (_). *Employees/Current* becomes `Employees_Current`.
- If the sheet name contains single quotation marks, enclose the name in double quotation marks to preserve the single quotation marks as part of the sheet name. Otherwise, you cannot access the sheet: `SHEET="'My#Test'";`

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

Restrictions Valid only on a Microsoft Windows PC.

For use with the client server model (DBMS=EXCELCS).

Requirement Available only from SAS 9. Requires the SAS PC Files Server which is not included in SAS Viya 3.4 and earlier.

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, in bytes, 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. If your data includes time values, specify USEDATE=NO.

See SCANTIME= statement on page 76 to assign the appropriate TIME format.

"Processing Date and Time Values between SAS and Microsoft Excel" on page 208

VERSION=*file-version*
specifies the version of the file that you want to create. Values include 2010 and later, 2007, 2003, 2002, 2000, 97, 95, and 5. It is optional to enclose the version in quotation marks.

Default 97 for .xls files

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

For use with the client/server model (DBMS=EXCELCS).

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

Export a SAS data set called Sdf.orders 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.orders
        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;
```

TIP If you submit PROC IMPORT code that reads a Microsoft Excel .xls, .xlsx, or other non-delimited file, you can issue a RECALL command to get your PROC IMPORT code back. This enables you to modify the code and then resubmit it.

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

The code in this example imports a range named Orders from an Excel file on the SAS 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 the SAS 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

Use the ACCESS and DBLOAD procedures to access data in Microsoft Excel 4 and Excel 5 files. Excel 2007 and later files, and. xlsx, .xlsb, and .xlsm files are not supported. These 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. These procedures have other SAS 6 limitations such as a maximum of 8-byte SAS variable names and a maximum of 200-character value.

This 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*

DBMS= Identifier	Option	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

DBMS= Identifier	Option	Valid Value	Default Value	PROC IMPORT	PROC EXPORT
	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 row of the Microsoft imported Excel 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. The data value `Occupancy Code` becomes the SAS variable name `Occupancy_Code`.

Starting in SAS/ACCESS 9.4, GETNAMES= prefixes an underscore to the data value rather than replacing the value's first character. For example, `2013.Changes` becomes `_2013_Changes`.

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.

VALIDVARNAME= must be V7. It cannot be ANY.

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. 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. This statement is optional.

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.

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';
  DBMS=EXCEL5 REPLACE
RUN;
```

Import and Export Microsoft Excel Files Using the XLS and XLSX Identifiers

The IMPORT and EXPORT procedures use the DBMS=XLS identifier to read and write .xls file formats directly. DBMS=XLS identifier supports Excel versions 5/95, 97, 2000, 2002, and 2003. It is available on UNIX and Microsoft Windows operating environments.

The IMPORT and EXPORT procedures use the DBMS=XLSX identifier to read and write .xlsx file formats directly. The identifier supports Excel 2007, 2010, and later files, as well as .xlsx, .xlsb, or .xlsm files. The DBMS=XLSX specification is the recommended (rather than DBMS=XLS). The DBMS=XLSX identifier is available on UNIX and Microsoft Windows operating environments.

Note: When you open an .xlsx workbook in Excel, all other sheets in the workbook have their formulas automatically recalculated. This includes any formulas that reference a sheet that has been added or has been replaced by `PROC EXPORT DBMS=XLSX`.

This table lists the statements that are available to import data from or export data to an Excel file using the DBMS=XLS and DBMS=XLSX identifiers.

Table 6.8 *Available Statements for Importing and Exporting Excel Files Using DBMS=XLS and DBMS=XLSX*

DBMS= Identifier	Option	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	DATAROW	1....	When GETNAMES=NO: 1, when GETNAMES=YES: 2	Yes	No
	GETNAMES	Yes | No	Yes	Yes	No
	RANGE	name | sheet $ul:lr	First row	Yes	No
	SHEET	Sheet name	First sheet	Yes	Yes

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

Default The last column 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 column names from the data values in the first row of the imported file. If data in the first row 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 column name `Occupancy_Code`.

Starting in SAS/ACCESS 9.4, the behavior for GETNAMES= changes. For .xlsx files, GETNAMES=YES prefixes an underscore to the data value rather than replacing the value's first special character. DBMS=XLSX turns a numeric column name into a character string, whereas DBMS=XLS does not. Column names become A, B, C, and so on.

Table 6.9 *Behavior of GETNAMES=*

GETNAMES=YES	DBMS=XLSX Result	DBMS=XLS Result
Column beginning with a special character or numeric: `2013.Changes`	`_2013_Changes`	`_2013_Changes`
All numeric column: `2014`	`2014`	`A` (if the first column in the worksheet)

TIP If a column name contains all numerics and you need that column to retain its name, you can save the .xls file as an .xlsx file, and then import the .xlsx file.

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

NO specifies that PROC IMPORT is to generate SAS column names as A, B, C, and so on, for both .xlsx and .xls files.

Restrictions PROC IMPORT only

If VALIDVARNAME=ANY is used, GETNAMES= might not prefix an underscore to the data value.

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 .xls or .xlsx 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 as column headings to the first row 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 as column names (or headings) to the first row of the exported file. It writes the first row of SAS data to the second row of the exported data file.

NO specifies that PROC EXPORT is to write the first row of SAS data to the first row of exported data file.

Restrictions Available only for DBMS=XLS for backward compatibility.

PROC EXPORT only.

RANGE=*'range-name' | 'absolute-range';*
subsets a worksheet by identifying the rectangular set of cells to import.

If you omit the RANGE= statement, the IMPORT procedure tries to select an existing worksheet and import the entire worksheet. You might get more columns or missing data than you want. The worksheet selected might not be the one that you wanted. Therefore, it is strongly recommended that you use the RANGE= statement.

The *range-name* is a user-defined table name that represents a range of cells within the worksheet in the Excel file. C2..F12 represents cells within column C, row 2, and column F, row 12. For more detail, see the RANGE= statement under Microsoft Excel files on page 74.

An absolute range identifies the top left cell that begins the range and the bottom right cell that ends the range; it is not necessarily the entire worksheet but it can be. You can name the range with a worksheet name and append a `$`. Hence, the range `SHEET="Invoice$"` refers to data in the whole worksheet. For more information, see the SHEET= statement on page 87.

If GETNAMES=YES is set, the first row of data in the range is used for the generated column names, and the data starts from the second row in the range. If GETNAMES=NO is set, the data starts from the first row and column names are generated by the IMPORT procedure.

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

You can use RANGE= to specify the row number where PROC IMPORT starts to read data. Set the end point to `"0"`, and the code then determines the last row and last column. Specify RANGE="*Sheetname*$A#:0"; where # is the first data row. Thus, `RANGE="sheet1$A3:0";` starts to read the data at row 3. If you use RANGE= for this purpose, do not specify the DATAROW= statement.

Restriction This statement is valid only for PROC IMPORT.

SHEET=*'sheet-name'*

identifies a particular worksheet in an Excel .xls or .xlsx workbook. Specify *sheet-name* to name the sheet as output. If you omit the SHEET= statement, the SAS data set name defines the sheet name and range name in the exported Excel file. If the sheet already exists, it is replaced. You can also use it to add a new sheet to an existing worksheet.

A worksheet is not a table but rather, a container of tables. A worksheet name with `$`, such as `"Invoice$"`, is a range covering all data in the Invoice worksheet. When you delete the range `"Invoice$"`, you delete all the data in the worksheet. The worksheet, as the container, remains. Therefore, the SAS log indicates that the data is deleted but that the worksheet name still exists.

The *sheet-name* can contain up to 31 characters. These examples show how SAS converts sheet names with invalid or special characters.

- If the sheet name in the EXPORT procedure contains a special character (such as a space), SAS converts it to an underscore (_). (See the exceptions below.) For example, `Employee Information` becomes `Employee_Information`.
- If the sheet name contains single quotation marks, enclose the name in double quotation marks to preserve the single quotation marks as part of the sheet name (for example, `SHEET="'My#Test'";`).
- For .xlsx files, certain special characters are not converted to an underscore in a sheet name, including an embedded space (`Employee ID`) and these: `~, !, %, ^, &, (, ), +, {, }`. Sheet names that contain these characters still need to be enclosed in quotation marks to preserve these characters.

- When exporting an .xlsx file, sheet names with trailing spaces are trimmed (for example, `SHEET='&Invoice+ '`).
- When exporting an .xlsx file, the SHEET= statement automatically creates a range by the same name. The range is the entire sheet. The underscore character replaces the special character for both the range and sheet names unless the sheet name is enclosed in quotation marks to preserve the special characters. For example, `SHEET="Invoice(2013)"` creates a new range by the name `"Invoice_2013_"`, and the sheet name is `"Invoice(2013)"` because it was quoted.
- For .xlsx files, if a sheet name with special characters is not enclosed in quotation marks, that sheet will not be the same as the range name. SAS converts the range-name's special characters into underscores and thereby changes its name. For example, an .xlsx sheet name `&Invoice+` is not the same as range name `&Invoice+` because SAS converts the range name into `_Invoice_` .

If both the range name and the sheet name are omitted, the IMPORT procedure tries to select an existing worksheet and import the entire worksheet. You might get more columns or missing data than you want. The sheet selected might not be the one that you wanted. It is strongly recommended that you specify the RANGE= statement to import that table that you want.

STARTCOL= ***start-column***
specifies the first column for data.

Default The first column that contains data.

Restriction Available only for DBMS=XLS for backward compatibility.

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: Import an XLSX File to a CAS Table

Import an .xlsx file with CHAR data types to a CAS table named Caslib.test with VARCHARs.

```
libname mycas cas;

%let sess=mysess;
%let caslib=mycaslibin;
%let table=caslib.test;
%let ext=xlsx;
%let path=path-name;

cas &sess;
```

```
libname caslib cas sessref=&sess;

proc import datafile="&path.&table..&ext"
    out=caslib.test
    dbms=xlsx
    replace;
    getnames=yes;
    sheet="test_date_col";
run;

proc contents data=caslib.test;
run;
```

Figure 6.2 *CONTENTS Procedure Results*

The CONTENTS Procedure

Data Set Name	CASLIB.TEST	**Observations**	18
Member Type	DATA	**Variables**	5
Engine	CAS	**Indexes**	0
Created	10/25/2017 15:19:07	**Observation Length**	56
Last Modified	10/25/2017 15:19:07	**Deleted Observations**	0
Protection		**Compressed**	NO
Data Set Type		**Sorted**	NO
Label			
Data Representation	WINDOWS_64		
Encoding	utf-8 Unicode (UTF-8)		

Engine/Host Dependent Information	
Data Limit	100MB
Caslib	CASUSERHDFS
Scope	Session

Alphabetic List of Variables and Attributes

#	Variable	Type	Len		Max Bytes Used	Format	Label
			Bytes	Chars			
1	Col1	Varchar	.	.	9	$9.	Col1
2	Col2	Varchar	.	.	9	$9.	Col2
3	Col3	Num	8			BEST.	Col3
4	Col4	Num	8			BEST.	Col4
5	Date_col	Num	8			DATE7.	Date_col

Example 2: Export a CAS Table to an XLSX File

This example exports a CAS table to an .xlsx file.

```
libname mycas cas;


data mycas.air;
   set sashelp.air;
run;

proc export data=mycas.air
   dbms=xlsx
   outfile="/userid/pathname/air1"
   replace;
run;
```

Here is what is written to the log.

```
56
 57         libname mycas cas;
 NOTE: Libref MYCAS was successfully assigned as follows:
       Engine:        CAS
       Physical Name: 15ccee55
 58
 59
 60         data mycas.air;
 61            set sashelp.air;
 62         run;

 NOTE: There were 144 observations read from the data set SASHELP.AIR.
 NOTE: The data set MYCAS.AIR has 144 observations and 2 variables.
 NOTE: DATA statement used (Total process time):
       real time           0.02 seconds
       cpu time            0.00 seconds


 63
 64         proc export data=mycas.air
 65            dbms=xlsx
 66            outfile="/userid/pathname/air1"
 67            replace;
 68         run;

 NOTE: The export data set has 144 observations and 2 variables.
 NOTE: "/userid/pathname/air1.xlsx" file was successfully created.
 NOTE: PROCEDURE EXPORT used (Total process time):
       real time           0.09 seconds
       cpu time            0.02 seconds
```

Example 3: Export SAS Data Sets to Excel 2010 Workbook and Replace Sheets

This example exports the SAS data sets, Sdf.invoice and Sdf.orders, to an Excel 2010 workbook with Invoice and Orders as sheet names. The Invoice and Orders sheets already exist, and REPLACE in each PROC EXPORT statement is used to overwrite or replace the data in these sheets. When you use REPLACE at the sheet level, the range includes the entire sheet (and not a subset of cells).

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

PROC EXPORT DATA=sdf.orders
        FILE="&tmpdir.text.xlsx
        DBMS=XLSX REPLACE;
    SHEET='Orders';
RUN;
```

Example 4: Export SAS Data Sets to Excel 2010 Workbook and Add a New Sheet

Starting with SAS/ACCESS 9.4, you can add a new sheet to an existing Excel workbook. Omit REPLACE and add the new sheet name; in this case, `'Invoice_%2013'`.

```
LIBNAME SDF V9 "&sasdir";
PROC EXPORT DATA=SDF.INVOICE
        FILE="&tmpdir.text.xlsx"
        DBMS=XLSX;
    SHEET='Invoice_%2013';
RUN;
```

Example 5: Import Data Using a Range Name

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

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

```
    GETNAMES=YES;
RUN;
```

Example 6: 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;
```

Exporting and Importing Excel Files with the SAS PC Files Server

Specifying DBMS=EXCELCS in the IMPORT and EXPORT procedures enables you to import and export Excel files between a remote Microsoft Windows computer running the PC Files server and a SAS session running on UNIX or another Microsoft Windows computer. The target Windows computer is identified in the IMPORT and EXPORT procedures with the SERVER= statement.

Note: The PC Files server is not included in SAS Viya 3.4 and earlier. Connect to the server using the SAS 9 client.

SERVER=*SAS-PC-Files-Server-name*
specifies the name of the SAS PC Files Server, where *SAS-PC-Files-Server-name* can be either the computer name or the associated IP address. You must bring up the listener on the SAS 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 SAS PC Files Server. The SERVER= statement is required when connecting to the PC Files Server from a UNIX computer. You can omit this option if you are running SAS and the SAS PC Files Server on the same machine. Omitting this option under this condition causes the SAS PC Files Server to start automatically in the background. For information to manage the SAS PC Files Server, see the *SAS PC Files Server: Installation and Configuration Guide*.

Note: The SAS PC Files Server is not included in SAS Viya 3.4 and earlier. Use the SAS 9 client.

Example 1: Export a SAS Data Set on UNIX to an Excel File on Microsoft Windows

This example exports a SAS data set named Customer on a UNIX system to an .xls file on a Windows system. The .xls file is also named Customer. Note that SAS is running on the UNIX operating platform and the .xls file is loaded on Microsoft Windows where the SAS PC Files Server is running. The target Windows system is specified in the SERVER= argument.

```
LIBNAME SDF "&sasdir";

PROC EXPORT DBMS=EXCELCS DATA=SDF.CUSTOMER
OUTFILE="&tmpdir.customer.xls"
 REPLACE;
SERVER=fileserver;
RUN;
```

Example 2: Import an Excel File on Microsoft Windows to a SAS Data Set on UNIX

This example imports data from an .xlsx file named Customer on Windows to a SAS data set of the same name on UNIX. Note that SAS is running on a UNIX platform and the .xls file is located on Microsoft Windows where the SAS PC Files Server is running.

```
LIBNAME SDF "&sasdir";

PROC IMPORT OUT= work.customer
 FILE="&tmpdir.customer.xlsx"
 DBMS=EXCELCS REPLACE;
 SERVER="&server";
RUN;
```

Microsoft Access Database Files

Microsoft Access File Essentials

SAS/ACCESS Interface to PC Files works with Microsoft Access database 97, 2000, 2002, 2003, 2007, 2010, and later files. 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.

Microsoft Access is a desktop relational database system that uses the Microsoft ACE data provider (.accdb file format) or the Microsoft Jet data provider (.mdb file format) to store and retrieve data.

Microsoft Access supports two file formats, .mdb and .accdb.

- Access 2000, 2002, and 2003 share the same .mdb file formats. ACCESS2000, ACCESS2002, and ACCESS2003 are treated as aliases for Access in SAS. For an existing .mdb file, you can specify DBMS=ACCESS on Windows platforms and SAS identifies the version for you.
- Access 2007 and later support .accdb file formats.

This table lists the maximum size limits for .mdb and .accdb files.

Table 6.10 *Microsoft Access Database (.mdb and .accdb) Maximum Size Limits per Method*

File size	2GB /32,768 objects
Number of fields per table	255 units
Number of characters per text field	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 SAS data set name. If a database field contains special characters, SAS converts them to underscores in the corresponding SAS variable name.

Microsoft Access Data Types

This section summarizes field data types that are available in Microsoft Access, their uses, and their storage sizes.

ATTACHMENTS
: specify attachment to images, spreadsheet (worksheet) 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).

CALCULATED
: creates an expression from one or more fields and designates a different result data type from the expression. Not available for the .mdb format.

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 | LONG TEXT
: use for lengthy text and numbers, such as notes or descriptions. Stores up to 32767 bytes. Access 2007 (and later) stores up to 65535 bytes but because the SAS limit is 32767 bytes, text longer than that is truncated.

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 FIELDSIZE property defines the specific Number type.

OLE OBJECT
: use for OLE objects (such as Microsoft Word documents, Microsoft Excel worksheets, 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 | SHORT 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 by default. You can adjust the character limit property up to 4,000. The FIELDSIZE= 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 byte.

  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, these 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 portion and a time portion, the format is in date/time format.
- Number display formats support date, time, or date/time formats.

In SAS software, these 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.

Using the DBMS= Identifiers for Microsoft Access Database Files

Overview

The IMPORT and EXPORT procedures for Access databases require statements to identify the type of data to import and indicate the import method. The procedures generate SAS LIBNAME engine code transparently. DBMS=ACCESS specifies the data to import or export resides in an Access database on a local Windows PC. You can specify Access LIBNAME engine data set options to control advanced processing. DBMS=ACCESSCS specifies the data resides on a Windows machine that is running the SAS PC File Server. Specify DBMS=ACCESSCS from UNIX or another Windows PC. With DBMS=ACCESSCS you can specify PC Files LIBNAME engine options.

See "Example 1: Import a Microsoft Access 2007 Database Table to a SAS Data Set" on page 105 and "Example 2: Export a SAS Data Set to Create a Microsoft Access Database File" on page 105 for additional information.

This table shows the options that are supported for the IMPORT and EXPORT procedures for Microsoft Access database files on Windows. Options are described after the tables.

Table 6.11 *IMPORT and EXPORT Procedure Statements for Microsoft Access Files on Windows*

DBMS= Identifier	Option	Valid Value	Default Value	PROC IMPORT	PROC EXPORT
ACCESS ACCESS97	DATABASE	Complete path and file name for the Microsoft Access database file.		Yes	Yes
	DBDSOPTS	Any valid data set options for the Access LIBNAME engine.		Yes	Yes
	DBPASSWORD	Access database password (used with USER= to access files secured at the database level).		Yes	Yes
	DBSASLABEL	Yes | No	Yes	Yes	No
	MEMOSIZE	1 to 32767	1024	Yes	No

DBMS= Identifier	Option	Valid Value	Default Value	PROC IMPORT	PROC EXPORT
	PASSWORD	Access database password (used with USER= to access files secured at the user level).		Yes	Yes
	SCANMEMO	Yes | No	Yes	Yes	No
	SCANTIME	Yes | No	Yes	Yes	No
	USEDATE	Yes | No	No	Yes	No
	USER	Access database user ID.		Yes	Yes
	DBSYSFILE	Complete path and file name for the Workgroup Administration file.		Yes	Yes

To use the client server method, specify DBMS=ACCESSCS in PROC IMPORT or in PROC EXPORT from SAS running on UNIX or Microsoft Windows operating platforms. DBMS=ACCESSCS indicates that your Access database files reside on the SAS PC Files Server on a remote Windows PC. For information about installing and connecting to the SAS PC Files server, see *SAS PC Files Server: Installation and Configuration Guide*.

For an existing .mdb or .accdb file, you can specify DBMS=ACCESSCS and SAS identifies the version of Microsoft Access for you. You can specify VERSION=97 when using DBMS=ACCESSCS only when you want to export and create a new .mdb file with Access version 97 formats.

Note: When DBMS=ACCESSCS is used, the SAS client cannot open an Access database file with both database password (DBPASSWORD=) protection and user-level security (PASSWORD=) protection. You must choose only one security protection.

This table lists the procedure statements to import or export data to or from an Access database file on the PC server from a UNIX or Microsoft Windows client running SAS. The DBMS= identifier for all options is ACCESSCS.

Table 6.12 *IMPORT and EXPORT Procedure Statements for Access Files When Using SAS PC Files Server*

Option	Valid Value	Default Value	PROC IMPORT	PROC EXPORT
DATABASE	Complete path and file name for the Microsoft Access database file on the PC file server.		Yes	Yes
DBDSOPTS	Any valid data set options for the PC Files engine.		Yes	Yes

Option	Valid Value	Default Value	PROC IMPORT	PROC EXPORT
DBPASSWORD	Access database password for USER= to access data secured at the database level.		Yes	Yes
DBSASLABEL	Yes | No	Yes	Yes	No
DBSYSFILE	Complete path and file name for the Workgroup Administration file.		Yes	Yes
MEMOSIZE	1 to 32767	1024	Yes	No
PASSWORD	Access database password for USER= to access data secured at the user level.		Yes	Yes
PORT	1 to 65535	9621	Yes	Yes
SCANMEMO	Yes | No	Yes	Yes	No
SCANTIME	Yes | No	Yes	Yes	No
SERVER	PC Files Server name.		Yes	Yes
SERVERPASS	PC Files Server password.		Yes	Yes
SERVERUSER	PC Files Server user ID.		Yes	Yes
SERVICE	PC Files Server Service name.		Yes	Yes
SSPI	Yes | No		Yes	Yes
USEDATE	Yes | No	No	Yes	No
USER	Access database user ID		Yes	Yes
VERSION	97 | 2000 | 2002 | 2003 | 2007 | 2010 | and later	2000	Yes	Yes

DATABASE=*database*
specifies the complete path and file name of the Microsoft Access database file on the PC file server. 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 or PC Files LIBNAME engines'*
Enables you to take advantage of useful data set options for the LIBNAME engine such as READBUFF= , INSERTBUFF= , DBTYPE=, DROP= , FIRSTOBS= , and OBS=. Note that DBDSOPTS= option is for advanced users who are familiar with the Access LIBNAME engine when DBMS=ACCESS or the PC Files

LIBNAME engine when DBMS=ACCESSCS. The PC Files engine is not included in SAS Viya 3.4 and earlier because the engine requires the SAS PC Files Server. Use the SAS 9 client to import and export data using the PC Files Server.

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 DBMS=ACCESSCS to import or export data in a Microsoft 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 write data in an Access database, the default value for INSERTBUFF= is 1. To improve performance for writing data, you should set the INSERTBUFF= option to 25 or higher.

For users who want to read MS Access ShortText, use the dbsastype option:

```
proc import datafile = "c:\temp\date.xlsx" out =have replace dbms =excel ;
dbdsopts="dbsastype=(age='numeric' weight='char(20)'
 weight='char(20)')";
run;
```

"DBSASTYPE= Data Set Option"

See "SAS Data Set Options for the LIBNAME Statement for PC Files Engine Data Set Option" on page 271

DBPASSWORD=*'database-file-password'*
enables you to open an Access database .mdb or .accdb file if database-level security is set in the file. This option enables you to open .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. (For user-level security, use PASSWORD=)

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.

Aliases YES (for COMPAT)

NO (for NONE)

Restriction Due to how the Microsoft ACE driver works, no more than 64 characters of column names are written to SAS variable labels.

DBSYSFILE=*complete path and file name 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 bytes, 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 32,767 is truncated when it is imported into SAS.

Alias DBMAX_TEXT

Restriction When SCANMEMO=YES, the length of the longest memo text that the scan finds is used as the SAS variable width. If that value is greater than the value of the MEMOSIZE= option, the smaller value in the MEMOSIZE= option is applied as the SAS variable width.

PASSWORD= *'database-file-password'*
enables you to open an Access database .mdb or .accdb file if user-level security is set in the file. This option enables you to open .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. (For files with database-level security, use DBPASSWORD=)

PORT= *port-number*
specifies the number of the port that is listening on the SAS PC Files Server. The valid value is between 1 and 65535. This port or service name is displayed on the SAS 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 (DBMS=ACCESSCS).

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 driver or the Microsoft Jet driver is used as the SAS variable length.

Reading values for multiple-values fields is not supported. Values are read into the indexes. If SCANMEMO=YES is set, it reads values into the first index only. If SCANMEMO=NO is set, it reads all of the indexes into SAS.

Restrictions SCANMEMO= applies only to MEMO data type columns; it does not apply to TEXT type columns.

When SCANMEMO=YES, if the maximum length of the 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.

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=*SAS-PC-Files-Server-name*

specifies the name of the SAS PC Files Server, where *SAS-PC-Files-Server-name* can be either the computer name or the associated IP address. You must bring up the listener on the SAS 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 For use with the client/server model (DBMS=ACCESSCS).

Requirement Available from SAS 9. Requires the SAS PC Files Server which is not included in SAS Viya 3.4 and earlier.

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

SERVERPASS=*'server-user-password'*
specifies the password for the PC file server user ID specified in SERVERUSER=. If the account has no password, omit this option. Always enclose the value in quotation marks in order to preserve the case of the password.

Alias SERVERPASSWORD | SERVERPW | SERVERPWD

Restriction For use with the client-model (DBMS=ACCESSCS)

Requirement Available only from SAS 9. Requires the SAS PC Files Server which is not included in SAS Viya 3.4 and earlier.

Notes Passwords are generally case sensitive.

Use the PASSWORD= or DBPASSWORD= option for Microsoft Access database passwords.

Example PROC EXPORT example using explicit user name and password for the SAS PC Files Server:

```
PROC EXPORT DATA=SDF.INVOICE
    OUTFILE="&tmpdir.customer.mdb"
    DBMS=ACCESSCS REPLACE
         SERVER=fileserv;
         SERVERUSER='mydomain\myusername';
     SERVERPASS='mypassword';
RUN;
```

SERVERUSER=*'server-user-name'*
specifies a domain and user ID that are valid for the PC running the SAS PC Files Server. Always enclose the value in quotation marks. Otherwise, the backslash can be misinterpreted by the SAS parser.

Alias SERVERUID

Restriction For use with the client server model (DBMS=ACCESSCS)

Requirement Available only from SAS 9. Requires the SAS PC Files Server which is not included in SAS Viya 3.4 and earlier.

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

Use the USER= option for Microsoft Access database User IDs.

Example See the previous example for using PROC EXPORT with explicit user name and password for the SAS PC Files Server:

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 SAS 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 (DBMS=ACCESSCS).

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

Requirement Available only from SAS 9. Requires the SAS PC Files Server which is not included in SAS Viya 3.4 and earlier.

Note This service name needs to be defined on both your local machine and your SAS PC Files Server.

SSPI=YES | NO
enables the SAS PC Files Server to allow Integrated Windows Authentication. This is a mechanism for Windows client and server to exchange credentials.

Default NO

Restrictions Valid only on Windows 64-Bit.

For use with the client server model (DBMS=ACCESSCS).

Requirement Available only from SAS 9. Requires the SAS PC Files Server, which is not available with SAS Viya 3.4 and earlier.

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

USEDATE=YES | NO
specifies whether to assign a DATE. or a DATETIME. format while importing a date/time column from a Microsoft Access Database.

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 and assigns the DATETIME. format for the corresponding date/time column in the Microsoft Access table. If you have time values in your data, specify USEDATE=NO.

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 214.

The SCANTIME= statement option on page 76 in order to assign the appropriate TIME format.

USER=*user-id*
specifies the user account name, if one is required to connect to the Microsoft Access database.

Interaction The DBPASSWORD= option or the PASSWORD= option are typically used with USER=.

VERSION=*file-version*
specifies the version of the file that you want to create. Values include 2010 or later, 2007, 2003, 2002, 2000, and 97. The default value depends on the extension of the file. It is optional to enclose the version in quotation marks.

Restriction For use with the client/server model (DBMS=ACCESSCS).

Requirement Available only with SAS 9. Requires the SAS PC Files Server which is not included with SAS Viya 3.4 and earlier.

Interaction If the file already exists on the SAS PC Files Server, then the statement is ignored.

Example 1: Import a Microsoft Access 2007 Database Table to a SAS Data Set

This code imports an Access database table named Customers in database Demo.accdb to a SAS data set. The Microsoft Access table was saved in version 2007 format.

```
PROC IMPORT OUT=Mydata.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 a Microsoft Access Database File

This code exports a SAS data set named Sdf.employee and creates a new Microsoft Access database file named Test2000.mdb. Note that Test2000.mdb does 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 an Access Database Table to a SAS Data Set 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 a Microsoft Access Database Table on a SAS PC Files Server

This code exports a SAS data set named Sdf.orders to a new Microsoft Access database file named Testpcfs.mdb, located on the SAS 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 WKn Files

WKn Files Essentials

This section introduces Lotus 1-2-3 WK*n* files.

Note: Lotus 1-2-3 is not included with SAS Viya 3.3 and earlier. Windows is supported in SAS Viya 3.4 and later.

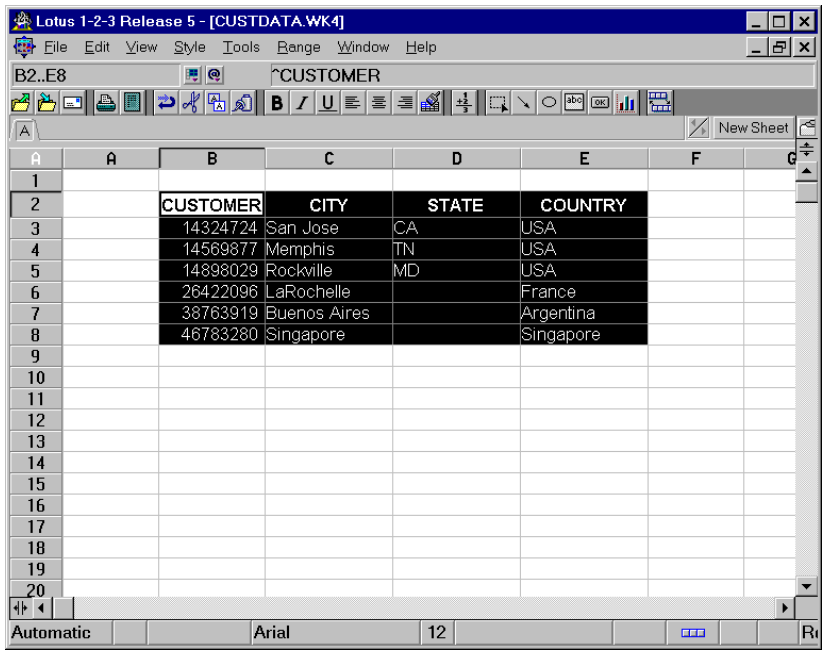
SAS/ACCESS Interface to PC Files works with WK1, WK3, and WK4 (releases 4 and 5) files. These files contain data in the form of Lotus 1-2-3 spreadsheets. They are referred to collectively in this document as WK*n* files, where *n* represents releases 1, 3, or 4. SAS/ACCESS Interface to PC Files does 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.

Figure 6.3 Columns and Rows of Data in a WKn File



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. 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. 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.

WKn 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..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).

A number can have both a date portion 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 WKn Files

The IMPORT and EXPORT procedures for WK*n* files use 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 these.

- not being case sensitive
- a maximum length of 8 bytes of SAS variable name
- maximum length of 200 characters of data values

This 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 6.13 *Import or Export Data from Lotus 1–2–3 Files*

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

Data Source	Syntax	Valid Value	Default Value	PROC IMPORT	PROC EXPORT
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 `Occupancy Code` becomes the SAS variable name `Occupancy_Code`.

Starting in SAS/ACCESS 9.4, GETNAMES= prefixes an underscore to the data value rather than replacing the value's first special character. For example, `2013.Changes` becomes `_2013_Changes`.

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

Restriction Valid only for the IMPORT procedure.

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. '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 Lotus 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 a Lotus WK4 File Data to a SAS Data Set

This example imports data from a Lotus .wk4 file named Invoice.wk4 into SAS data set named Mydata.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=Mydata.invoice
            DATAFILE="&wkndir.Invoice.wk4"
            DBMS=WK4 REPLACE;
   SHEET='A';
   RANGE='A1..D12';
   GETNAMES=YES;
RUN;
```

Example 3: Export Data to a Lotus 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.orders without the variable SPECINST.

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

```
RUN;
```

Example 4: Import Data from a Lotus WK1 File to 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

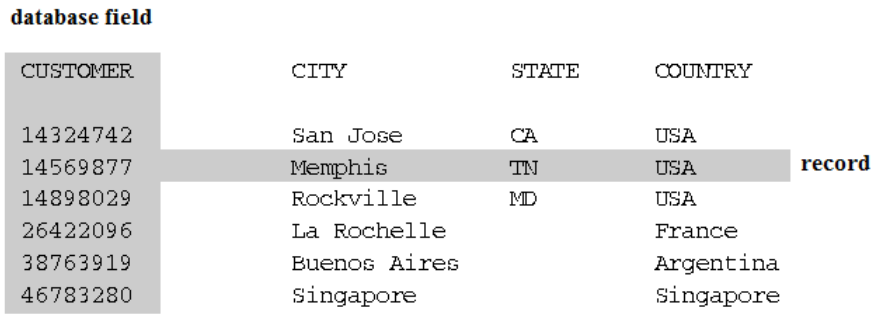
dBase DBF Files Essentials

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 118.

dBase .dbf files are in a file format that dBase creates. dBase is a relational database management system for PC systems. dBase .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 display shows four database fields from Customer.dbf and highlights a database field and a record.

Figure 6.4 Database Field and Record



DBF Data Types

Every field in a dBase .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.

Note: A database field name can have no more than 10 characters. For this reason, SAS variable names are truncated to 10 characters when they are exported to dBase.

CHARACTER (*N*)
specifies a field for character string data. The maximum length of *N* is 255 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 6.14 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 that 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*. Here is an example:`02/20/95` for February 20, 1995.

Dates in .dbf files can be subtracted from one another. The result is the number of days between the two dates. A number (of days) can also be added to a date. The result is 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 reads 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 IMPORTand EXPORT procedures use dBase .dbf file formats to access data in .dbf files on UNIX and Microsoft Windows operating environments.

PROC IMPORT imports data from .dbf files in versions 3, 4, and 5 formats. PROC EXPORT exports data to .dbf files with version 5 formats. SAS variable names are truncated to 10 characters when they are exported to dBase because of the dBase field name 10-character limit.

See "Example 1: Export Data to a dBase DBF File from a SAS Data Set" on page 116 for additional information.

Table 6.15 *IMPORT and EXPORT Procedure Statements for dBase DBF Files*

DBMS = Identifier	Syntax	Valid Values	Default Value	PROC IMPORT	PROC EXPORT
dBase .dbf files	DBENCODING	Encoding-value	Current SAS session encoding	Yes	Yes
	GETDELETED	Yes | No	Yes	Yes	No

DBENCODING = *12–char 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 dBase DBF File from a SAS Data Set

This example exports data to a dBase .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 dBase DBF File into a SAS Data Set

This example imports data from a .dbf file named Invoice.dbf into SAS data set named Mytest.test5. The data is imported without the **DELETE flag** field.

```
PROC IMPORT OUT=MYTEST.TEST5
            DATAFILE="&dbfdir.Invoice.dbf"
            DBMS=DBF REPLACE;
   GETDEL=NO;
RUN;
```

Example 3: Export Data to a dBase 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 dBase 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 DBMS=DBFMEMO identifier to import dBase Memo fields into SAS, 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 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 file name 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 dBase 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 Mytest.test.

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

JMP Files

JMP File Essentials

A JMP file is a file format that is created using JMP software and has a .jmp file extension. JMP is an interactive statistics package from SAS that is available for Microsoft Windows and Macintosh. For more information about JMP concepts or terms, see the JMP documentation that is packaged with your system or that is located on their website, http://www.jmp.com/support/help/.

A JMP file contains data that is organized in a tabular format of fields and records. Each field can contain one type of data, and each record can hold one data value for each field.

JMP variable names can be up to 255 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. Therefore, you can access JMP files without a license for SAS/ACCESS Interface to PC Files. You can use these SAS language elements with JMP files: a LIBNAME statement, the IMPORT and EXPORT procedures, and the Import and Export wizards.

JMP files cannot be displayed or accessed using the SAS DATASETS procedure.

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. Use the CAS engine to connect to CAS and import the .jmp table to a CAS table where JMP CHAR data types are stored as CAS VARCHAR. When exporting VARCHAR data types from a CAS

table to a .jmp table, the VARCHAR data types are stored in JMP as CHAR. Characters can be letters, digits, spaces, or special characters. For more information about SAS CAS, see *SAS Cloud Analytic Services: User's Guide*.

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

`meta=libref.member`

Starting in SAS 9.4, the META data type has been replaced by support for extended attributes. Extended attributes are customized metadata for your JMP and SAS files. They are user-defined characteristics that you associate with a JMP 7 and later file or variable, or with a SAS 9.4 and later data set or variable.

If a file has extended attributes, they are automatically transferred to the new file when that file is imported or exported. For example, when exporting a SAS data set to JMP, PROC EXPORT looks for extended attributes on the SAS data set. If the attributes exist, the procedure uses the attributes to build the new .jmp file. When importing a .jmp file with extended attributes, the attributes are automatically attached to the new SAS data set.

META can remain in programs, yet it generates a NOTE in the log stating that META has been replaced by extended attributes. The META data type is ignored.

For more information about using extended attributes, see *Using JMP* and the *Base SAS Procedures Guide*.

Note: Extended attributes are not supported for CAS tables.

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
is generated by JMP and is used to store several row-level characteristics. It is transferred to and from SAS as an 8-byte floating point number but at this time, it is not meant for user manipulation.

If the JMP file contains row state information, PROC IMPORT stores this information as a new variable with the name **_rowstate_**. If PROC EXPORT sees a column named **_rowstate_**, it converts it back into row state information in the output JMP file.

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.

LOCAL
specifies the DATE or DATETIME format for your computer's regional settings. When you are reading data, the LOCAL values are mapped to a SAS number and scaled to the local time and date. The JMP local date and datetime display format maps to the appropriate SAS LOCAL display format. When you are writing data, the SAS output format for the numeric variable is checked to determine whether it is a local date or datetime format. If so, the SAS numeric value is scaled to a JMP date value with the appropriate 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 SAS JMP Files Data

SAS imports data from JMP files that are saved with JMP 7 and later formats, and it exports data to JMP files with JMP 7 and later formats. SAS also supports importing and exporting JMP files with more than 32,767 variables.

Note: Starting in SAS 9.4, files in JMP 3 through JMP 6 formats are no longer supported.

The IMPORT and EXPORT procedures provide two identifiers for accessing SAS JMP files.

JMP File Formats (DBMS= JMP)
The DBMS=JMP specification uses JMP file formats (JMP 7 and later) to access data in .jmp files on UNIX and Microsoft Windows operating platforms.

SAS PC Files Server (DBMS=PCFS)
The DBMS=PCFS specification uses the client/server model to access data in JMP files on Microsoft Windows from UNIX, or Microsoft Windows operating environments. This method requires running the SAS PC Files Server on Microsoft Windows. The PC Files Server is not included in SAS Viya 3.4 and earlier. You can connect to it using the V9 client.

IMPORT Procedure and EXPORT Procedure Supported Syntax

FMTLIB=*libref.format-catalog*
When exporting a SAS data set to a .jmp file, if the FMTLIB= statement is present, the specified format catalog is used to convert SAS user-defined formats to .jmp value labels. When importing a .jmp file, FMTLIB= saves value labels to the specified SAS format catalog.

META=*libref.member-data-set;*
The META statement is no longer supported for importing or exporting a JMP file, and it is ignored. Instead, extended attributes are automatically used. If a file has extended attributes, they are automatically transferred to the new file when that file is imported or exported. META can remain in programs. However, doing so generates a NOTE in the log and the statement is ignored.

Alias METADATA

Example 1: Import a JMP File to a CAS Table

This example imports a JMP file named Demo.jmp to a CAS table named Caslib.test.

```
libname mycas cas;

%let sess=mysess;
%let caslib=test;
%let table=demo;
%let ext=jmp;
%let path=path-name;

cas &sess;
libname caslib cas sessref=&sess;

proc import datafile="&path.&table..&ext";
    out=caslib.test
    dbms=jmp
    replace;
 run;

proc contents data=caslib.test;
run;
```

Figure 6.5 CONTENTS Procedure Results

The CONTENTS Procedure

Data Set Name	CASLIB.TEST	Observations	40
Member Type	DATA	Variables	7
Engine	CAS	Indexes	0
Created	10/25/2017 15:30:23	Observation Length	64
Last Modified	10/25/2017 15:30:23	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	WINDOWS_64		
Encoding	utf-8 Unicode (UTF-8)		

Engine/Host Dependent Information	
Data Limit	100MB
Caslib	CASUSERHDFS
Scope	Session

Alphabetic List of Variables and Attributes

#	Variable	Type	Len		Max Bytes Used	Format	Label
			Bytes	Chars			
2	Abrasion	Num	8				Abrasion
3	Date	Num	8			MMDDYY10.	Date
6	Date_Time	Num	8			DATETIME16.	Date/Time
5	Shift	Varchar	.	.	1	$8.	Shift
4	Time	Num	8			TIME.	Time
1	_rowstate_	Num	8				_rowstate_
7	time2	Num	8			TIME.	time2

Example 2: Export a CAS Table to a JMP File

This example exports a CAS table named Mycas.Air to a JMP file named Air1.jmp.

```
libname mycas cas;

data mycas.air;
   set sashelp.air;
run;
```

```
proc export data=mycas.air
   dbms=jmp
   outfile="/pathname/air1"
   replace;
run;
```

Here is what is written to the log.

```
58
59         libname mycas cas;
NOTE: Libref MYCAS was successfully assigned as follows:
      Engine:        CAS
      Physical Name: 76b575c6
60
61
62         data mycas.air;
63            set sashelp.air;
64         run;

NOTE: There were 144 observations read from the data set SASHELP.AIR.
NOTE: The data set MYCAS.AIR has 144 observations and 2 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds


65
66         proc export data=mycas.air
67            dbms=jmp
68            outfile="/pathname/air1"
69            replace;
70         run;

NOTE: The export data set has 144 observations and 2 variables.
NOTE: "/pathname/air1.jmp" file was successfully created.
NOTE: PROCEDURE EXPORT used (Total process time):
      real time           0.02 seconds
      cpu time            0.01 seconds
```

Example 3: 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=sdr.customer
            FILE="&tmpdir.customer.jmp"
            DBMS=JMP REPLACE;
RUN;
```

Example 4: Export a SAS Data Set on UNIX to a JMP File Using DBMS-=PCFS

This example runs SAS on UNIX and requires access through the SAS PC Files Server. The example exports a SAS data set named Sdf.customer to a JMP file named Customer.jmp.

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

Example 5: Import a JMP File to a SAS Data Set

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="&jmpdir.customer.jmp"
            DBMS=JMP REPLACE;
RUN;
```

Example 6: 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 the SAS PC Files Server is running.

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

Example 7: 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 Mydata.customer. Note that SAS is running on a UNIX platform and the .jmp file is located on Microsoft Windows where the SAS PC Files Server is running.

```
PROC IMPORT OUT= MYDATA.CUSTOMER
             FILE="&csjmpdir.customer.jmp"
             DBMS=PCFS REPLACE;
     SERVER=”&server”;
RUN;
```

Paradox DB File Formats

Paradox File Essentials

All versions of Paradox under 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 file name 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, Mydata.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

SAS/ACCESS supports SPSS version 18 and earlier under Microsoft Windows. SPSS files have a .sav file extension. SPSS files that have short variable names are exported. See “SPSS SAV Files” on page 127 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`.

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.

When reading an SPSS file with PROC IMPORT, SAS generates a user-defined format for each SPSS variable that has a value label. SAS can generate up to 4089 formats; more than 4089 formats are ignored.

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.

The EXPORT procedure supports user-defined formats, which are created using the FORMAT procedure. However, user-defined formats with a list of values in a range do not export to SPSS. This example shows creating a format called MYFMT that defines a range of values:

```
proc format;
value myfmt 0 = "no"
1-9 = "yes";
```

Note that the format definition contains a single range, `0 = "no"` and also a list of values in a range, `1-9 = "yes"`. If you use PROC EXPORT to export a data set that uses the user-defined MYFMT, the 0 value is formatted and appears as “no” in the SPSS table, but the nine “yes” values are left unformatted.

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

PROC IMPORT with the V9 engine supports SPSS numeric and character field types that map directly to SAS numeric and character fields.

If you use the CAS engine to import from a CAS table in memory, SPSS CHAR data types are imported as VARCHAR data types in a CAS table.

This list shows other SPSS data types and how the IMPORT procedure converts them to SAS formats.

- SPSS character variables that are longer than 255 bytes are combined into a single variable of up to a length of 32,767 bytes.
- Datetime, Dtime convert to a SAS datetime value and SAS datetime format, respectively.
- Time 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 *mmddyy* format.
- Moyr converts to a SAS date value in the *mmddyy* format.

PROC EXPORT converts data into SPSS variable types.

If you use the CAS engine to export character data, CAS VARCHAR types are converted to SPSS CHAR.

Date and Time values are stored in SPSS numeric fields that are 8–byte floating-point numbers, with these format conversions:

- SAS character variables up to a length of 32,767 bytes can be exported.
- COMMA converts to SPSS format type Comma.
- DOLLAR converts to SPSS format type Dollar.
- DOLLAR converts to SPSS format type Dollar.
- DATE converts to SPSS format type Date.
- MMDDYY converts to SPSS format Adate.
- MMMYY converts to SPSS format Moyr.
- DATETIME converts to SPSS format Dtime.
- TIME converts to SPSS format Time

Importing and Exporting SPSS Files

SPSS Files (DBMS=SPSS)
: The IMPORT and EXPORT procedures support two DBMS= identifiers for SPSS .sav files. The DBMS=SPSS specification uses SPSS file formats to access data in SPSS .sav files on UNIX and Microsoft Windows operating platforms.

SAS PC Files Server (DBMS=PCFS)
: The DBMS=PCFS specification uses the client/server model to access data in SPSS .sav files on Microsoft Windows from UNIX or Microsoft Windows 64-bit operating environments. This method requires running the SAS PC Files Server on Microsoft Windows. The PC Files Server is not included in SAS Viya 3.4 and earlier. Connect to it using the SAS V9 client.

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: Import an SPSS SAV File to a CAS Table

This example imports a .sav file with CHAR data types to a CAS table named Caslib.test with VARCHAR data types.

```
libname mycas cas;

%let sess=mysess;
%let caslib=test;
%let table=demo;
%let ext=sav;
%let path=path-name;

cas &sess;
libname caslib cas sessref=&sess;

proc import datafile="&path.&table..&ext";
    out=caslib.test
    dbms=sav
    replace;
 run;

proc contents data=caslib.test;
run;
```

Figure 6.6 CONTENTS Procedure Results

The CONTENTS Procedure

Data Set Name	CASLIB.TEST	Observations	23
Member Type	DATA	Variables	6
Engine	CAS	Indexes	0
Created	10/25/2017 15:32:25	Observation Length	88
Last Modified	10/25/2017 15:32:25	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	WINDOWS_64		
Encoding	utf-8 Unicode (UTF-8)		

Engine/Host Dependent Information	
Data Limit	100MB
Caslib	CASUSERHDFS
Scope	Session

Alphabetic List of Variables and Attributes

#	Variable	Type	Len		Max Bytes Used	Format	Label
			Bytes	Chars			
1	field1	Varchar	.	.	70	$79.	field1
2	field2	Varchar	.	.	14	$14.	field2
3	field3	Varchar	.	.	12	$12.	field3
4	field4	Varchar	.	.	14	$18.	field4
5	field5	Varchar	.	.	9	$12.	field5
6	field6	Num	8			F12.	field6

Example 2: Export a CAS Table to an SPSS SAV File

This example exports a CAS table named Mycas.Air, to the SPSS file named Air1.sav.

```
libname mycas cas;


data mycas.air;
   set sashelp.air;
```

```
run;

proc export data=mycas.air
   dbms=sav
   outfile="/pathname/air1"
   replace;
run;
```

Here is what written to the log.

```
58
 59         libname mycas cas;
 NOTE: Libref MYCAS was successfully assigned as follows:
       Engine:        CAS
       Physical Name: 76b575c6
 60
 61
 62         data mycas.air;
 63            set sashelp.air;
 64         run;

 NOTE: There were 144 observations read from the data set SASHELP.AIR.
 NOTE: The data set MYCAS.AIR has 144 observations and 2 variables.
 NOTE: DATA statement used (Total process time):
       real time           0.01 seconds
       cpu time            0.01 seconds


 65
 66         proc export data=mycas.air
 67            dbms=sav
 68            outfile="/pathname/air1"
 69            replace;
 70         run;

 NOTE: The export data set has 144 observations and 2 variables.
 NOTE: "/pathname/air1.sav" file was successfully created.
 NOTE: PROCEDURE EXPORT used (Total process time):
       real time           0.02 seconds
       cpu time            0.02 seconds
```

Example 3: 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 4: 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 Mydata.customer.

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

Example 5: Import Data from an SPSS SAV File and Apply FMTLIB= Statement

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 6: 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 the SAS PC Files Server is running.

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

Example 7: 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 Mydata.customer. Note that SAS is running on a UNIX platform. The SPSS file is located on Microsoft Windows where the SAS PC Files Server is running.

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

Stata DTA Files

Stata DTA Files Essentials

SAS/ACCESS supports Stata 15 and earlier versions under Microsoft Windows. Stata files have a .dta file extension.

See "Example 3: Export a SAS Data Set to a Stata DTA File on a Local System" on page 139 for additional information.

FILES
: Import of Stata 15 and earlier data under Microsoft Windows and UNIX is supported. For example, more than 32,767 variables, greater than 2,000,000,000 rows (IMPORT procedure only), and UTF-8 are supported. Export of Stata 15 and earlier is supported. You can now export more than 32,767 variables. UTF-8 is also supported.

MISSING VALUES
: Stata supports missing values. SAS missing values are written as Stata missing values. By default, SAS prints a missing numeric value as a single period (.) and a missing character value as a blank space. When you export a SAS data set to a Stata file, a single missing value (.) is written to the Stata file.

  Starting in SAS/ACCESS 9.4, when you import a Stata file, SAS enables you to read multiple Stata missing values and map them to multiple SAS special missing values, `.a–.z` or as a single dot (`.`) (up to 27 missing values).

VARIABLE NAMES

When using importing, Stata variable names can be up to 32 bytes 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 6.16 *Stata Reserved Words*

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

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.

Stata Data Types

Stata supports character field types as strings with a maximum length. When importing string data, SAS can handle strings up to a maximum length of 32,767 bytes when using the V9 engine because SAS CHAR data types have a limitation of 32,767 bytes. When importing string data into a CAS table using the CAS engine, strings from a Stata file of up to 536,870,911 characters are imported as VARCHAR.

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

Stata date variables become SAS numerics with a date format.

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

Importing and Exporting Stata Data Files

Note: Stata file encoding must be consistent with SAS session encoding.

Stata .dta Files (DBMS=DTA)
: The DBMS=DTA specification accesses data in Stata .dta files on UNIX and Microsoft Windows operating environments.

> **TIP** If you get an error message when trying to import a Stata file, save the file to an older version of Stata and then import the file into SAS. For example, you could save a Stata 13 file as a Stata 12 file and then import it.

SAS PC Files Server (DBMS=PCFS)
: The DBMS=PCFS specification uses the client/server model to access data in Stata files on Microsoft Windows from UNIX or Microsoft Windows operating environments. This method requires running the SAS PC Files Server on Microsoft Windows. The SAS PC Files Server is not available in SAS Viya 3.4 and earlier. Connect using the SAS 9 client.

IMPORT and EXPORT Procedures Supported Syntax

FMTLIB=*libref.format-catalog*. When importing a Stata file, if the FMTLIB= statement is present, 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: Import a Stata DTA File to a CAS Table

This example imports a Stata .dta file named Demo.dta to a CAS table named Caslib.test.

```
libname mycas cas;

%let sess=mysess;
%let caslib=test;
%let table=demo;
%let ext=dta;
%let path=path-name;

cas &sess;
libname caslib cas sessref=&sess;

proc import datafile="&path.&table..&ext";
    out=caslib.test
    dbms=dta
    replace;
 run;

proc contents data=caslib.test;
run;
```

Figure 6.7 CONTENTS Procedure Results

The CONTENTS Procedure

Data Set Name	CASLIB.TEST	Observations	23
Member Type	DATA	Variables	6
Engine	CAS	Indexes	0
Created	10/25/2017 15:27:57	Observation Length	88
Last Modified	10/25/2017 15:27:57	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	WINDOWS_64		
Encoding	utf-8 Unicode (UTF-8)		

Engine/Host Dependent Information	
Data Limit	100MB
Caslib	CASUSERHDFS
Scope	Session

Alphabetic List of Variables and Attributes

#	Variable	Type	Len		Max Bytes Used	Format	Label
			Bytes	Chars			
1	field1	Varchar	.	.	80	$80.	field1
2	field2	Varchar	.	.	14	$14.	field2
3	field3	Varchar	.	.	12	$12.	field3
4	field4	Varchar	.	.	18	$18.	field4
5	field5	Varchar	.	.	12	$12.	field5
6	field6	Num	8				field6

Example 2: Export a CAS Table to a Stata DTA File

This example exports a CAS table named Mycas.Air to the Stata file named Air1.dta.

```
libname mycas cas;

data mycas.air;
```

```
   set sashelp.air;
run;

proc export data=mycas.air
   dbms=dta
   outfile="/pathname/air1"
   replace;
run;
```

Here is what is written to the log.

```
58
59         libname mycas cas;
NOTE: Libref MYCAS was successfully assigned as follows:
      Engine:        CAS
      Physical Name: 76b575c6
60
61
62         data mycas.air;
63            set sashelp.air;
64         run;

NOTE: There were 144 observations read from the data set SASHELP.AIR.
NOTE: The data set MYCAS.AIR has 144 observations and 2 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds


65
66         proc export data=mycas.air
67            dbms=dta
68            outfile="/pathname/air1"
69            replace;
70         run;

NOTE: The export data set has 144 observations and 2 variables.
NOTE: "/pathname/air1.dta" file was successfully created.
NOTE: PROCEDURE EXPORT used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds
```

Example 3: Export a SAS Data Set to a Stata DTA 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=DTA REPLACE;
RUN;
```

Example 4: Import a Stata DTA File on a Local System to a SAS Data Set

This example imports Stata file, Customer.dta on a local system to a SAS data set Mydata.customer.dta.

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

Example 5: 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 the SAS PC Files Server is running.

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

Example 6: Import Data from a Stata DTA 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 Mycust.customer. Note that SAS is running on a UNIX platform. The Stata file is located on Microsoft Windows where SAS PC Files Server is running.

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

PART 3

SAS/ACCESS Interface to PC Files LIBNAME Engines

Chapter 7
SAS/ACCESS LIBNAME Engines 143

7

SAS/ACCESS LIBNAME Engines

Overview of SAS LIBNAME Engines for Microsoft Excel and Microsoft Access 143
Software Requirements 144
Comparing SAS LIBNAME Engines for PC Files Data 145
Assigning a Libref Interactively 147
Sorting PC Files Data 149
Using SAS Functions with PC Files Data 149
Macro Variables 149
System Options 151
Dictionary 151
SASTRACE= System Option 151
SASTRACELOC= System Option 154

Overview of SAS LIBNAME Engines for Microsoft Excel and Microsoft Access

SAS/ACCESS Interface to PC File Formats includes SAS LIBNAME engines that enable you to read and write Microsoft Excel workbooks and Microsoft Access database files. SAS LIBNAME engines are a way to filter data instead of using SAS procedures to filter data.

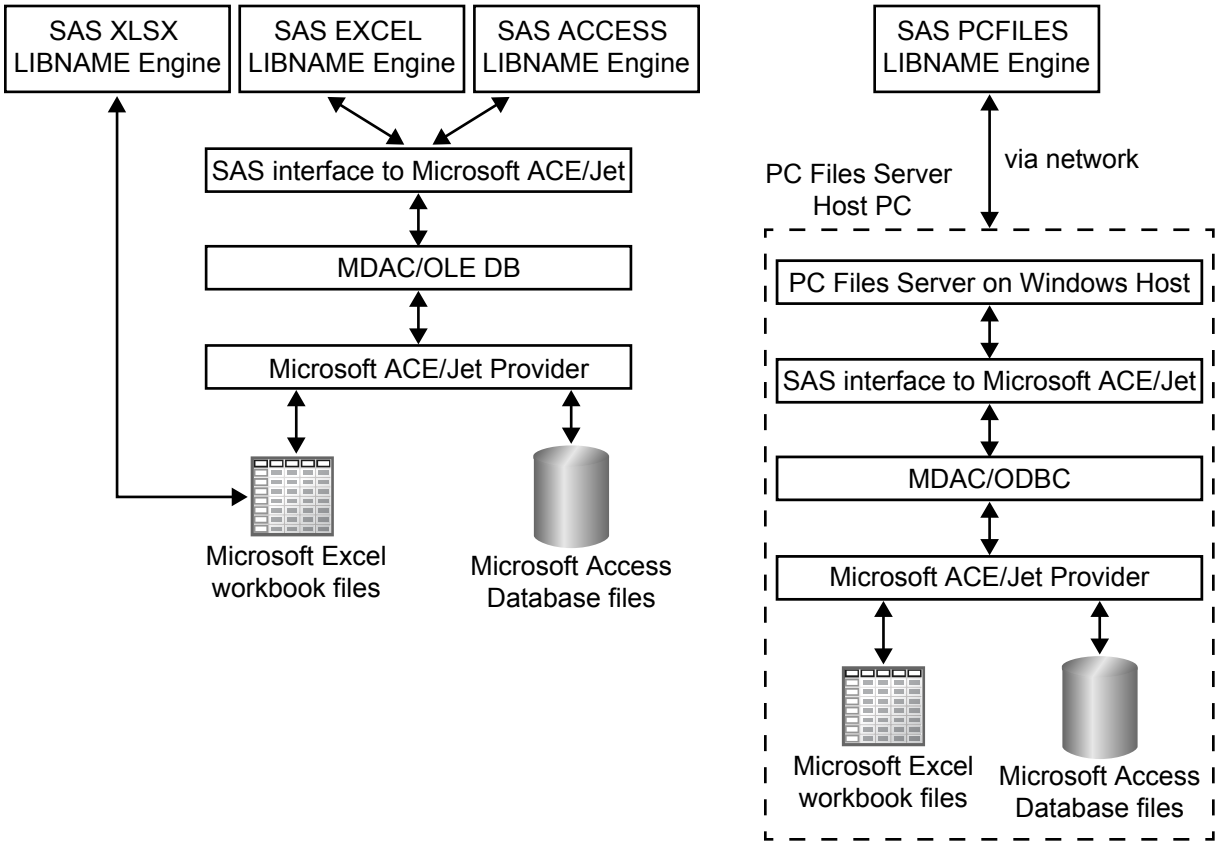
The SAS LIBNAME engines are called using a SAS language element called the LIBNAME statement, which enables you to assign a libref. With the LIBNAME statement you can reference workbooks and databases directly in a DATA step or SAS procedure. You can also read from and write to a Microsoft Access object or Microsoft Excel object. Microsoft Excel and Microsoft Access data can be imported to a SAS data set and exported from a SAS data set to Excel or Access databases.

There are three SAS LIBNAME engines that connect to Microsoft Excel: the Excel engine, PC Files engine, and XLSX engine. For more information, see

- LIBNAME Statement: Excel and Access Engines on page 159
- LIBNAME Statement: XLSX Engine on page 241
- LIBNAME Statement: PC Files Engine on page 255

This figure shows the available LIBNAME engines for Excel and Access databases and the software that is used with it.

Figure 7.1 *The Available LIBNAME Engines for Excel and Access Databases*



Software Requirements

The Excel, Access, and PC Files engines share most of the code that calls OLE DB or ODBC 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 later) provider software. By default, SAS software installs and checks the required Microsoft software.

The XLSX LIBNAME engine bypasses the calls to Jet or ACE APIs, and directly accesses the Microsoft Excel 2007 or later files.

For the most current system requirements for SAS/ACCESS, review the host-specific system requirements from the SAS 9.4 Install Center.

- Documentation for SAS 9.4 Installation on Microsoft Windows and Windows 64 Install Center
- Documentation for SAS 9.4 Installation on UNIX The SAS PC File Server is required in order to use the PC Files engine.

It enables you to read and write Excel or Access data stored on Windows, UNIX, or Linux systems. For more information about the SAS PC File Server, see "Quick Start for Installation" in *SAS PC Files Server: Installation and Configuration Guide*.

Comparing SAS LIBNAME Engines for PC Files Data

Each engine's features vary depending on how you plan to access the data and where it resides. This table compares the three LIBNAME engines that provide access to Microsoft Excel files.

The following table shows each engine's features:

Table 7.1 *Comparison of the SAS LIBNAME Engines for Microsoft Excel*

Feature	Excel LIBNAME Engine	PC Files LIBNAME Engine	XLSX LIBNAME Engine
Host support	Microsoft Windows	Windows, UNIX	Windows, UNIX
Requires the SAS PC Files Server	No	Yes	No
Requires Microsoft Access Database Engine (ACE)	Yes	Yes	No
Supports SAS LIBNAME options	Yes	Yes	Limited
Supports SAS data set options	Yes	Yes	Limited
Supports SAS SQL procedure and pass-through	Yes	Yes	No

Feature	Excel LIBNAME Engine	PC Files LIBNAME Engine	XLSX LIBNAME Engine
Reads data support for file types	xlsx, .xlsb, .xlsm, .x ls	xlsx, .xlsb, .xlsm, .x ls	.xlsx
Creates data support for file types (new table)	.xlsx, .xlsb, .xls	xlsx, .xlsb, .xls	xlsx
Updates data support for file types	.xlsx, .xlsb, .xls	.xlsx, .xlsb, .xls	xlsx

Note: Although this table focuses on the LIBNAME engines' support for Microsoft Excel, the PC Files LIBNAME engine and Access LIBNAME engine also support Microsoft Access databases. The Access LIBNAME engine lets you read and write Microsoft Access data, and it has the same capabilities as the Excel LIBNAME engine.

The **Excel LIBNAME engine** enables you to read and write Microsoft Excel data from a wide range of Excel releases, using several file formats, as shown in the table. The Excel engine runs only on Microsoft Windows, and it enables you to use the SQL procedure and its pass-through facility. You can use Excel data as if it were a SAS data set in a DATA step and in SAS procedures.

In the following example, you define a LIBNAME statement to access a Microsoft Excel 2007 or later workbook, which contains a number of tables, including Invoice. After you have defined the LIBNAME statement, you can access and use the Excel data as if it were a SAS data set:

```
libname xdb excel "\\mydriver\pcfdata\Invoice.xlsx";
```

The **Access LIBNAME engine** enables you to read and write Microsoft Access database files. The Access engine runs only on Windows. Like the Excel engine, the Access engine lets you use Access database data as if it were a SAS data set in a DATA step or in SAS procedures. You can use the SQL procedure pass-through facility with Microsoft Access data and includes capabilities similar to how the Excel engine uses the SQL pass-through facility.

In the following example, you use the Access LIBNAME statement to access a database named Demo. After you have defined the LIBNAME statement, you can read and write Access database data as if it were a SAS data set:

```
Libname mydb access "\\mydriver\accdata\demo.accdb"    ;
```

The **PC Files LIBNAME engine** also enables you to read and write Microsoft Excel data from a wide range of Excel releases, using several file formats, as shown in the table. The PC Files engine connects to Excel data through the SAS PC Files Server, a 32-bit or 64-bit application that resides on Windows. A SAS session that is running on UNIX or another Windows machine can access Excel data that is stored

on the PC Files Server on Windows. The PC Files Server is not included in SAS Viya 3.4 and earlier but you can connect to it using the SAS 9 client.

Like the Excel LIBNAME engine, the PC Files engine enables you to use the SQL procedure and its pass-through facility. You can use the Excel data as if it were a SAS data set in a DATA step and in SAS procedures. In this example, the Microsoft Excel data is stored on the same Windows machine as the SAS PC Files Server. The PC Files LIBNAME engine accesses the Excel data from SAS, which is running on a UNIX machine or a local Windows machine. When you use the SAS PC Files Server on a different machine than SAS is running on, you must specify the SERVER= and PORT= options to identify that machine:

```
libname q4unx pcfiles path="c:\users\elgesns\sales\Q4workbook.xlsx"
server="D12345" port=9621;
```

You specify the PC Files LIBNAME engine only when you need to use the SAS PC Files Server, such as when you plan to connect from SAS running on UNIX. For example, if you are running 64-bit SAS on a 64-bit Windows machine, you can use the Excel LIBNAME engine to access Microsoft Excel data directly; you do not need the SAS PC Files Server. For more information, see " PC Files LIBNAME Engine and the PC Files Server" on page 254.

The **XLSX LIBNAME engine** enables you to read and write Excel files between UNIX and Microsoft Windows. SAS can be running on Windows or UNIX. The Excel files can reside on Windows or UNIX as well. The XLSX engine is available starting with the second maintenance release of SAS 9.4.

Unlike the Excel and PC Files LIBNAME engines, the XLSX LIBNAME engine does not go through the Microsoft Access Database Engine (ACE) to access the Excel data. Instead, you can access the files directly. Doing so might give you greater performance, yet you cannot apply most SAS options that would enable SAS to process the Excel data being read.

For example, you can access an .xlsx file stored on a UNIX file system. Then you can write a DATA step or use PROC SQL to rename columns or apply labels, handle null values, apply formats, and so on.

```
libname myhr XLSX "/usr/wolfen/benefits/HRworkbook.xlsx";
proc sql;
select lname label="Last Name", empid, start_date format=date9.
from myhr.perm_employees
where state="NC";
quit;
```

Assigning a Libref Interactively

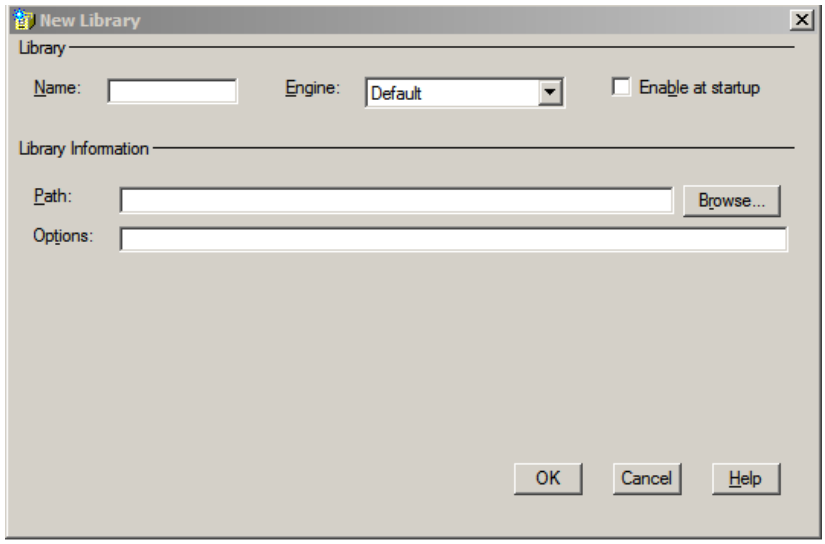
An easy way to associate a libref with PC files data is to use the New Library window.

From the SAS session command, submit the `dmlibassign` command to open the New Library window.

Or, you can access the New Library window by right-clicking the libraries icon in the Explorer window and selecting **New**.

Note: Use of a LIBNAME statement locks the file, and you must use a LIBNAME statement with the CLEAR option to unlock the file.

Figure 7.2 *New Library Window*



- **Name** can be 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 SAS V9 engine is assigned.
- **Enable at startup** select to assign the specified libref 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.
- Library Information: represents the SAS/ACCESS connection options, and it varies according to the SAS/ACCESS engine that you specify. Enter the appropriate information for your PC file format.
- **Path** specifies the full path of the file associated with the libref.
- **OK**: click this button to assign the libref, or click **Cancel** to exit the window without assigning a libref.

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 file name assigned for the libref. For a normal SAS libref, it returns the path name 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 clear a libref that refers to PC files data, you omit this argument. For full details about how to use SAS functions, see *SAS Functions and CALL Routines: Reference*.

Macro Variables

The automatic macro variables SYSDBMSG, SYSDBRC, SQLXMSG, and SQLXRC are portable, but the SAS/ACCESS engine and the Microsoft ACE driver 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:

Note: Because the XLSX LIBNAME engine does not rely on Microsoft ACE software, these macro variables are not supported for this engine.

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

SYSDBRC
contains ACE provider-specific error codes that are generated when you use SAS/ACCESS software to access 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 SAS SQL pass-through facility generates return codes and error messages that are available to you through these SAS macro variables:

SQLXMSG
contains ACE provider-specific error messages.

SQLXRC
contains ACE provider-specific error codes.

SQLXMSG and SQLXRC can be used only with the pass-through facility. See "Return Codes" on page 293 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

When using the VIEWTABLE window to display table names that include special characters, you need to specify the SAS system option VALIDMEMNAME=EXTEND. For more information about using the VIEWTABLE window, see "Working with VIEWTABLE" in *SAS Programmer's Guide: Essentials* .

SASTRACE= and SASTRACELOC= are SAS system options that have specific SAS/ACCESS applications.

Dictionary

SASTRACE= System Option

Generates trace information from an Excel, Access, or PC Files engine.

Valid in: OPTIONS statement, configuration file, SAS invocation

Default: none

Restriction: SASTRACE= is not valid for the XLSX LIBNAME engine

See: "SASTRACELOC= System Option" on page 154

Syntax

SASTRACE=',,,d' | ' ,,d,' | ' d,' | ',,,s' | OFF

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.

OFF
specifies to turn tracing off.

Details

This option's behavior 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 the "SASTRACELOC= System Option" on page 154.
- 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, the SAS LIBNAME engines for Microsoft Access and Microsoft Excel 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= System Option

Writes SASTRACE= option information to a specified location.

Valid in: OPTIONS statement, configuration file, SAS invocation

Default: stdout

Restriction: SASTRACELOC= is not valid for the XLSX LIBNAME engine

See: "Overview to SAS Language Elements That Use Encoding Values" in *SAS National Language Support (NLS): Reference Guide*.

Syntax

SASTRACELOC=stdout | SASLOG | FILE *'path-and-file-name'*;

Details

This option enables you to specify where to put the trace messages that the SASTRACE= system option generates. By default, the output goes to the default output location (stdout) for your operating environment. Specify SASTRACELOC=SASLOG to send the output to the SAS log.

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,d' sastraceloc=file 'c:\work\trace.log';
```

PART 4

LIBNAME Statements for Excel and Access Engines on Microsoft Windows

Chapter 8
LIBNAME Statement for Excel and Access Engines on Microsoft Windows 159

Chapter 9
LIBNAME Statement Options for Excel and Access Engines 171

Chapter 10
Data Set Options for Excel and Access LIBNAME Statement 179

Chapter 11
LIBNAME Statement: File-Specific Reference for Excel and Access Engines 201

Chapter 12
SQL Pass-Through Facility for Excel and Access Engines 217

8

LIBNAME Statement for Excel and Access Engines on Microsoft Windows

Dictionary 159
LIBNAME Statement: Access and Excel Engines 159

Dictionary

LIBNAME Statement: Access and Excel Engines

Associates a SAS libref with a Microsoft Excel workbook or Microsoft Access database.

Valid in: Anywhere

Restrictions: Excel and Access engines support up to 255 columns because of how ACE and Jet drivers work.

Microsoft Windows is required.

Syntax

LIBNAME < *libref*> <*engine-name*> <*'physical-path-and-file-name.ext'*> <*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 Microsoft Excel spreadsheet or Microsoft Access database is stored. The association between a libref and a SAS library lasts only for the duration of the SAS session or until you change the libref or discontinue it with another LIBNAME statement.

engine-name
is the SAS/ACCESS engine name, either Excel or Access. The SAS/ACCESS LIBNAME statement associates a libref with a SAS/ACCESS engine that supports connections to a particular PC file type. SAS/ACCESS supports multiple LIBNAME engines, two of which are described in this chapter:

EXCEL for Microsoft Excel 5, 95, 97, 2000, 2002, 2003, 2007, 2010, and later.

ACCESS for Microsoft Access 97, 2000, 2002, 2003, 2007, 2010, and later.

Note The engine name is optional if the *physical-path-file-name.ext* is specified. The file extension provides enough information for SAS.

Examples This example illustrates two options for using LIBNAME statement with physical file name, 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 .xls.

```
LIBNAME xdb EXCEL PATH='C:\PCFData\Demo.xls';
```

'physical-path-file-name.ext'
is the *physical-path-and-file-name.ext* of the data source.

- Microsoft Excel data source extensions include: .xlsx, .xls, .xlsb, .xlsm.
- Microsoft Access data source extensions include: .mdb and .accdb.

Note Providing the *physical-path-and-file-name.ext* sets the NOPROMPT ENGINE option.

See "Connection Details" on page 164 "Details" on page 164

Example Physical Path and File Name Omitting Engine Name:

```
LIBNAME xdb 'C:\PCFData\Demo.xlsx';
```

```
LIBNAME adb 'C:\PCFData\Demo.accdb';
```

Example "Example: Using the Excel LIBNAME Engine with the SQL Procedure" on page 165

SAS/ACCESS-engine-connection-options
provide connection options for the Excel or Access engine 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 164

SAS/ACCESS LIBNAME-options
define how SAS interacts with your data source, providing enhanced control of how 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.

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;
```

Connection Options

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

INIT=*connection-string*
used for OLE DB, 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 file name or other connection options, such as PATH and UDL.

Note This is used rarely, specifically in SAS 9.2 and later. The Microsoft ACE engine is 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 later formats.

Default ACE

PATH=*data-source full-path and file-name with extension*
specifies the full path and file name 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 file name.

Alias DATASRC | DS

Restriction Use PATH= only when the physical file name is not specified in the LIBNAME statement

Requirement Use of this option requires the engine name to be specified. The file extensions .mdb or .accdb for Microsoft Access and .xls or .xlsx 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 file name 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 file name for a successful connection.

NOPROMPT prompting is not available.

PROMPT enables prompting for connection information for the data source.

UDL enables browsing so that you can select an existing data link file (.udl).

UDL=*path-for-udl-file*
specifies the path and file name 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 file name 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.

VERSION=2010 and later | 2007 | 2003 | 2002 | 2000 | 97 | 95 | 5
sets the version for a new Excel workbook or an Access database. It is optional to enclose the version in quotation marks.

Alias VER

Default 97 for .xls files; 2007 and later for .xlsb and .xlsx files.

Notes For Excel 2007 and later, files have the same .xlsx file format. Excel 2003, 2000, and 97 share the same .xls file format, though different from that of .xlsx. Excel 5 and 95 share a different file format.

You do not need to specify this option for an existing Excel file.

Options for Access LIBNAME Statement Only

These options are for only the SAS LIBNAME statement for the Access engine.

DBPASSWORD=*database-file-password*
lets you access database files with database-level security, which can be defined instead of user-level security.

Alias DBPWD | DBPW

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 only.

PASSWORD=*user-password*
specifies a password required by Access database for files that are secured at the user level.

Alias PWD | PW | PASS | PASSWORD

Restriction Microsoft Access database only.

Note Passwords are case sensitive.

USER=*user-id*
specifies a user account name, if one is required to connect to the Microsoft Access database. If you have user-level security set on your file, you need to

use the USER= and PASSWORD= options to access your file. If you have database-level security, use USER= and DBPASSWORD= option.

Alias UID

Note Use the SERVERUSER= option to connect to the SAS PC server.

Details

When 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 mypclib 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 mypclib 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';
```

Example: Using the Excel LIBNAME Engine with the SQL Procedure

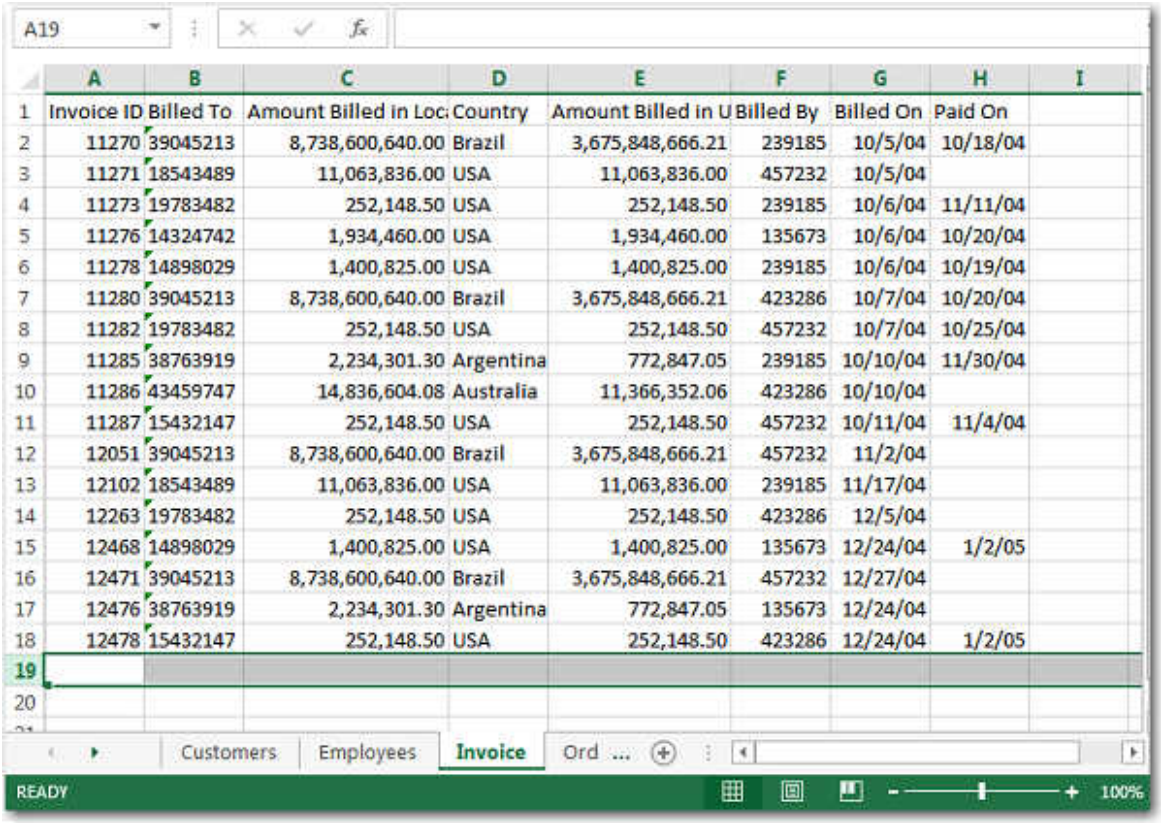
In this example, the SAS/ACCESS LIBNAME statement connects to the Microsoft Excel workbook, Demo, which includes a number of ranges (or tables). By specifying the SAS libref `xdb` with the table names, you can read the Excel data into SAS as if they were data sets. The comments in the code give more details about the steps taken:

```
   /*  Associate the libref xdb with the .xlsx workbook */
   libname xdb excel "\\mymachine\pcfdata\demo.xlsx";

   /* Filter and print the Excel table, Invoice.xlsx, as a SAS  */
   /* data set. When xdb.Invoice is read in the subsequent code,  */
   /* its data is a subset of the Invoice.xlsx table.  */

   proc print data=xdb.Invoice;
      where paid_on is not missing;
   run;

   /* Query the (filtered) joining of the two tables, Invoice and
   Customers, */
   /* from the Excel file. */
```

```
proc sql;
   select Invoice_ID, Name, Customers.Country,
Amount_Billed_In_US_Dollars,
       First_Ordered_Date format=mmddyy10., Billed_On format=mmddyy10.
   from xdb.Invoice, xdb.Customers
   where (Invoice.Billed_To=Customers.Customer_ID);
quit;
```

The Invoice.xlsx and Customers.xlsx files are read into the SQL procedure; the .xlsx data is accessed using the Excel LIBNAME engine.

Output 8.1 *Invoice.xlsx Worksheet (Before filtering)*



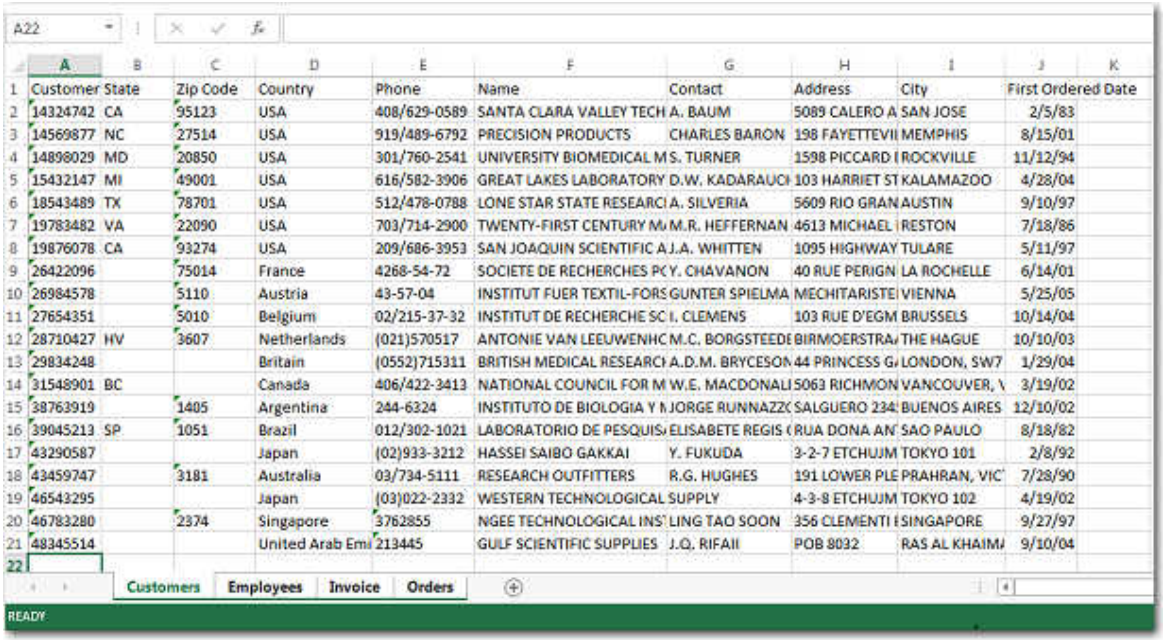
A19

	A	B	C	D	E	F	G	H	I
1	Invoice ID	Billed To	Amount Billed in Loc	Country	Amount Billed in U	Billed By	Billed On	Paid On	
2	11270	39045213	8,738,600,640.00	Brazil	3,675,848,666.21	239185	10/5/04	10/18/04	
3	11271	18543489	11,063,836.00	USA	11,063,836.00	457232	10/5/04		
4	11273	19783482	252,148.50	USA	252,148.50	239185	10/6/04	11/11/04	
5	11276	14324742	1,934,460.00	USA	1,934,460.00	135673	10/6/04	10/20/04	
6	11278	14898029	1,400,825.00	USA	1,400,825.00	239185	10/6/04	10/19/04	
7	11280	39045213	8,738,600,640.00	Brazil	3,675,848,666.21	423286	10/7/04	10/20/04	
8	11282	19783482	252,148.50	USA	252,148.50	457232	10/7/04	10/25/04	
9	11285	38763919	2,234,301.30	Argentina	772,847.05	239185	10/10/04	11/30/04	
10	11286	43459747	14,836,604.08	Australia	11,366,352.06	423286	10/10/04		
11	11287	15432147	252,148.50	USA	252,148.50	457232	10/11/04	11/4/04	
12	12051	39045213	8,738,600,640.00	Brazil	3,675,848,666.21	457232	11/2/04		
13	12102	18543489	11,063,836.00	USA	11,063,836.00	239185	11/17/04		
14	12263	19783482	252,148.50	USA	252,148.50	423286	12/5/04		
15	12468	14898029	1,400,825.00	USA	1,400,825.00	135673	12/24/04	1/2/05	
16	12471	39045213	8,738,600,640.00	Brazil	3,675,848,666.21	457232	12/27/04		
17	12476	38763919	2,234,301.30	Argentina	772,847.05	135673	12/24/04		
18	12478	15432147	252,148.50	USA	252,148.50	423286	12/24/04	1/2/05	
19									
20									

Customers | Employees | Invoice | Ord ...

READY 100%

Output 8.2 *Customers.xlsx Worksheet*



	A	B	C	D	E	F	G	H	I	J	K
1	Customer	State	Zip Code	Country	Phone	Name	Contact	Address	City	First Ordered	Date
2	14324742	CA	95123	USA	408/629-0589	SANTA CLARA VALLEY TECH	A. BAUM	5089 CALERO A	SAN JOSE	2/5/83	
3	14569877	NC	27514	USA	919/489-6792	PRECISION PRODUCTS	CHARLES BARON	198 FAYETTEVI	MEMPHIS	8/15/01	
4	14898029	MD	20850	USA	301/760-2541	UNIVERSITY BIOMEDICAL M	S. TURNER	1598 PICCARD I	ROCKVILLE	11/12/94	
5	15432147	MI	49001	USA	616/582-3906	GREAT LAKES LABORATORY	D.W. KADARAUCI	103 HARRIET ST	KALAMAZOO	4/28/04	
6	18543489	TX	78701	USA	512/478-0788	LONE STAR STATE RESEARCI	A. SILVERIA	5609 RIO GRAN	AUSTIN	9/10/97	
7	19783482	VA	22090	USA	703/714-2900	TWENTY-FIRST CENTURY M/	M.R. HEFFERNAN	4613 MICHAEL I	RESTON	7/18/86	
8	19876078	CA	93274	USA	209/686-3953	SAN JOAQUIN SCIENTIFIC A	J.A. WHITTEN	1095 HIGHWAY	TULARE	5/11/97	
9	26422096		75014	France	4268-54-72	SOCIETE DE RECHERCHES PC	Y. CHAVANON	40 RUE PERIGN	LA ROCHELLE	6/14/01	
10	26984578		5110	Austria	43-57-04	INSTITUT FUER TEXTIL-FORS	GUNTER SPIELMA	MECHITARISTE	VIENNA	5/25/05	
11	27654351		5010	Belgium	02/215-37-32	INSTITUT DE RECHERCHE SC	I. CLEMENS	103 RUE D'EGM	BRUSSELS	10/14/04	
12	28710427	HV	3607	Netherlands	(021)570517	ANTONIE VAN LEEUWENHC	M.C. BORGSTEEDE	BIRMOERSTRA/	THE HAGUE	10/10/03	
13	29834248			Britain	(0552)715311	BRITISH MEDICAL RESEARCI	A.D.M. BRYCESON	44 PRINCESS G/	LONDON, SW7	1/29/04	
14	31548901	BC		Canada	406/422-3413	NATIONAL COUNCIL FOR M	W.E. MACDONALI	5063 RICHMON	VANCOUVER, \	3/19/02	
15	38763919		1405	Argentina	244-6324	INSTITUTO DE BIOLOGIA Y N	JORGE RUNNAZZ(	SALGUERO 234	BUENOS AIRES	12/10/02	
16	39045213	SP	1051	Brazil	012/302-1021	LABORATORIO DE PESQUIS/	ELISABETE REGIS (	RUA DONA AN	SAO PAULO	8/18/82	
17	43290587			Japan	(02)933-3212	HASSEI SAIBO GAKKAI	Y. FUKUDA	3-2-7 ETCHUJM	TOKYO 101	2/8/92	
18	43459747		3181	Australia	03/734-5111	RESEARCH OUTFITTERS	R.G. HUGHES	191 LOWER PLE	PRAHRAN, VIC	7/28/90	
19	46543295			Japan	(03)022-2332	WESTERN TECHNOLOGICAL	SUPPLY	4-3-8 ETCHUJM	TOKYO 102	4/19/02	
20	46783280		2374	Singapore	3762855	NGEE TECHNOLOGICAL INS	LING TAO SOON	356 CLEMENTI I	SINGAPORE	9/27/97	
21	48345514			United Arab Emi	213445	GULF SCIENTIFIC SUPPLIES	J.Q. RIFAII	POB 8032	RAS AL KHAIM/	9/10/04	
22											

Customers | Employees | Invoice | Orders

READY

This display shows the SAS data set, XDB.INVOICE, which was filtered (by Paid_On) and read into SAS from Invoice.xlsx.

Obs	Invoice_ID	Billed_To	Amount_Billed_in_Local_Currency	Country	Amount_Billed_in_US_Dollars	Billed_By	Billed_On	Paid_On
1	11270	39045213	8738600640.0	Brazil	3675848666.21	239185	05OCT2004	18OCT2004
2	11273	19783482	252148.5	USA	252148.50	239185	06OCT2004	11NOV2004
3	11276	14324742	1934460.0	USA	1934460.00	135673	08OCT2004	20OCT2004
4	11278	14898029	1400825.0	USA	1400825.00	239185	08OCT2004	19OCT2004
5	11280	39045213	8738600640.0	Brazil	3675848666.21	423286	07OCT2004	20OCT2004
6	11282	19783482	252148.5	USA	252148.50	457232	07OCT2004	25OCT2004
7	11285	38763919	2234301.3	Argentina	772847.05	239185	10OCT2004	30NOV2004
8	11287	15432147	252148.5	USA	252148.50	457232	11OCT2004	04NOV2004
9	12468	14898029	1400825.0	USA	1400825.00	135673	24DEC2004	02JAN2005
10	12478	15432147	252148.5	USA	252148.50	423286	24DEC2004	02JAN2005

SAS Log for LIBNAME Statement and SQL Queries

```
...
193  libname xdb excel "\\mymachine\pcfdata\demo.xlsx";
NOTE: Libref XDB was successfully assigned as follows:
      Engine:        EXCEL
      Physical Name:
      \\mymachine\pcfdata\demo.xlsx
195
196
197  /* Filter and print the Excel worksheet, Invoice.xlsx, as a SAS data set. */
198  /* When xdb.Invoice is read in the subsequent code, its data is a subset */
199  /* of the Invoice.xlsx worksheet.  */
200
201  proc print data=xdb.Invoice;
202     where paid_on is not missing;
203  run;

NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.03 seconds
      cpu time            0.03 seconds


204
205  /* Query the (filtered) xdb.Invoice data set and the Customers.xlsx file as a */
206  /* SAS data set. The two are joined and further filtered in the PROC SQL query
*/
207  /* and formats are applied to the columns. */
208
209  proc sql;
210        select Invoice_ID, Name, Customers.Country, Amount_Billed_In_US_Dollars,
211            First_Ordered_Date format=mmddyy10., Billed_On format=mmddyy10.
212       from xdb.Invoice, xdb.Customers
213       where (Invoice.Billed_To=Customers.Customer_ID) ;
214  quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.04 seconds
      cpu time            0.04 seconds
```

This display shows the results of the PROC SQL query.

Output 8.3 *Query Results from Joining and Filtering xdb.Invoice and xdb.Customers*

Invoice ID	Name	Country	Amount Billed in US Dollars	First Ordered Date	Billed On
11276	SANTA CLARA VALLEY TECHNOLOGY SPECIALISTS	USA	1934460	02/05/1983	10/06/2004
11278	UNIVERSITY BIOMEDICAL MATERIALS	USA	1400825	11/12/1994	10/06/2004
12468	UNIVERSITY BIOMEDICAL MATERIALS	USA	1400825	11/12/1994	12/24/2004
11287	GREAT LAKES LABORATORY EQUIPMENT MANUFACTURERS	USA	252148.5	04/28/2004	10/11/2004
12478	GREAT LAKES LABORATORY EQUIPMENT MANUFACTURERS	USA	252148.5	04/28/2004	12/24/2004
11271	LONE STAR STATE RESEARCH SUPPLIERS	USA	11063836	09/10/1997	10/05/2004
12102	LONE STAR STATE RESEARCH SUPPLIERS	USA	11063836	09/10/1997	11/17/2004
11273	TWENTY-FIRST CENTURY MATERIALS	USA	252148.5	07/18/1998	10/06/2004
12263	TWENTY-FIRST CENTURY MATERIALS	USA	252148.5	07/18/1998	12/05/2004
11282	TWENTY-FIRST CENTURY MATERIALS	USA	252148.5	07/18/1998	10/07/2004
11285	INSTITUTO DE BIOLOGIA Y MEDICINA NUCLEAR	Argentina	772847.1	12/10/2002	10/10/2004
12476	INSTITUTO DE BIOLOGIA Y MEDICINA NUCLEAR	Argentina	772847.1	12/10/2002	12/24/2004
11270	LABORATORIO DE PESQUISAS VETERINARIAS DESIDERIO FINAMOR	Brazil	3.6758E9	08/18/1982	10/05/2004
12471	LABORATORIO DE PESQUISAS VETERINARIAS DESIDERIO FINAMOR	Brazil	3.6758E9	08/18/1982	12/27/2004
12051	LABORATORIO DE PESQUISAS VETERINARIAS DESIDERIO FINAMOR	Brazil	3.6758E9	08/18/1982	11/02/2004
11280	LABORATORIO DE PESQUISAS VETERINARIAS DESIDERIO FINAMOR	Brazil	3.6758E9	08/18/1982	10/07/2004
11286	RESEARCH OUTFITTERS	Australia	11366352	07/28/1990	10/10/2004

See Also

"LIBNAME Statement" on page 201

9

LIBNAME Statement Options for Excel and Access Engines

Dictionary 171
LIBNAME Statement: Options for Excel and Access Engines 171

Dictionary

LIBNAME Statement: Options for Excel and Access Engines

provides additional control over how SAS processes PC files data.

Restriction: The LIBNAME statement options in this chapter do not apply to the SAS XLSX engine.

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.

See: "LIBNAME Statement: Access and Excel Engines" on page 159

Syntax

LIBNAME <*libref*> *engine-name* <*'physical-path and file name.ext'*> <*SAS/ACCESS engine-connection-options*>

<*SAS/ACCESS LIBNAME-options*>;

Summary of Optional Arguments

ACCESS=READONLY
AUTOCOMMIT=YES | NO
CURSOR_TYPE=KEYSET_DRIVEN | STATIC
DBCOMMIT=*number-of-rows*
DBENCODING= *12–char SAS encoding-value*
DBGEN_NAME=DBMS | SAS
DBMAX_TEXT=*integer between 1 and 32,767*
DBNULLKEYS=YES | NO
DBSASLABEL=COMPAT | NONE
DEFER=NO | YES
DIRECT_SQL=YES | NO | NONE | *specific-functionality*
INSERT_SQL=YES | NO
INSERTBUFF=*number-of-rows*
MSENGINE=ACE | JET
PREPARE=YES | NO
READBUFF=*number-of-rows*
SCANTEXT=YES | NO
SCANTIME=YES | NO | ANY
SPOOL=YES | NO
STRINGDATES=YES | NO
UNICODE=YES | NO
USEDATE=YES | NO

Optional Arguments

ACCESS=READONLY
indicates that tables and views can be read but not updated; specifically, the Excel or Access LIBNAME connection is read-only.

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.

CURSOR_TYPE=KEYSET_DRIVEN | STATIC
specifies the cursor type for read-only cursors and for cursors to be updated. If you do not set CURSOR_TYPE= , the ACE 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

DBCOMMIT=*number-of-rows*

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

Defaults 1,000 [inserting]

0 [updating; commit occurs when data set or procedure completes]

Note If you specify both DBCOMMIT= and ERRLIMIT= options, and these options collide during processing, DBCOMMIT= is issued first and ERRLIMIT= is issued second. Because the DBCOMMIT= option is issued before the ERRLIMIT= option, the DBCOMMIT= option overrides the ERRLIMIT= option in this situation.

DBENCODING= *12–char SAS encoding-value*

specifies the encoding character set to use in Microsoft Excel or Access databases.

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.

DBGEN_NAME=DBMS | SAS

specifies that the data source columns are renamed and the format used for the names.

DBMS specifies the following:

- 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, in bytes, for a character string. Character strings longer than 32,767 are truncated. This option applies only 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 do the following:

- 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 ACE 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 of Access Database files. 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 writes only 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 applies only to the **MEMO data type** field; it does not apply to the **TEXT type** field. Text is defined as less than 256 characters long.

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 ACE provider is set as the SAS variable length. If the length that is returned from the Microsoft ACE 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 rowset 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 rowset 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 enables some character data, such as DBCS, to be returned correctly. This setting works the same as setting DBENCODING='UTF-16'. If SAS is run in DBCS or UTF8, the UNICODE= option defaults to YES.

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

Defaults YES for Microsoft Excel workbooks.

NO for Microsoft Access databases.

10

Data Set Options for Excel and Access LIBNAME Statement

Overview of SAS Data Set Options for the Excel and Access LIBNAME Statement 180

Dictionary 180
AUTOCOMMIT= Data Set Option 180
CURSOR_TYPE= Data Set Option 181
DBCOMMIT= Data Set Option 182
DBCONDITION= Data Set Option 183
DBCREATE_TABLE_OPTS= Data Set Option 184
DBENCODING= Data Set Option 184
DBFORCE= Data Set Option 185
DBGEN_NAME= Data Set Option 186
DBKEY= Data Set Option 187
DBLABEL= Data Set Option 187
DBMAX_TEXT= Data Set Option 188
DBNULL= Data Set Option 189
DBNULLKEYS= Data Set Option 190
DBSASLABEL= Data Set Option 191
DBSASTYPE= Data Set Option 192
DBTYPE= Data Set Option 192
ERRLIMIT= Data Set Option 193
INSERT_SQL= Data Set Option 194
INSERTBUFF= Data Set Option 195
NULLCHAR= Data Set Option 196
NULLCHARVAL= Data Set Option 197
READBUFF= Data Set Option 197
SASDATEFMT= Data Set Option 198

Overview of SAS Data Set Options for the Excel and Access LIBNAME Statement

You specify SAS/ACCESS data set options on a SAS data set when accessing PC files data using the LIBNAME statement. For more information, see "LIBNAME Statement: Access and Excel Engines" on page 159.

A data set option applies only to the SAS 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=, DBLABEL=, 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: The XLSX LIBNAME engine supports the DBLABEL= data set option. For more information, see "DBLABEL= Data Set Option" on page 187.

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= Data Set Option

Determines whether the engine commits (saves) updates as soon as you submit them.

Valid in: DATA and PROC steps

Default: LIBNAME option setting

Restriction: Valid only with Microsoft Access database files

Syntax

AUTOCOMMIT=YES | NO

Syntax Description

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

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

CURSOR_TYPE= Data Set Option

Specifies the cursor type for read-only cursors and for cursors to be updated.

Valid in: DATA and PROC steps

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 ACE provider uses a default. The OLE DB properties applied to an open rowset are as follows:

Table 10.1 *OLE DB Properties Applied to an Open Rowset*

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.

DBCOMMIT= Data Set Option

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

Valid in: DATA and PROC steps

Default: LIBNAME option default: 1000

Restriction: Valid only with Microsoft Access database files.

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, the Access engine 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  mymdb.dept(DBCOMMIT=10);
 SET mysas.staff;
RUN;
```

DBCONDITION= Data Set Option

Specifies criteria for subsetting and ordering data.

Valid in: DATA and PROC steps

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 DBKEY= option is ignored when you use DBCONDITION.

DBCREATE_TABLE_OPTS= Data Set Option

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.

DBENCODING= Data Set Option

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.

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= Data Set Option

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= Data Set Option

Specifies whether to rename columns automatically when they contain disallowed characters.

Valid in: DATA and PROC steps

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.

DBKEY= Data Set Option

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

Default: none

Interaction: affects DBNULLKEYS= and DBCONDITION=

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= Data Set Option

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 256 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= Data Set Option

Determines the length, in bytes, 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.

DBNULL= Data Set Option

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

Restriction: Supported only for Microsoft Access databases

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.

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(DBNULLl=(_ALL_=no jobcode=YES));
 SET mydblib.employees;
RUN;
```

DBNULLKEYS= Data Set Option

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

Valid in: DATA and PROC steps

Default: LIBNAME setting

Interaction: affected by DBKEY=

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= data set 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 = ?)
```

DBSASLABEL= Data Set Option

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

Valid in: DATA and PROC steps

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= Data Set Option

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

Valid in: DATA and PROC steps

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= Data Set Option

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= Data Set Option

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

Valid in: DATA and PROC steps

Default: 1

Restriction: valid only with Microsoft Access database files.

Interaction: DBCOMMIT= option overrides ERRLIMIT= in certain situations

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 10th error. The MyDBLib libref was assigned in a prior LIBNAME statement.

```
DATA mydblib.employee3 (ERRLIMIT=10);
   SET mydblib.employees;
   WHERE salary>40000;
RUN;
```

INSERT_SQL= Data Set Option

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.

INSERTBUFF= Data Set Option

Specifies the number of rows in a single insert.

Valid in: DATA and PROC steps

Default: LIBNAME option setting

Restriction: valid only with Microsoft Access database files.

Syntax

INSERTBUFF=*number-of-rows*

Syntax Description

number-of-rows
specifies the number of rows to insert in the Access database. 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 even when multiple inserts are performed.

Note: The SAS 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.

NULLCHAR= Data Set Option

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, NULLCHAR= determines what is inserted when NULL values are not allowed.

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

NULLCHARVAL= Data Set Option

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

Interaction: affected by NULLCHAR= option; affects DBFORCE= option

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.

READBUFF= Data Set Option

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

Valid in: DATA and PROC

Default: LIBNAME option setting

Restriction: valid only with Microsoft Access database files

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.

SASDATEFMT= Data Set Option

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 SASDATEFMT= option enables you to convert date values from a SAS date format to different SAS date format.

Use SASDATEFMT= 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.

Note: For non-English date types, SAS automatically converts the data to the SAS type of NUMBER. The SASDATEFMT= option does not currently handle these date types. 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.

11

LIBNAME Statement: File-Specific Reference for Excel and Access Engines

Microsoft Excel Workbook Files **201**
LIBNAME Statement 201
Connection Options 202
Data Types Conversion 205
Processing Date and Time Values between SAS and Microsoft Excel 208
Microsoft Access Database Files **209**
Statements 209
Connection Options 209
Data Types Conversion 210
Processing Date and Time Values between SAS and Microsoft Access 214

Microsoft Excel Workbook Files

LIBNAME Statement

By default, the Excel 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 update data and append data in an Excel file, set the LIBNAME option SCANTEXT=NO on page 177.

The SAS PROC SQL pass-through facility connects to a Microsoft Excel file in READ/WRITE mode. Therefore, you can read, write, and update data by passing SQL command statements.

CAUTION

Due to the use of the Microsoft Jet ODBC provider and the Microsoft ACE ODBC provider, the SAS engines for Microsoft Excel have limited Update and Delete capabilities. There might be other unsolved issues. Therefore, you should avoid using the Update and Delete features. It is recommended that you back up your Excel files before you try to use any Update functions.

Connection Options

You can use these connection options 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 providers handle this option; values can be re-set by using the Windows registry.

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 providers, 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.

Setting in the Windows registry 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 11.1 *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 or later 32-bit	**HKEY_LOCAL_MACHINE ⇨ Software ⇨ Wow6432Node ⇨ Microsoft ⇨ Office ⇨ 14.0 ⇨ Access Connectivity Engine ⇨ Engines ⇨ Excel**
Windows 7 or X64 system	Office 2013 32-bit	**HKEY_LOCAL_MACHINE ⇨ Software ⇨ Wow6432Node ⇨ Microsoft ⇨ Office ⇨ 15.0 ⇨ Access Connectivity Engine ⇨ Engines ⇨ Excel**
Windows 7 or X64 system	Office 2010 or later 64-bit	**HKEY_LOCAL_MACHINE ⇨ Software ⇨ Microsoft ⇨ Office ⇨ 14.0 ⇨ Access Connectivity Engine ⇨ Engines ⇨ Excel**

Environment	Office Version or Engine	Registry Key
Windows 7 or X64 system	Office 2013 64-bit	**HKEY_LOCAL_MACHINE ⇨ Software ⇨ Microsoft ⇨ Office ⇨ 15.0 ⇨ Access Connectivity Engine ⇨ Engines ⇨ Excel**
Windows	ACE Engine	**HKEY_LOCAL_MACHINE ⇨ Software ⇨ Microsoft ⇨ Office ⇨ 12.0 ⇨ Access Connectivity Engine ⇨ Engines ⇨ Excel**
Windows	Jet Engine (when using MSENGINE=JET only)	**HKEY_LOCAL_MACHINE ⇨ Software ⇨ Microsoft ⇨ Jet ⇨ 4.0 ⇨ Engines ⇨ Excel**

Table 11.2 *Registry Settings for the MIXED Option*

TypeGuessRows	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. CAUTION Changing the TypeGuessRows value could cause a scan to fail if you set it the value higher than 16384. Changes also affect 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 therefore recommended that you set the value to 0.
ImportMixedTypes	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. For the MIXED=YES option to work correctly, you should 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 ACE provider always assigns character type for columns with data of mixed data types and converts numeric data to character data.

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 ACE engine. Changing settings such as TypeGuessRows for Microsoft ACE engine in the Windows registry affects all software that uses Microsoft ACE 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 ACE provider always assigns character type for columns with mixed data types. It also converts numeric data to character data.

VERSION=2010 or later | 2007 | 2003 | 2002 | 2000 | 97 | 95 | 5
specifies the version of the Excel workbook or Access database file that you want to create if the file does not exist. It is optional to enclose the version in quotation marks.

You can omit VERSION= to create a 2007 or later Excel .xlsb or .xlsx file. The file extension indicates to SAS the VERSION= value.

Alias VER

Default 97 for the Excel engine

Notes You do not need to specify this option if you do not know the version of your Microsoft Excel workbook or Access database file. However, if you want to create a new workbook or Access database file, you can use this option to specify the version that you create.

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 11.3 *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 [3]	Numeric
Fraction	See Note [3]	Numeric
Currency	DOLLAR21.2	Numeric
Accounting	DOLLAR21.2	Numeric
Date	DATE9. See Note [1, 2]	Numeric
Time	TIME8.[1, 2]	Numeric

1 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.

2 Note that the SAS date time value uses 01Jan1960 as a cutoff line, whereas the ACE provider date value uses 30Dec1899 as a cutoff line for internal values.

3 To access Fraction or Percent format data in your Excel file, you can use the FORMAT statement to assign the FRACT. or PERCENT. format in your DATA step code.

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= Data Set Option" on page 192 during output processing.

Table 11.4 *Default Excel Formats Assigned for SAS Variable Formats*

SAS Variable Format	XLS and XLSX Column Data Type
$BINARYw.	Text
$CHARw.	Text
$HEX w.	Text
$w.	Text

SAS Variable Format	XLS and XLSX Column Data Type
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
DATETIMEw.d	Date
DDMMYYw.	Date
HHMMw.d	Time
JULDAYw.	Date
JULIANw.	Date
MMDDYYw.	Date
MMYYw.d	Date
MONTHw.	Date
MOYYw.	Date
WEEKDATEw.	Date

SAS Variable Format	XLS and XLSX Column Data Type
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 Excel 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 Microsoft Excel file database. The time value is set to less than or equal to 24 hours. Set the time environment variables with this code:

```
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 Database Files

Statements

The 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 Access engine for the Microsoft Access database uses the Microsoft ACE 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 "Overview: SAS SQL Pass-Through Facility for Excel and Access Engines" on page 217.

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 Access LIBNAME engine.

Table 11.5 *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 [5]	Numeric
Number (FieldSize=Integer)	6.	Numeric
Number (FieldSize=Long Integer)	11.	Numeric
Number (FieldSize=Single)		Numeric

MDB Field Data Type	SAS Variable Format	SAS Variable Type
Number (FieldSize=Double)		Numeric
AutoNumber (FieldSize=Long Integer)	11.	Numeric
AutoNumber (FieldSize=Replication ID)	$38.	Character
CURRENCY	DOLLAR21.2	Numeric
Date/Time	DATETIME. [1, 2]	Numeric
Text	$w. [3]	Character
Memo	$w. [4]	Character
OLE Object	$w. [4]	Character
Hyperlink	$w. [4]	Character

1 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.

2 The SAS date/time value uses 01Jan1960 as the cutoff date. The ACE provider date/ time value uses 30Dec1899 as the cutoff date.

3 The width of $w. is equal to the field size of the column defined in your Microsoft Access table.

4 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.

5 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 Access LIBNAME engine. You can override these default conversions by using the DBTYPE data set option during processing.

Table 11.6 *Default DB Data Types Assigned for SAS Variable Formats*

SAS Variable Format	MDB Data Type
$BINARYw.	Text (VarChar) or Memo (LongText) [1]

SAS Variable Format	MDB Data Type
$CHARw.	Text (VarChar) or Memo (LongText)[1]
HEX w.	Text (VarChar) or Memo (LongText) [1]
$w.	Text (VarChar) or Memo (LongText)[1]
w.d	Number [2]
BESTw.	Number [2]
BINARYw.	Number [2]
COMMA w.d	Number [2]
COMMAXw.d	Number [2]
Ew.	Number [2]
FRACTw.	Number [2]
HEXw.	Number [2]
NEGPARENw.d	Number [2]
PERCENTw.d	Number [2]
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

SAS Variable Format	MDB Data Type
MMYYw.d	Date/Time
MONTHw.	Date/Time
MOYYw.	Date/Time
WEEKDATEw.	Date/Time
WEEKDATXw.	Date/Time
WEEKDAYw.	Date/Time
WORDDATEw.	Date/Time
WORDDATXw.	Date/Time

1 If the character format length is greater than 255 characters, the loaded format is Memo. Otherwise, the loaded format is Text.

2 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 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;
```

Setting Environment Variables: LOAD_DBL

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;
```

12

SQL Pass-Through Facility for Excel and Access Engines

Overview: SAS SQL Pass-Through Facility for Excel and Access Engines 217

Dictionary 218
SQL Pass-Through Facility for ACCESS and EXCEL Engines, Statements 218
CONNECT Statement 220
DISCONNECT Statement 225
EXECUTE Statement 226
CONNECTION TO Component 227
Microsoft Jet and ACE Providers Supported Data Types Format 232
Jet Special Queries 234
Special Jet Command 236

Overview: SAS SQL Pass-Through Facility for Excel and Access Engines

The SQL procedure is one of the ways that SAS implements the Structured Query Language (SQL). 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.

SQL pass-through uses SAS/ACCESS to connect to a local data source and to send statements directly to the data source for execution. The SAS PC Files Server is not required. Using PROC SQL is an alternative to the SAS/ACCESS LIBNAME statement on page 159. 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 the "CONNECT Statement" on page 220 and the "DISCONNECT Statement" on page 225.

- Send dynamic, non-query, data source-specific SQL statements to a data source using the "CONNECT Statement" on page 220 or the "EXECUTE Statement" on page 226.

- Retrieve data directly from a data source using the "CONNECTION TO Component" on page 227

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 Pass-Through Facility for ACCESS and EXCEL Engines, Statements

Implements the Structured Query Language through a SAS procedure.

Restriction: The Access and Excel engines require that SAS and the Microsoft ACE driver have the same bitness and SAS must be running on Windows.

See: *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* **AS** *<alias>*
<(*connect-statement-arguments*)>
<(*database-connection-arguments*)>;

CONNECT USING *libref* **AS** *<alias>*;

Required Arguments

data-source-name
Specifies the engine to use to connect to the data, either Excel or Access. You can also specify an optional alias in the CONNECT statement.

libref
specifies the libref for which the database connection has already been established through the LIBNAME statement.

Optional Arguments

AS *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 225.

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 Excel or Access 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 219 for more information about these macro variables.

CONNECT Statement Arguments

CONNECT statement arguments are supported by the pass-through facility's CONNECT statement for the Excel or Access engine. 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 Access Database table as soon as they are submitted. No rollback is possible.

NO
specifies that the 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.

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.

Aliases 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

Database Statement Arguments

The arguments that are listed below are available with the SQL pass-through facility for the Access and Excel engines. These arguments provide information to the pass-through facility to connect to local Excel files or to Access database files.

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

DBPASSWORD=*database-password*
specifies a password for an Access database that has database-level security assigned. Use with the USER= ID option. Use PASSWORD= for databases with 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.

Restriction Microsoft Access only

PASSWORD=*user-password*
specifies a password for an Access database that has user-level security assigned. Use with the USER= ID option. Passwords are case sensitive.

Restriction Microsoft Access only

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 or .accdb file, you need to use this option and the PASSWORD= option to access your file. If you have database-level security use DBPASSWORD=

Restriction Microsoft Access only

VERSION=*2007 | 2003 | 2002 | 2000 | 97 | 95 | 5*
sets the version of Microsoft Excel workbook. The default value is 97.

Alias VER

Note You do not need to specify this option for an existing Microsoft Excel or Access file. If you want to create a new Microsoft Excel workbook file or Access 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.

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 ACCESS 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 293 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 enter 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-specificSQL 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 SQL 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

CONNECTION TO *data- source* **AS** *alias (connect statement arguments) (database connection-options)*;

Summary of Optional Arguments

ALIAS
DATABASE CONNECTION ARGUMENTS
DATA SOURCE NAME
MSENGINE=ACE | JET

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.

MSENGINE=ACE | JET
specifies the Microsoft 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 Microsoft Windows.

Default ACE

Restriction It is recommended that this option is used to create only a Windows 95 format file.

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.

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 file name 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 file name"*
specifies the path and file name 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*
specifies the password for Access database files with database-level security. The USER= option specifies the user account for the password.

Alias DBPWD | DBPW

Restriction Microsoft Access database only.

Notes Use PASSWORD= option to access files secured at the user-level.

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 for Access database files with user-level security assigned. The user ID is specified with USER= .

Alias PWD | PW | PASS | PASSWORD

Restriction Microsoft Access database files only.

Notes For files secured at the database level, use DBPASSWORD= to specify the password.

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 databases, you need to use the USER= and PASSWORD= options to access your file if you user-level security is set on your .mdb or .accdb file. If you have database-level security, use USER= and DBPASSWORD= options.

Alias UID

Restriction Microsoft Access database files only.

Note Use the SERVERUSER= option in the CONNECT statement to specify the domain name and user ID for the PC running the SAS PC Files 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.

Examples

Example 1: 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`);
```

Example 2: Use the MSENGINE= Option

```
proc sql;
  connect to excel(path='file name.xlsx');

  create table work.class as
  select *
  from connection to excel
    (select * from class);

  disconnect from excel;
quit;
```

Microsoft Jet and ACE Providers Supported Data Types Format

Valid data types that are supported by the Microsoft Jet and ACE providers.

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		1
COUNTER	10			2
GUID	16			
CURRENCY	19			
DATETIME	8		#	3
VARCHAR	255	max length		4
LONGTEXT	536,870,910			5
VARBINARY	255			
BIGBINARY	4000			

Data Type	Column Size	Create Parameters	Prefix and Suffix	Comments
LONGBINARY	1,073,741,823			[5]

1 When you use the data type DECIMAL, you can specify precision and scale.

2 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.

3 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#).

4 When you use the data types VARCHAR or VARBINARY, you must specify the maximum length.

5 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 Microsoft Excel and Drop and Create a Table

Connects to Excel, drops the DEMO table, and creates a new table.

```
PROC SQL;
   CONNECT TO EXCEL AS db (PATH='c:\temp\demo.xlsx');
   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 Microsoft Access and Drop and Create a Table

Connects to 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;
```

Jet Special Queries

Queries that return information about tables, columns, and procedures.

Syntax

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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Examples

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 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 Command

The Access and Excel LIBNAME engines that access Microsoft Access and Microsoft Excel data support several special commands in the SQL 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;
```

PART 5

LIBNAME Statement for XLSX and JMP Engines under UNIX and Microsoft Windows

Chapter 13
LIBNAME Statement for the XLSX Engine under UNIX and Microsoft Windows 241

Chapter 14
LIBNAME Statement for the JMP Engine under UNIX and Microsoft Windows 247

13

LIBNAME Statement for the XLSX Engine under UNIX and Microsoft Windows

Dictionary 241
LIBNAME Statement: XLSX Engine 241

Dictionary

LIBNAME Statement: XLSX Engine

Associates a SAS libref with a Microsoft Excel workbook that resides on UNIX or Microsoft Windows. SAS PC Files Server is not required.

Valid in:	Anywhere
Restriction:	You cannot use special characters in the FILENAME statement in .xlsx files.
Notes:	PC File Server is a server component of the product SAS/ACCESS to PC File. SAS/ACCESS to PC File must be licensed? It is not necessary to install SAS PC File Server. Invalid UTF-8 characters are replaced with question marks so that Excel can open .xlsx files. Reading files with .xlsm extensions are supported.

Syntax

LIBNAME < *libref*> **XLSX** <*'physical-path-and-file-name*.xlsx'>
<*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 Microsoft Excel .xlsx spreadsheet is stored. The association between a libref and a SAS library lasts only for the duration of the SAS session or until you change the libref or discontinue it with another LIBNAME statement.

XLSX
is the SAS LIBNAME engine name for reading and writing Microsoft Excel files in the .xlsx file format. The LIBNAME statement associates a libref with the XLSX engine, which supports connections to Microsoft Excel 2007, 2010, and later files that reside on UNIX and Microsoft Windows. This LIBNAME engine is new starting in SAS/ACCESS 9.4M2 Interface to PC Files.

The engine name XLSX is required. Otherwise, SAS/ACCESS assumes the use of the EXCEL LIBNAME engine for Microsoft Excel for .xlsx files.

Note: The XLSX engine enables you to read and write .xlsx data directly across UNIX and Microsoft Windows systems. If you have Excel .xlsx files stored on a UNIX system, you do not need to use the SAS PC Files Server (or the PC Files LIBNAME engine) to access the files. If you have SAS running on UNIX and your .xlsx files reside remotely on Windows, you need to use the LIBNAME Statement: PC Files Engine on page 255 to access them.

The XLSX engine enables you to replace an existing worksheet or to add a new worksheet, but it cannot be used to update the values in individual worksheets.

When importing (reading) .xlsx data, the XLSX engine reads mixed data (that is, columns containing numeric and character values) and converts it to character data values.

The XLSX engine allows the sequential reading of data only. That is, it does not support random access. It does not support certain tasks that require random access, such as the RANK procedure, which requires the reading of rows in a random order.

Example This LIBNAME statement specifies **wsinv** as a libref to a SAS library. The engine name, XLSX, specifies an engine that supports the connection to the Microsoft Excel .xlsx file format.

```
LIBNAME wsinv XLSX 'c:\WestSector\Q1_Invoices.xlsx';
```

***'physical-path-and-file-name*.xlsx'**
is the physical-path and file name of a Microsoft Excel 2007 or later workbook. Using the extension (.xlsx) is required. Enclose the path and workbook name in quotation marks.

See "Details" on page 244

SAS/ACCESS LIBNAME-options
define how SAS interacts with the external data source (in this case, XLSX files), providing enhanced control of how SAS processes data source objects.

Because the XLSX LIBNAME engine works on any version of Microsoft Excel 2007 or later, it has no engine-specific LIBNAME options, such as VERSION=. The XLSX LIBNAME engine does support the "PRESERVE_TAB_NAMES= LIBNAME Statement Option" in *SAS/ACCESS for Relational Databases: Reference*.

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;
```

ACCESS=READONLY | TEMP
READONLY assigns a read-only attribute to an entire SAS library. SAS does not allow you to open a data set in the library in order to update information or write new information.

TEMP specifies that the SAS library be treated as a scratch library. That is, the system does not consume CPU cycles to ensure that the files in a Temp library do not become corrupted.

REPEMPTY=YES | NO
controls replacement of like-named temporary or permanent SAS data sets when the new one is empty.

YES specifies that a new empty data set replace an existing data set. This is the default.

NO specifies that a new empty data set with a given name not replace an existing data set with the same name.

For both the convenience of replacing existing data sets with new ones that contain data and the protection of not overwriting existing data sets with new empty ones that are created by mistake, set REPLACE=YES and REPEMPTY=NO.

Details

The **XLSX LIBNAME engine** lets you read and write Excel files between UNIX and Microsoft Windows. SAS can be running on Windows or UNIX. The Excel files can also reside on Windows or UNIX. The XLSX engine is available starting with the release of SAS 9.4M2.

Unlike the Excel and PC Files LIBNAME engines, the XLSX LIBNAME engine does not go through the Microsoft Access Database Engine (ACE) to access the Excel data. Instead, you can access the files directly. Doing so might give you greater performance, yet you cannot apply most SAS options that would enable SAS to process the Excel data being read.

For example, you can access an .xlsx file stored on a UNIX file system. Then you can write a DATA step or use PROC SQL to rename columns or apply labels, handle null values, apply formats, and so on.

```
libname myhr XLSX "/usr/wolfen/benefits/HRworkbook.xlsx";
proc sql;
select lname label="Last Name", empid, start_date format=date9.
from myhr.perm_employees
where state="NC";
quit;
```

You can use the XLSX engine to read from and write to a Microsoft Excel file as if it were a SAS data set. The LIBNAME statement associates a libref with the XLSX engine to access tables in a workbook. The XLSX engine enables you to connect to an Excel .xlsx data source and to specify an external data object name in a two-level SAS name.

Here is an 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.

This statement assigns the libref, **myxlsx**, to a Microsoft Excel .xlsx file and uses the XLSX engine:

```
LIBNAME myxlsx XLSX 'c:\demo.xlsx';
```

The **Demo.xlsx** workbook contains a number of objects, including several tables, such as **Staff**. After you assign the libref, you can reference the Microsoft Excel workbook 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, **myxlsx.Staff** is the two-level SAS name for the **Staff** table in the Microsoft Excel workbook, **Demo**.

```
PROC SQL;
   SELECT idnum, lname
   FROM myxlsx.staff
   WHERE state='NY'
   ORDER BY lname;
QUIT;
```

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

```
DATA newds;
 SET myxlsx.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=myxlsx.staff;
RUN;
```

This statement lists the database objects in the **myxlsx** library:

```
PROC DATASETS LIBRARY=myxlsx;
QUIT;
```

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 mypclib LIST;
```

To list attributes of all libraries:

```
LIBNAME _ALL_ LIST;
```

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 mypclib CLEAR;
```

To clear all user-defined librefs:

```
LIBNAME CLEAR;
```

See Also

"Microsoft Excel Workbook Files" on page 201

14

LIBNAME Statement for the JMP Engine under UNIX and Microsoft Windows

Dictionary 247
LIBNAME Statement: JMP Engine 247

Dictionary

LIBNAME Statement: JMP Engine

Associates a libref with a JMP data table and enables you to read and write JMP data tables.

Valid in: Anywhere

Category: Data Access

See: 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.

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. SAS imports data from JMP files that are saved with version 7 or later formats, and it exports data to JMP files with version 7 or later formats. SAS no longer supports JMP files with versions 3 through 6 formats.

  For more information, see *SAS/ACCESS Interface to PC Files: Reference*.
- the LIBNAME statement for the JMP engine

Note: The JMP LIBNAME engine does not support extended attributes. If you want extended attributes, either use the IMPORT procedure or use the EXPORT procedure with **dbms=jmp**.

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;
```

PART 6

LIBNAME Statement for the PC Files Engine with the PC File Server

Chapter 15
LIBNAME Statement: PC Files Engine with the PC File Server 253

Chapter 16
SQL Pass-Through Facility for the PC Files Engine 291

Chapter 17
Special PC Files Query Support on Microsoft Windows and UNIX 307

15

LIBNAME Statement: PC Files Engine with the PC File Server

LIBNAME Statement for PC Files Engine on UNIX and Microsoft Windows **254**
Overview 254
PC Files LIBNAME Engine and the PC Files Server 254

Dictionary **255**
LIBNAME Statement: PC Files Engine 255
LIBNAME Statement Options for PC Files Engine Statement 263
SAS Data Set Options for the LIBNAME Statement for PC Files Engine Data Set Option 271
AUTOCOMMIT= Data Set Option 271
CURSOR_TYPE= Data Set Option 272
DBCOMMIT= Data Set Option 273
DBCONDITION= Data Set Option 274
DBCREATE_TABLE_OPTS= Data Set Option 274
DBENCODING= Data Set Option 275
DBFORCE= Data Set Option 276
DBGEN_NAME= Data Set Option 276
DBINDEX= Data Set Option 277
DBKEY= Data Set Option 278
DBLABEL= Data Set Option 279
DBMAX_TEXT= Data Set Option 280
DBNULL= Data Set Option 280
DBNULLKEYS= Data Set Option 281
DBSASLABEL= Data Set Option 282
DBSASTYPE= Data Set Option 283
DBTYPE= Data Set Option 284
ERRLIMIT= Data Set Option 285
INSERT_SQL= Data Set Option 286
INSERTBUFF= Data Set Option 286
NULLCHAR= Data Set Option 287
NULLCHARVAL= Data Set Option 288
READBUFF= Data Set Option 289
SASDATEFMT= Data Set Option 289

LIBNAME Statement for PC Files Engine on UNIX and Microsoft Windows

Overview

The PC Files LIBNAME engine assigns a libref to Microsoft Excel, Microsoft Access, and ODBC data that resides on the SAS PC Files Server. Librefs enable you to reference workbooks, spreadsheets (or worksheets), 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. For more information, see "When is the PC Server Required?" on page 14.

PC Files LIBNAME Engine and the PC Files Server

The PC Files engine runs on UNIX or Microsoft Windows. The SAS PC Files Server runs only on Windows. The PC Files LIBNAME engine makes the connection from UNIX or Windows to the PC Files Server to read and write Excel and Access data or an ODBC table that resides on the server. You use LIBNAME options to specify the connection information, such as the name of the Windows machine that is running the SAS PC Files Server, the server name and password, and so on.

SAS PC Files Server runs on both 32- and 64-bit Microsoft Windows as either a 32-bit or a 64-bit application. You install the server on Windows, which enables you to access data on Windows or UNIX. To access the Excel or Access data, SAS/ACCESS Interface to PC Files must be installed on the client. The SAS PC Files Server must be installed and running on the Windows machine where the Excel or Access data resides.

SAS PC Files Server enables SAS to do the following:

- SAS on Windows 64–bit can access Microsoft Excel and Access database files when 32-bit Microsoft Office is present.
- SAS on UNIX can access Excel and Access database files that are stored on the PC Files server on Windows.

Use the PC Files LIBNAME engine only when you need to use the SAS PC Files Server. The server must be installed on a Windows PC by using the SAS Deployment Wizard or separately using SAS Downloads on support.sas.com. For more information about deploying and administering the server, see *SAS PC Files Server: Installation and Configuration Guide*.

Note: The SAS PC Files Server is not included in SAS Viya 3.4 and earlier. You can connect to the PC Files Server from a SAS 9 client, if one is installed.

Dictionary

LIBNAME Statement: PC Files Engine

Associates a SAS libref with a Microsoft Excel workbook or Access database or with an ODBC data source.

Valid in: Anywhere

Restrictions: The PC Files LIBNAME engine requires a connection to the PC File Server.

The PC File Server is not included with SAS Viya 3.4 and earlier. Connect to it using the SAS 9 client.

The PC Files engine supports up to 255 columns of an Excel file.

Syntax

Form 1: **LIBNAME** *libref* **PCFILES** *SAS/ACCESS-connection-options SAS/ACCESS-LIBNAME-options*;

Form 2: **LIBNAME** *libref* CLEAR | _ALL_ CLEAR;

Form 3: **LIBNAME** *libref* LIST | _ALL_ LIST;

Optional Arguments

libref
is any SAS name that associates SAS with the SAS library where the Microsoft Excel spreadsheet, Microsoft Access database or ODBC data is stored. The association between a libref and a SAS library lasts only for the duration of the SAS session or until you change the libref or discontinue it with another LIBNAME statement.

PCFILES
is the LIBNAME engine running on UNIX or Microsoft Windows to read and write Excel and Access files on the SAS PC Files Server. The PC Files server runs only on Windows.

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 Statement Options for PC Files Engine Statement" on page 263

SAS/ACCESS-connection-options
provide connection information to PC Files engine to connect to your PC files. For the PC Files LIBNAME engine, the connection is made through the SAS PC Files Server.

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.

CLEAR
clears one or all librefs.

Specify *libref* to clear single libref. Specify _ALL_ to clear all librefs.

ALL
specifies that CLEAR= and LIST= arguments apply the argument to all librefs.

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.

Details

PC Files Engine Connection Options through SAS PC Files Server

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*
specifies the password for Microsoft Access database files with database-level security. Used with the USER= ID option. This security level can be defined instead of user-level security.

Alias DBPWD | DBPW

Restriction Microsoft Access databases 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 SAS PC Files Server is running.

MSENGINE=ACE | JET
determines the Microsoft database engine (ACE or Jet) 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 and later formats.

Default ACE

PASSWORD=*user-password*
specifies a password for Microsoft Access database files secured by user-level security. The user account is specified by USER= ID.

Alias PWD | PW | PASS | PASSWORD

Notes Use DBPASSWORD= for Access database files that are secured at the database level.

Passwords are case sensitive.

PATH='*path-name-and-file-name*'
specifies the full path and file name 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.xlsx';
```

Restriction When creating a new Microsoft Excel file, do not use the .xlsm extension. Instead, use only the .xls, .xlsx or .xlsb extension.

Notes When your read an Excel file, use files with these extensions: .xls, .xlsx, .xlsb, or .xlsm.

Always use the .mdb or .accdb file extension for Microsoft Access files.

When you create a new .xlsx file with the PC Files engine, an extra sheet named "SAS_Empty" is also created and cannot be deleted using the ACE driver. This empty sheet can be ignored. This sheet is

also created when you specify DBMS=EXCELCS, which uses the PC Files engine.

PORT=*port-number*
specifies the port number or service name that SAS PC Files Server is listening to on the PC.

Alias SERVICE | SERVICE_NAME

Default 9621

PRESERVE_COMMENTS=YES | NO
specifies whether comments should be kept and passed down to your data source for processing.

Category Data Set Control

Default NO

Note When PRESERVE_COMMENTS=YES, comments can be passed in an explicit SQL query to your data source for processing. The comment is passed from the CONNECT statement in PROC SQL. The comment that you specify can contain database-specific keywords (hints), special characters, and markup text that are used by your native query processor. To see the query that is passed to your data source, including any comments that you add, specify SASTRACE=',,,d' in the OPTIONS statement.

SERVER=*PC-server-host-name*
specifies the name of the PC that is running the SAS PC Files Server. This name is required for 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 (wxp320), a fully qualified network name (*wxp320.domain.com*), or an IP address.

SERVERPASS='*server-user-password*'
specifies the password for the SAS PC Files Server for the User ID given. If the account has no password, omit this option. Always enclose the value in quotation marks in order to preserve the case of the password.

Alias SERVERPASSWORD | SERVERPW | SERVERPWD

Notes Passwords are case sensitive.

Use the PASSWORD= or DBPASSWORD= option for Access 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 that is running SAS 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 Access 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 SAS PC Files Server to allow Integrated Windows Authentication. This is a mechanism for the Microsoft Windows client and the 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, .xlsx, .mdb or .accdb file extension.

USER=*user ID*
specifies a user account name, if one is required to connect to Microsoft Access.

Alias UID

Restriction Microsoft Access database files only.

Interaction If you have user-level security assigned on your .mdb or .accdb file, use the PASSWORD= option. If you have database-level security, use the DBPASSWORD= option

Note Use the SERVERUSER= option to connect to a PC file server.

VERSION=2010 or later | 2007 | 2003 | 2002 | 2000 | 97 | 95 | 5
sets the version for a new Excel workbook or Access database. It is optional to enclose the version in quotation marks.

VERSION= can be used to override the version specified in the PATH= option. You can omit this option for an existing Excel or Access file.

Alias VER

Default 97

Note Excel 2007 or later (.xlsx files) have a different file format than that of Excel (2003, 2000, and 97) for .xls files. Excel 5 and 95 share a file format that is different from the later versions of Excel.

Access Data Directly from a PC File

You can use PC Files engine to directly access PC data from Windows and UNIX. You can read from and write to a variety of PC file data residing on a PC, including Microsoft Excel, Microsoft Access, and any other ODBC data source.

The engine uses ODBC to support assigning a libref to Microsoft Excel and Microsoft Access files on a PC from SAS on UNIX. You can reference worksheets, 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 mypclib 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 mypclib LIST;

LIBNAME _ALL_ LIST;
```

Examples

Example 1: Assigning a Libref to a Microsoft Access Database

```
LIBNAME mymdb PCFILES PORT=9621 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, myxlsx, for a Microsoft Excel workbook:

```
LIBNAME myxlsx PCFILES SERVER=D2323 PATH='c:\demo.xlsx';
```

The demo.xlsx workbook contains a number of sheets, such as sheet1. After you assign the libref, you can reference the Excel worksheet 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 myxlsx.'sheet1$'n;
RUN;
```

When using a LIBNAME statement with Excel, refer to Excel worksheets as n-literals because of the “$” character. If you are referencing a named range in a Microsoft Excel worksheet, it is not necessary to refer to it as an n-literal.

Reference a named range called pageone in a Microsoft Excel workbook:

```
DATA a;
```

```
   SET myxlsx.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 SAS PC Files Server does not support the INSERTBUFF= option with value greater than 1.

```
PROC SQL;
    SELECT * FROM myxlsx.Invoice (READBUFF=25);
QUIT;
```

Example 3: Assigning a Libref to a Microsoft SQL Server Database

Create a libref, mysqlsrv, to a Microsoft SQL Server database using ODBC, using the SAS PC Files Server on Windows:

```
LIBNAME mysqlsrv PCFILES SERVER=D2323 DSN=MQIS USER=scott
            PWD=tiger SCHEMA=dbo;
```

Using the mysqlsrv libref, create a SAS data set called sqltest from the crime table in the SQL Server database. Here are two examples:

```
DATA work.sqltest;
   SET mysqlsrv.crime;
RUN;
```

```
PROC sql;
   CREATE TABLE work.sqltest AS SELECT * FROM mysqlsrv.crime;
QUIT;
```

Using the mysqlsrv libref, create a SQL Server table called newtable from the SAS data set, sqltest:

```
DATA mysqlsrv.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 SAS 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;
```

LIBNAME Statement Options for PC Files Engine Statement

provide enhanced control over how SAS processes PC Files engine data.

See: "LIBNAME Statement: PC Files Engine" on page 255. Many of these LIBNAME options are also available as data set options.

"SAS Data Set Options for the LIBNAME Statement for PC Files Engine Data Set Option" on page 271 for additional information. These advanced options do not need to be specified for many of the tasks that you perform.

Syntax

LIBNAME <*libref*> **PCFILES** <*'physical-path and file-name.ext'*>
<*SAS/ACCESS engine-connection-options*>
<*SAS/ACCESS LIBNAME-options*>;

Summary of Optional Arguments

ACCESS=READONLY
AUTOCOMMIT=YES | NO
CURSOR_TYPE=DYNAMIC | FORWARD_ONLY | KEYSET_DRIVEN | STATIC
DBCOMMIT=*number-of-rows*
DBGEN_NAME=DBMS | SAS
DBINDEX=YES | NO | *index-name*

DBMAX_TEXT=*n*
DBNULLKEYS=YES | NO
DBSASLABEL=COMPAT | NONE
DEFER=NO | YES
DIRECT_SQL=YES | NO | NONE | *specific-functionality*
INSERT_SQL= YES | NO
INSERTBUFF=*number-of-rows*
PRESERVE_COMMENTS=YES | NO
READBUFF=*number-of-rows*
SCAN_TEXTSIZE=YES | NO
SCAN_TIMETYPE=YES | NO
SPOOL=YES | NO
STRINGDATES=YES | NO
USE_DATETYPE=YES | NO

Optional Arguments

ACCESS=READONLY
indicates that tables and views can be read but not updated; specifically, the LIBNAME connection is read-only.

AUTOCOMMIT=YES | NO
determines whether the PC Files engine commits (saves) updates to Microsoft Access database tables 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 PC Files LIBNAME engine automatically performs the commit when it reaches the end of the file.

Default NO

Restriction This option is valid only with Microsoft Access Database files.

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 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

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

Note: If you specify both DBCOMMIT= and ERRLIMIT=, the DBCOMMIT= is issued before the rollback. Because the DBCOMMIT= option is issued before the rollback, the DBCOMMIT= option overrides the ERRLIMIT= option (rollback) in this situation.

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

Restriction This option is valid only for Microsoft Access database files.

DBINDEX=YES | NO | *index-name*
If you plan to use the KEY= keyword in a DATA step join, you must use the DBINDEX=yes or DBINDEX=*index-name* LIBNAME option or data set option. The following example shows how to use this option:

```
LIBNAME db pcfiles path="path-to-MSAccess-file" DBINDEX=yes;
DATA work.newtab;
SET work.keytab;
SET db.oldtab key=keycolumn;
RUN;
```

Restrictions DBINDEX= is not valid with PROC IMPORT or PROC EXPORT.

DBINDEX= is not valid with the Excel engine or the Access engine.

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, in bytes, 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 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.

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; `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 ACE 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

INSERT_SQL= YES | NO
determines the method to use to insert rows into the Access table.

YES
specifies that the PC Files engine uses the data source's SQL insert method to insert rows into a table.

NO
specifies that the PC Files 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 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

PRESERVE_COMMENTS=YES | NO
specifies whether comments should be kept and passed down to your data source for processing.

Category Data Set Control

Default NO

Note When PRESERVE_COMMENTS=YES, comments can be passed in an explicit SQL query to your data source for processing. The comment is passed from the CONNECT statement in PROC SQL. The comment that you specify can contain database-specific keywords (hints), special characters, and markup text that are used by your native query processor. To see the query that is passed to your data source, including any comments that you add, specify SASTRACE=',,,d' in the OPTIONS statement.

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 ACE 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

Defaults YES [Microsoft Excel workbook]

NO [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 rowset 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 rowset 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_DATETYPE=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

Defaults YES [Microsoft Excel workbook]

NO [Microsoft Access database]

SAS Data Set Options for the LIBNAME Statement for PC Files Engine Data Set Option

Specifies SAS/ACCESS data set options on a SAS data set when you access data using the LIBNAME statement for the PC Files engine.

Notes: A data set option applies only to the SAS data set on which it is specified, and it remains in effect for the duration of the DATA step or procedure.

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

See: "Overview of SAS Data Set Options for the Excel and Access LIBNAME Statement" on page 180.

For information about using SAS data set options, refer to *SAS Data Set Options: Reference*.

Example: This generic example shows the format for 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 SAS data set option REPLACE= is not supported by SAS/ACCESS interfaces.

AUTOCOMMIT= Data Set Option

Determines whether the engine commits (saves) updates as soon as you submit them.

Valid in: DATA and PROC steps

Default: LIBNAME option setting

Restriction: Valid only with Microsoft Access database files

Syntax

AUTOCOMMIT=YES | NO

Syntax Description

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

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

CURSOR_TYPE= Data Set Option

Specifies the cursor type for read-only cursors and for cursors to be updated.

Valid in: DATA and PROC steps

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 ACE provider uses a default. The OLE DB properties applied to an open rowset are as follows:

Table 15.1 *OLE DB Properties Applied to an Open Rowset*

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.

DBCOMMIT= Data Set Option

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

Valid in: DATA and PROC steps

Default: LIBNAME option default: 1000

Restriction: Valid only with Microsoft Access database files.

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, the Access engine 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  mymdb.dept(DBCOMMIT=10);
 SET mysas.staff;
RUN;
```

DBCONDITION= Data Set Option

Specifies criteria for subsetting and ordering data.

Valid in: DATA and PROC steps

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 DBKEY= option is ignored when you use DBCONDITION.

DBCREATE_TABLE_OPTS= Data Set Option

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.

DBENCODING= Data Set Option

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 values, see “Encoding Values in SAS Language Elements” in *SAS National Language Support (NLS): 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= Data Set Option

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= Data Set Option

Specifies whether to rename columns automatically when they contain disallowed characters.

Valid in: DATA and PROC steps

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.

DBINDEX= Data Set Option

Detects and verifies that indexes exist on a DBMS table.

Default: NO

Syntax

DBINDEX=YES | NO | <'>*index-name*<'>

Syntax Description

YES
When PC Files accesses Microsoft Access databases and you plan to use the KEY= keyword in a DATA step join, you must use the `DBINDEX=yes` or `DBINDEX=`*`index-name`* LIBNAME option or data set option.

NO
indicates that no automated index search is performed.

index-name
verifies the index name that is specified for the index columns.

See Also

"DBINDEX=YES | NO | *index-name* " on page 265

DBKEY= Data Set Option

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

Default: none

Interaction: affects DBNULLKEYS= and DBCONDITION=

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= Data Set Option

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 256 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= Data Set Option

Determines the length, in bytes, 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.

DBNULL= Data Set Option

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

Restriction: Supported only for Microsoft Access databases

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.

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(DBNULLl=(_ALL_=no jobcode=YES));
 SET mydblib.employees;
RUN;
```

DBNULLKEYS= Data Set Option

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

Valid in: DATA and PROC steps

Default: LIBNAME setting

Interaction: affected by DBKEY=

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= data set 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 = ?)
```

DBSASLABEL= Data Set Option

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

Valid in: DATA and PROC steps

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= Data Set Option

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

Valid in: DATA and PROC steps

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= Data Set Option

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= Data Set Option

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

Valid in: DATA and PROC steps

Default: 1

Restriction: valid only with Microsoft Access database files.

Interaction: DBCOMMIT= option overrides ERRLIMIT= in certain situations

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 10th error. The MyDBLib libref was assigned in a prior LIBNAME statement.

```
DATA mydblib.employee3 (ERRLIMIT=10);
```

```
   SET mydblib.employees;
   WHERE salary>40000;
RUN;
```

INSERT_SQL= Data Set Option

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.

INSERTBUFF= Data Set Option

Specifies the number of rows in a single insert.

Valid in: DATA and PROC steps

Default: LIBNAME option setting

Restriction: valid only with Microsoft Access database files.

Syntax

INSERTBUFF=*number-of-rows*

Syntax Description

number-of-rows
specifies the number of rows to insert in the Access database. 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 even when multiple inserts are performed.

Note: The SAS 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.

NULLCHAR= Data Set Option

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, NULLCHAR= determines what is inserted when NULL values are not allowed.

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

NULLCHARVAL= Data Set Option

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

Interaction: affected by NULLCHAR= option; affects DBFORCE= option

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.

READBUFF= Data Set Option

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

Valid in: DATA and PROC

Default: LIBNAME option setting

Restriction: valid only with Microsoft Access database files

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.

SASDATEFMT= Data Set Option

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 SASDATEFMT= option enables you to convert date values from a SAS date format to different SAS date format.

Use SASDATEFMT= 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.

Note: For non-English date types, SAS automatically converts the data to the SAS type of NUMBER. The SASDATEFMT= option does not currently handle these date types. 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.

16

SQL Pass-Through Facility for the PC Files Engine

Overview: SQL Pass-Through Facility for the PC Files Engine 291

Dictionary 292
SQL Pass-Through Facility for the PC Files Engine Statement: Syntax 292
CONNECT Statement 293
DISCONNECT Statement 299
EXECUTE Statement 300
CONNECTION TO Component 302

Overview: SQL Pass-Through Facility for the PC Files Engine

The SQL procedure is one of the ways that SAS implements the Structured Query Language (SQL). 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.

The SQL pass through facility uses SAS/ACCESS to connect to a data source on the PC file server and to send statements directly to the data source for execution. Using PROC SQL is an alternative 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 SQL pass-through facility, you can do the following:

- Establish and terminate connections with a data source using the "CONNECT Statement" on page 293 and the "DISCONNECT Statement" on page 299.

- Send dynamic, non-query, data source-specific SQL statements to a data source using the "CONNECT Statement" on page 293 or the "EXECUTE Statement" on page 300.
- Retrieve data directly from a data source using the "CONNECTION TO Component" on page 302.

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

SQL Pass-Through Facility for the PC Files Engine Statement: Syntax

Implements the Structured Query Language through a SAS procedure.

See: *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 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* **AS** *<alias>*
<(connect-statement-arguments)>
<(database-connection-arguments)>;

CONNECT USING *libref* **AS** *<alias>*;

Required Arguments

data-source-name
: Specifies the engine to use to connect to the data, which is the PC Files engine. You can also specify an optional alias in the CONNECT statement.

libref
: specifies the libref for which the database connection has already been established through the LIBNAME statement.

Optional Arguments

AS *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 299.

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.

CONNECT Statement Arguments

CONNECT statement arguments are supported by the pass-through facility's CONNECT statement for the PC Files engine. 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 Access Database table as soon as they are submitted. No rollback is possible.

NO
specifies that the 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.

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.

Aliases 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

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 the SAS 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 SAS 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 SAS 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 SAS 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 SAS 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 SAS 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 Access database user IDs.

SERVERPASS=*'server-user-password'*
specifies the password for the SAS PC Files Server for the user ID given. If the account has no password, omit this option. Always enclose the value in quotation marks in order to preserve the case of the password.

Alias SERVERPASSWORD | SERVERPW | SERVERPWD

Notes Passwords are generally case sensitive.

Use the PASSWORD= or DBPASSWORD= option with the USER= option for Access database passwords.

SSPI=YES | NO
enables the SAS 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 or .accdb file. A database password is case sensitive.

Restriction Microsoft Access only.

Interaction Use with USER= option.

Note Use PASSWORD= option if the file has user-level security assigned.

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 or .accdb file, use this option with the PASSWORD= option to access your file. If you have database-level security set, use this option with the DBPASSWORD= option

VERSION=*2007 | 2003 | 2002 | 2000 | 97 | 95 | 5*
sets the version of Microsoft Excel workbook or Access database file. The default value is 97.

Alias VER

Note You do not need to specify this option for an existing Microsoft Excel or Access database file. If you want to create a new Microsoft Excel or Access database 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.

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 ACCESS 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 293 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 enter 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-specificSQL 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 SQL 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

CONNECTION TO *data- source* **AS** *alias (connect statement arguments) (database connection-options)*;

Summary of Optional Arguments

ALIAS
DATABASE CONNECTION ARGUMENTS
DATA SOURCE NAME
MSENGINE=ACE | JET

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.

MSENGINE=ACE | JET
specifies the Microsoft 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 Microsoft Windows.

Default ACE

Restriction It is recommended that this option is used to create only a Windows 95 format file.

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.

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 file name 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 file name"*
specifies the path and file name 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*
specifies the password for Access database files with database-level security. The USER= option specifies the user account for the password.

Alias DBPWD | DBPW

Restriction Microsoft Access database only.

Notes Use PASSWORD= option to access files secured at the user-level.

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 for Access database files with user-level security assigned. The user ID is specified with USER= .

Alias PWD | PW | PASS | PASSWORD

Restriction Microsoft Access database files only.

Notes For files secured at the database level, use DBPASSWORD= to specify the password.

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 databases, you need to use the USER= and PASSWORD= options to access your file if you user-level security is set on your .mdb or .accdb file. If you have database-level security, use USER= and DBPASSWORD= options.

Alias UID

Restriction Microsoft Access database files only.

Note Use the SERVERUSER= option in the CONNECT statement to specify the domain name and user ID for the PC running the SAS PC Files 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.

Examples

Example 1: 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`);
```

Example 2: Use the MSENGINE= Option

```
proc sql;
  connect to excel(path='file name.xlsx');

  create table work.class as
  select *
  from connection to excel
    (select * from class);

  disconnect from excel;
quit;
```

17

Special PC Files Query Support on Microsoft Windows and UNIX

Special PC Files Queries on Microsoft Windows and UNIX 307
Macro Variable to Pass Content to PC Files Queries 309

Special PC Files Queries on Microsoft Windows and UNIX

On Microsoft Windows and UNIX, SAS/ACCESS Interface to PC Files supports special queries. Many databases provide or use system tables that enable 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 PC Files 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", "FKschema", "FKtable-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

The SAS PC Files Server must be running on the Windows 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::SQLLGetTypeInfo);
   CREATE TABLE mytab AS SELECT * FROM CONNECTION TO DB
               (PCFILES::SQLTables "","","","");
QUIT;
PROC PRINT DATA=mytab;
RUN;
```

Macro Variable to Pass Content to PC Files Queries

You can pass extra content to your data source when you pass an explicit SQL query with PROC SQL. To do this, set the LIBNAME option PRESERVE_COMMENTS=YES and pass the content in a comment. The comment can contain keywords or markup that the data source query processor uses.

With this LIBNAME option, you can also pass similar content at the beginning of SQL statements that SAS generates for other operations, such as a call to PROC PRINT. To do this, you define the PCFILES_SQL_COMMENT macro variable. The text that you assign to the macro variable is added at the front of each generated SQL command that is passed to your data source. This text can contain keywords

or markup that your data source query processor uses. To see the query that is passed to your data source, including any text that you have added, specify SASTRACE=',,,d' in the OPTIONS statement.

PART 7

ACCESS and DBLOAD Procedures

Chapter 18
ACCESS Procedure 313

Chapter 19
DBLOAD Procedure 341

Chapter 20
File-Specific Reference for the ACCESS and DBLOAD Procedures 355

18

ACCESS Procedure

Overview: ACCESS Procedure ***313***
Overview: ACCESS Procedure 314
Using ACCESS Procedure Statements 314

Concepts: ACCESS Procedure ***315***
Overview of SAS/ACCESS Descriptors for PC Files 315
Access Descriptors 316
View Descriptors 316
Extracting Data Using a View 317
Tasks and Associated Statements 317

Syntax: ACCESS Procedure ***318***
PROC ACCESS Statement 318
ASSIGN Statement 320
CREATE Statement 321
DROP Statement 324
FORMAT Statement 325
LIST Statement 326
MIXED Statement 327
PATH Statement 328
QUIT Statement 329
RENAME Statement 329
RESET Statement 331
SELECT Statement 332
SUBSET Statement 333
TYPE Statement 333
UNIQUE Statement 334
UPDATE Statement 335

Usage: ACCESS Procedure ***338***
SAS Passwords for Descriptors 338

Results: ACCESS Procedure ***339***
General Guidelines for Performance and Efficient View Descriptors 339
Extracting Data Using a View 339

Overview: ACCESS Procedure

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. This procedure is generally not used anymore and is not supported. It is 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 to 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 “Overview of SAS/ACCESS Descriptors for PC Files” on page 315.

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 re-create the descriptors.

Names for descriptors and columns 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 318 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	`PROC ACCESS DBMS=DBF|DIF|WKn|XLS;`
	`  CREATE libref.member-name.ACCESS;`
	`  required-database-description-statements;`
	`  optional-editing-statements;`
	`RUN;`

create an access descriptor and a view descriptor	`PROC ACCESS DBMS=DBF|DIF|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;`

create a view descriptor from an existing access descriptor	`PROC ACCESS DBMS=DBF|DIF|WKn|XLS`
	`  ACCDESC=libref.access-descriptor;`
	`  CREATE libref.member-name.VIEW;`
	`  SELECT column-list;`
	`  optional-editing-statements;`
	`RUN;`

As the table indicates, you can create one or more access descriptors and view descriptors in one execution of PROC ACCESS, or you can create the descriptors in separate executions.

See "CREATE Statement" on page 321 for additional information.

Concepts: ACCESS Procedure

Overview of SAS/ACCESS Descriptors for PC Files

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

See Overview: ACCESS Procedure on page 314 for additional information.

Table 18.2 *ACCESS Procedure Tasks*

Task	Options and Statements
create an access descriptor	**PROC ACCESS** **DBMS**=DBF | DIF | WK*n* | XLS; **CREATE** *libref.member-name*;

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

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: ACCESS Procedure

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 <*options*>;

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 WKn 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.invq4
          OUT=dlib.invq4;
RUN;
```

See Chapter 5, “EXPORT Procedure,” on page 41

Chapter 4, “IMPORT Procedure,” on page 31

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 321 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 begins 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 re-create 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 re-create 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 321, 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:\sasdemo\specprod.wk4';
   GETNAMES=yes;
   ASSIGN=yes;
   RENAME= productid prodid
          fibername fiber;
   FORMAT productid  4.
          weight     e16.9
          fibersize  e20.13
          width      e16.9;
RUN;
```

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-identifier–1<'>* *<... <'>column-identifier-n<'>>*;

Details

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

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 321 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.

Alias: FMT

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 *<'>column-identifier–1<'><=>SAS format-name*
<... <'>column-identifier-n<'><=>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 321 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 367 for additional information.

PATH Statement

Specifies the path and file name 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 file name<.PC-file-extension>'* | *<'>file name<'>* | *fileref*;

Required Arguments

PATH=*'path and file name.<PC-file-extension>'*
specifies the fully qualified path and file name. You must enclose the entire path and file name 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=*<'>file name<'>*
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 file name includes characters that are invalid in SAS names, such as the dollar sign ($) or if the file name begins with a number, you must enclose the entire file name in quotation marks.

PATH=*fileref*
specifies a fileref that references the path and name of the file. Use the FILENAME statement to assign a file name.

Details

The PATH= statement indicates the path and name of the file that you want to access. The length of the file name 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 321 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-identifier–1<'> <=><'>SAS variable-name–1<'> <…<'>column-identifier-n<'> <=><'>SAS variable-name-n<'>>*;

Details

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

An editing statement, such as RENAME, must follow the CREATE statement and the database-description statements when you create a descriptor. See "Create Statement" on page 321 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-identifier–1<'>* <... *<'>column-identifier-n<'>>*;

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 "SELECT Statement" on page 332.

<'>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 321 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-identifier–1<'>* *<<'>column-identifier-n<'>>*;

Details

The SELECT statement specifies which PC file columns in the access descriptor to include in the view descriptor. This is a required statement, and 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 321 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 321 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 "CREATE Statement" on page 321 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 "RENAME Statement" on page 329 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 UNIQUE=YES, the SAS/ACCESS interface appends numbers to any duplicate SAS variable names, thus making each variable name unique.

See "CREATE Statement" on page 321.

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. This container 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;
```

Usage: ACCESS Procedure

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.

Results: ACCESS Procedure

General Guidelines for Performance and Efficient View Descriptors

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.

19

DBLOAD Procedure

Overview: DBLOAD Procedure ***341***
What Does the DBLOAD Procedure Do? 341
PROC DBLOAD Operating Environments 342
PROC DBLOAD Naming Conventions 342
PROC DBLOAD Statement 342

Syntax: DBLOAD Procedure ***343***
PROC DBLOAD Statement 344
ACCDESC Statement 345
DELETE Statement 345
ERRLIMIT Statement 346
FMTLIB Statement 347
LABEL Statement 347
LIMIT Statement 348
LIST Statement 348
LOAD Statement 349
PATH Statement 350
QUIT Statement 350
RENAME Statement 351
RESET Statement 352
WHERE Statement 352

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.

CAUTION

The DBLOAD Procedure for PC Files is obsolete. This procedure is generally not used anymore and is not supported. It is documented here for the sake of completeness. You are encouraged to rely on more recent functionality to produce similar results.

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 the following:

- 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: DBLOAD Procedure

PROC DBLOAD DATA=*input data* **DBMS**=*data-base-identifier*;

PATH='*path and file name.PC file-extension*' | '*file name*' | *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	Load data to PCs and creates PC files
ACCDESC	Create an access description
DELETE	Delete variables from the new data set
ERRLIMIT	Limit loading obs after n number of errors
FMTLIB	Specify a format library
LABEL	Use SAS labels as default column names
LABEL	Limit number of observations loaded into new file
LIST	List information about SAS variables to be loaded into new file
LOAD	Create a file and transfer data to it from an input SAS data set
PATH	Indicate the path and name of the PC file that you create and load
QUIT	Exit the procedure without further processing

Statement	Task
RENAME	Rename the PC File columns associated with the listed SAS variables
RESET	Reset column names, data types, and enable setting of null values
WHERE	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 data-set*.

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

File Type	Statement
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.

Aliases: 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-name*;

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 file name can vary with the operating environment.

See: SAS documentation for your operating environment for any restrictions.

Syntax

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

Required Arguments

'path and file name<.PC-file-extension>'
specifies the fully qualified path and file name. Enclose the entire path and file name in quotation marks, including the appropriate PC file extension. If you omit the file extension, SAS/ACCESS supplies it for you.

<'>file name<'>
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 file name includes characters that are invalid in a SAS name, or if it begins with a number, enclose the file name 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 DATA Step 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';
```

20

File-Specific Reference for the ACCESS and DBLOAD Procedures

Overview: ACCESS Procedure and DBLOAD Procedure **356**

ACCESS Procedure: XLS Files **356**
- Access Descriptors 356
- Arguments 357
- Syntax 357
- SAS Formats: XLS Specifics 358
- SAS Formats: Customized for XLS Strings 362

DBLOAD Procedure: XLS Specifics **363**
- DBLOAD Procedure Statements and Options 363
- DBLOAD Procedure Data Conversions for XLS Files 364
- Setting Environment Variables for XLS Files 367

ACCESS Procedure: WKn Specifics **369**
- ACCESS Procedure Syntax for WKn Files 369
- ACCESS Procedure Data Conversions for WKn Files 372

DBLOAD Procedure: WKn Specifics **373**
- DBLOAD Procedure Syntax for WKn Files 373
- DBLOAD Procedure Data Conversions for WKn Files 375
- Setting Environment Variables for WKn Files 376

ACCESS Procedure: DBF Specifics **377**
- ACCESS Procedure Syntax for DBF Files 377
- ACCESS Procedure Data Conversions for DBF Files 379

DBLOAD Procedure: DBF Specifics (Windows) **379**
- DBLOAD Procedure Syntax for DBF Files 379
- DBLOAD Procedure Data Conversions for DBF Files 381
- Setting Environment Variables for DBF Files 381

ACCESS Procedure: DIF Specifics **382**
- ACCESS Procedure Syntax for DIF Files 382
- ACCESS Procedure Data Conversions for DIF Files 384
- Datetime Conversions in the ACCESS Procedure 385

DBLOAD Procedure: DIF Specifics 385
DBLOAD Procedure Syntax for DIF Files 385
Datetime Conversions in the DBLOAD Procedure 387
Setting Environment Variables for DIF File Data Types 388

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.

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 367.

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-file-name.*XLS' | *<'>file name<'>* | *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;

QUIT;

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

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

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

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[1]	@[2]	CHAR	$*w.*
Numeric[3]	General	NUM	BEST
Numeric	0	NUM	*w.d*
Numeric	0.00	NUM	*w.d*
Numeric	#,##0	NUM	COMMA*w.d*
Numeric	#,##0.00	NUM	COMMA*w.d*
Numeric	#,##0_);(#,##0)	NUM	NEGPAREN*w.d*
Numeric	#,##0_);[Red](#,##0)	NUM	NEGPAREN*w.d*

1. Label data.
2. The XLS character format for Excel Version 5.
3. Number, formula, or missing data.

XLS File Data		SAS Variable Format	
Data Type	**XLS Format String**	**Type**	**Format**
Numeric	#,##0.00_); (#,##0.00)	NUM	NEGPAREN*w.d*
Numeric	#,##0.00_);[Red] (#,##0.00)	NUM	NEGPAREN*w.d*
Numeric	$#,##0_);($#,##0)	NUM	DOLLAR*w.d*
Numeric	$#,##0_);[Red] ($#,##0)	NUM	DOLLAR*w.d*
Numeric	($#,##0.00_); ($#,##0.00)	NUM	DOLLAR*w.d*
Numeric	($#,##0.00_);[Red] ($#,##0.00)	NUM	DOLLAR*w.d*
Numeric	_($*#,##0_);_($*(#,## #0);_($*"-"_);_(@_)	NUM	DOLLAR*w.d*
Numeric	_(*#,##0_);_(*(#,##0);_(*"-"_);_(@_)	NUM	NEGPAREN*w.d*
Numeric	_($*#,##0.00_);_($*(#,##0.00);_($*"-"?? _);_(@_)	NUM	DOLLAR*w.d*
Numeric	_(*#,##0.00_);_(*(#,# #0.00);_(*"-"?? _);_(@_)	NUM	NEGPAREN*w.d*
Numeric	0%	NUM	PERCENT*w.d*
Numeric	0.00%	NUM	PERCENT*w.d*
Numeric	0.00E+00	NUM	E*w.d*
Numeric	##0.0E+0	NUM	E*w.d*
Numeric	*m/d/yy*	NUM	MMDDYY*w.*
Numeric	*d-mmm-yy*	NUM	MMDDYY*w.*
Numeric	*d-mmm*	NUM	DATE*w.*

XLS File Data		SAS Variable Format	
Data Type	**XLS Format String**	**Type**	**Format**
Numeric	*mmm-yy*	NUM	MONYY*w.*
Numeric	*h:mm* AM/PM	NUM	TIME*w.*
Numeric	*h:mm:ss* AM/PM	NUM	TIME*w.*
Numeric	*h:mm*	NUM	TIME*w.*
Numeric	*hh:mm*	NUM	TIME*w.*
Numeric	*h:mm:ss*	NUM	TIME*w.*
Numeric	*hh:mm:ss*	NUM	TIME*w.*
Numeric	*m/d/yy h:mm*	NUM	DATETIME*w.*
Numeric	*ddmmmyy*	NUM	DATE*w.*
Numeric	*ddmmmyyyy:hh:mm:ss*	NUM	DATETIME*w.*
Numeric	*dd*	NUM	DATE*w.*
Numeric	*dd/mm/yy*	NUM	DDMMYY*w.*
Numeric	*dddd*	NUM	DATE*w.*
Numeric	*mm/dd/yy*	NUM	MMDDYY*w.*
Numeric	*mm:ss*	NUM	MMSS*w.*
Numeric	*mm yy*	NUM	MONYY*w.*
Numeric	*mm yyyy*	NUM	MONYY*w.*
Numeric	*mm:yy*	NUM	MONYY*w.*
Numeric	*mm:yyyy*	NUM	MONYY*w.*
Numeric	*mm-yy*	NUM	MONYY*w.*
Numeric	*mm-yyyy*	NUM	MONYY*w.*
Numeric	*mmyy*	NUM	MONYY*w.*

XLS File Data		SAS Variable Format	
Data Type	**XLS Format String**	**Type**	**Format**
Numeric	*mmyyyy*	NUM	MONYY*w.*
Numeric	*mm.yy*	NUM	MONYY*w.*
Numeric	*mm.yyyy*	NUM	MONYY*w.*
Numeric	*mm/yy*	NUM	MONYY*w.*
Numeric	*mm/yyyy*	NUM	MONYY*w.*
Numeric	*mmmm*	NUM	MONYY*w.*
Numeric	*m*	NUM	MONYY*w.*
Numeric	*mmmyy*	NUM	MONYY*w.*
Numeric	*mmmyyyy*	NUM	MONYY*w.*
Numeric	*dddd, mmmm dd, yyyy*	NUM	MONYY*w.*
Numeric	*dddd, dd mmmm yyyy*	NUM	MONYY*w.*
Numeric	*mmmm dd, yyyy*	NUM	MONYY*w.*
Numeric	*dd mmmm yyyy*	NUM	MONYY*w.*
Numeric	*yy*	NUM	YYMMDD*w.*
Numeric	*yyyy*	NUM	YYMMDD*w.*
Numeric	*yy mm*	NUM	YYMMDD*w.*
Numeric	*yyyy mm*	NUM	YYMMDD*w.*
Numeric	*yy:mm*	NUM	YYMMDD*w.*
Numeric	*yyyy:mm*	NUM	YYMMDD*w.*
Numeric	*yy-mm*	NUM	YYMMDD*w.*
Numeric	*yyyy-mm*	NUM	YYMMDD*w.*
Numeric	*yymm*	NUM	YYMMDD*w.*

XLS File Data		SAS Variable Format	
Data Type	**XLS Format String**	**Type**	**Format**
Numeric	*yyyymm*	NUM	YYMMDD*w.*
Numeric	*yy.mm*	NUM	YYMMDD*w.*
Numeric	*yyyy.mm*	NUM	YYMMDD*w.*
Numeric	*yy/mm*	NUM	YYMMDD*w.*
Numeric	*yyyy/mm*	NUM	YYMMDD*w.*
Numeric	*yy-mm-dd*	NUM	YYMMDD*w.*
Numeric	*yymmm*	NUM	YYMMDD*w.*
Numeric	*yyyymmm*	NUM	YYMMDD*w.*

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	DOLLAR*w.d*
Numeric	"E"	NUM	E*w.d*
Numeric	"*m*, *d* and *y*"	NUM	MMDDYY*w.*
Numeric	"*m* and *h*"	NUM	TIME*w.d*
Numeric	"*m* and *s*"	NUM	TIME*w.d*
Numeric	"*m* and *y*"	NUM	MONYY*w.*
Numeric	"*m*"	NUM	DATE*w.*
Numeric	"*d*"	NUM	DATE*w.*

XLS File Data		SAS Variable Format	
Data Type	**XLS Format String**	**Type**	**Format**
Numeric	*"y"*	NUM	DATE*w.*
Numeric	"0.0"	NUM	*w.d*
Numeric	Fraction values (#?/?)	NUM	BEST*w.d*
Numeric	Percent values (0.0%)	NUM	PERCENT*w.d*
Numeric	All others	NUM	BEST*w.d*

DBLOAD Procedure: XLS Specifics

DBLOAD Procedure Statements and Options

The DBLOAD procedure enables you to read data, format data, and set environment variables for XLS-specific data. See Chapter 19, "DBLOAD Procedure," on page 341.

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 "Supported Data Sources and Environments" on page 10.

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 367.

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 changes the format of SAS variables only 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
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

SAS Variable Format		XLS File Data	
Type	**Format**	**XLS Format String**	**Data Type**
NUM	JULDAY*w.*	*m/d/yy*	NUMBER
NUM	JULIAN*w.*	*m/d/yy*	NUMBER
NUM	MMDDYY*w.*	*mm/dd/yy*	NUMBER
NUM	MMSS*w.d*	*mm:ss*	NUMBER
NUM	MMYY*xw.*	*mm yy*	NUMBER
NUM	MMYYC	*mm:yy*	NUMBER
NUM	MMYYD	*mm-yy*	NUMBER
NUM	MMYYN	*mmyy*	NUMBER
NUM	MMYYP	*mm.yy*	NUMBER
NUM	MMYYS	*mm/yy*	NUMBER
NUM	MONNAME*w.*	*mmmm*	NUMBER
NUM	MONTH*w.*	*m*	NUMBER
NUM	MONYY*w.*	*mmmyy*	NUMBER
NUM	NEGPAREN*w.d*	#,##0_);(#,##0)	NUMBER
NUM	NENGO*w.*	*m/d/yy*	NUMBER
NUM	PERCENT*w.d*	0%	NUMBER
NUM	QTR*w.*	*m/d/yy*	NUMBER
NUM	QTRR*w.*	*m/d/yy*	NUMBER
NUM	SSN*w.*	000-00-0000	NUMBER
NUM	TIME*w.d*	*h:mm:ss*	NUMBER
NUM	TOD*w.*	*h:mm:ss*	NUMBER
NUM	W	0	NUMBER

SAS Variable Format		XLS File Data	
Type	**Format**	**XLS Format String**	**Data Type**
NUM	WEEKDATE*w.*	*dddd, mmmm dd, yyyy*	NUMBER
NUM	WEEKDATX*w.*	*dddd, dd mmmm yyyy*	NUMBER
NUM	WEEKDAY*w.*	m/d/yy	NUMBER
NUM	WORDDATE*w.*	*mmmmdd, yyyy*	NUMBER
NUM	WORDDATX*w.*	*dd mmmm yyyy*	NUMBER
NUM	YEAR*w.*	*yy* or *yyyy*	NUMBER
NUM	YYMM	*yy mm*	NUMBER
NUM	YYMMC	*yy:mm*	NUMBER
NUM	YYMMD	*yy-mm*	NUMBER
NUM	YYMMN	*yymm*	NUMBER
NUM	YYMMP	*yy.mm*	NUMBER
NUM	YYMMS	*yy/mm*	NUMBER
NUM	YYMMDD*w.*	*yy-mm-dd*	NUMBER
NUM	YYMON*w.*	*yymmm*	NUMBER
NUM	Z*w.d*	0*w.d*	NUMBER
NUM	FRACT*w.*	# ?/?	NUMBER

Excel column widths are set to *w* and are displayed in the column. If the data is larger than column width, it is displayed as number signs (###). In that case, you can view it by adjusting the column width.

Setting Environment Variables for XLS Files

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 generates 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: WKn Specifics

See the ACCESS procedure for PC files for general information about this feature. This is the WK<userSuppliedValue>n</userSuppliedValue>-specific syntax for the ACCESS procedure and the description of the ACCESS procedure data conversions.

ACCESS Procedure Syntax for WKn 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-file-name*<.WK1 | WK3 | WK4>*'* | <*'>file name<'>* | *fileref*;
RANGE *'range-name'* | *'range-address'*;
SCANTYPE<=>YES | NO | Y | N | *number-of-rows*;
SKIPROWS=*number-of-rows-to-skip*;
UPDATE *libref.member-name.*ACCESS | VIEW;
WORKSHEET<=>*worksheet-letter* | *worksheet-name*;

MIXED<=>YES | NO | Y | N;
QUIT;
RENAME *<'>column-identifier-1<'><=><'>SAS variable-name-1<'>* *<…<'>column-identifier-n<'><=><'>SAS variable-name-n<'>>*;
RESET ALL | *<'>column-identifier-1<'>* *<…<'>column-identifier-n<'>>*;
SELECT ALL | *<'>column-identifier-1<'>* *<…<'>column-identifier-n<'>>*;
SUBSET *selection-criteria*;
TYPE *<'>column-identifier-1<'>*<=> C |N *<…<'>column-identifier-n<'>>*<=>C | N;
UNIQUE<=>YES | NO | Y | N;
RUN;

The QUIT statement 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 WKn Files" on page 376.

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 WKn Files" on page 376.

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 WKn 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 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 Variable Formats for WKn 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	E*w.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.

WK*n* File Data				SAS Variable Format	
Data Type	**Data Format**	**Column Width**	**Decimal Number**	**Type**	**Format**
Numeric	MON-YY	*w*		NUM	MONYY5.
Numeric	MM-DD-YY	*w*		NUM	MMDDYY8.
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 WKn 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. WKn 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: WKn Specifics

See Chapter 19, "DBLOAD Procedure," on page 341 for more information about this feature. This is WK<userSuppliedValue>n</userSuppliedValue>-specific syntax for the DBLOAD procedure and description of DBLOAD procedure data conversions. See "DBLOAD Procedure Data Conversions for WKn Files" on page 375.

DBLOAD Procedure Syntax for WKn 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-file-name*<.WK1 | WK3 | WK4>' | *<'>filer name<'>* | *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;
QUIT;
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*>;
WHERE *SAS where-expression*;
RUN;

The QUIT statement 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*n*-specific statements are as follows:

PUTNAMES =YES | NO | Y | N;
writes column names to the first row of the new WK*n* 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*n* file. Data is written beginning in the first row of the WK*n* 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 WKn 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*n* 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		WK*n* File Data			
Type	**Data Format**	**Data Type**	**Column Format**	**Column Width**	**Number**
CHAR	$*w.*	LABEL	DEFAULT	*w*	
CHAR	$CHAR*w.*	LABEL	DEFAULT	*w*	
NUM	*w.d*	NUMBER	FIXED	*w*	*d*
NUM	F*w.d*	NUMBER	FIXED	*w*	*d*
NUM	E*w.d*	NUMBER	SCIENTIFIC	*w*	*d*
NUM	DOLLAR*w.d*	NUMBER	CURRENCY	*w*	*d*
NUM	PERCENT*w.d*	NUMBER	PERCENT	*w*	*d*
NUM	COMMA*w.d*	NUMBER	COMMA	*w*	*d*
NUM	BEST*w.*	NUMBER	DEFAULT	*w*	
NUM	BEST*w.*	NUMBER	GENERAL	*w*	
NUM	DATE5.	NUMBER	DD-MON	7	
NUM	DATE7.	NUMBER	DD-MON-YY	10	
NUM	MONYY5.	NUMBER	MON-YY	7	

SAS Variable Format		WK*n* File Data			
Type	**Data Format**	**Data Type**	**Column Format**	**Column Width**	**Number**
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, “ACCESS Procedure,” on page 313 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 *options*;
CREATE *libref.member-name*.ACCESS | VIEW;
PATH=*'path-and-file-name<.DBF>'* | *<'>file name<'>* | *fileref*;
UPDATE *libref.member-name*.ACCESS | VIEW;
ASSIGN=YES | Y | NO | 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<'>*;
QUIT;
RENAME *<'>column-identifier-1<'>=<'>SAS variable-name-1<'>*
<...<'>column-identifier-n<'> <=><'>SAS variable-name-n<'>>;
RESET ALL | *<'>column-identifier-1<'>* *<...<'>column-identifier-n<'>* >;
SELECT ALL | *<'>column-identifier-1<'>* *<...<'>column-identifier-n<'>>* ;
SUBSET *selection criteria*;
UNIQUE=YES | NO;

RUN ;

The QUIT statement causes the ACCESS 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(*n*)	$*n*.(*n*<= 200) $200. (*n*>200)
Numeric(*N*,*n*)	(*N*,*n*)
Float(*N*,*n*)[1]	(*N*,*n*)
Date	MMDDYY8.
Logical	$1.

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, "DBLOAD Procedure," on page 341 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 381.

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-file-name<.DBF>*' | *<'>file name<'>* | *fileref*;
VERSION=*dBASE-product-number*;
ACCDESC=*libref.access-descriptor*;
DELETE *variable-identifier-1* <*...variable-identifier-n*>;
ERRLIMIT=*error-limit*;
LABEL;
LIMIT=*load-limit*;
LIST ALL | FIELDS | *variable-identifier*;
LOAD;
QUIT;
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 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(*n*)
w.	NUMERIC
w.d.	NUMERIC
datetime*w.d*	DATE
date*w*.	DATE
e*w*.	FLOAT
binary*w*.	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 384.

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 *options*;
CREATE <*libref.*>*member-name.*ACCESS | VIEW;

The SAS/ACCESS interface to DIF uses the following procedure statements:

PROC ACCESS *options*;
CREATE *libref.member-name.*ACCESS | VIEW;
PATH=*'path-and-file-name<.DIF>'* | *<'>file name<'>* | *fileref*;
UPDATE *libref.member-name.*ACCESS | VIEW;
ASSIGN=YES | Y | NO | N;
DIFLABEL;
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<'>*;
QUIT;
RENAME *<'>column-identifier-1<'><=><'>SAS variable-name-1<'>*
<...*<'>column-identifier-n<'> <=><'>SAS variable-name-n<'>*>;
RESET ALL | *<'>column-identifier-1<'>* <...*<'>column-identifier-n<'>* >;

SELECT ALL | *<'>column-identifier-1<'>* *<...<'>column-identifier-n<'>>* ;
SKIPROWS=*number-of-rows-to-skip*;
SUBSET *selection criteria*;
TYPE *<'>variable-identifier-1<'>*<=>C | N
<...<'>variable-identifier-n<'><=>C | N '>;
UNIQUE=YES | NO;

RUN ;

The QUIT statement 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, "DBLOAD Procedure," on page 341for 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 387.

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-file-name'*<.DIF> | *<' >file name<' > | 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 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 DATETIME*w.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 you can then specify a date format for the column. DATE formats use only the integer portion of the data. Alternatively, you can also specify a TIME format that uses only 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 8

Appendixes

Appendix 1
DBF Procedure 391

Appendix 2
DIF Procedure 397

21

DBF Procedure

Overview: DBF Procedure ***391***
Overview: PROC DBF 391

Syntax: DBF Procedure ***392***
PROC DBF Statement 392

Usage: DBF Procedure ***393***
Converting DBF Fields to SAS Variables 393
Converting SAS Variables to DBF Fields 394
Transferring Other Software Files to DBF Files 394

Examples: DBF Procedure ***395***
Example 1: Converting a dBase II File to a SAS Data Set on UNIX 395
Example 2: Converting a SAS Data Set to a DBF File in a Microsoft Windows Environment 395
Example 3: Converting a SAS Data Set to a DBF File in a z/OS Environment 395

Overview: DBF Procedure

Overview: PROC DBF

The DBF procedure converts a dBase (DBF) file to a SAS data set, or it converts a SAS data set to a DBF file. The data sets are compatible with the current release of SAS software. You can use the DBF procedure under the UNIX, Microsoft Windows, IBM z/OS operating environments.

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. Future versions of dBase files might not be compatible with the current version of the DBF procedure.

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.

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.

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 license for SAS/ACCESS Interface to PC Files.

Syntax: DBF Procedure

PROC DBF
DB2 | DB3 | DB4 | DB5=*file name | fileref*
<DATA=*<libref.>SAS data set*>
<OUT=*<libref.>SAS data set*>;

PROC DBF Statement

Converts a dBase (DBF) file to a SAS data set or a SAS data set to a DBF file.

Syntax

PROC DBF DB2 | DB3 | DB4 | DB5=*file name | fileref*
<DATA=*<libref.>SAS data set*>
<OUT=*<libref.>SAS data set*>;

Required Argument

DB2 | DB3 | DB4 | DB5=*file name | fileref*
specifies the version of the dBase file and the file name or fileref of a .dbf file. It is recommended that you always use a FILENAME statement to define the physical file name with a logical name to avoid certain DOS naming limitations.

The DB*n* option must correspond to the version of dBase with which the .dbf file is compatible. The values are 2, 3, 4, or 5.

If you specify a file name, specify the file name without the .dbf extension, The file must be in the current directory. The file name must be in uppercase. The following PROC DBF statement creates the EMP.DBF file from the SAS data set MYLIB.EMPLOYEE:

```
PROC DBF DBF5=EMP data=mylib.employee;
RUN;
```

You cannot specify the file extension .dbf or a full path name:

```
PROC DBF DB5='/my/unix_directory/emp.dbf';
```

If you specify a fileref instead of a file name, the FILENAME statement must specify the file name with the .dbf extension. This example assigns the fileref MYREF to the MYFILE.DBF file:

```
FILENAME myref '/my_dir/myfile.dbf';
```

Optional Arguments

DATA=*<libref.>SAS data set*
specifies the name of the SAS data set used to create a .dbf file. Use this option to convert a SAS data set. to a .dbf file. If you use this option, do not use the OUT= option.

OUT=*<libref.>SAS data set*
specifies the name of the SAS data set you are creating from a .dbf file. Use this option to convert a .dbf file to a SAS data set. If you use this option, do not use the DATA= option.

If OUT= is omitted, SAS creates a temporary data set in the WORK library. The temporary data set is named *Data1, Data2,*, and so forth.

Usage: DBF Procedure

Converting DBF Fields to SAS Variables

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

Numeric field values in a .dbf file are stored in character form. .dbf numeric fields become SAS numeric variables with a length of 16. If a .dbf numeric value is missing, the corresponding dBase numeric field is filled with the character 9 by default.

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

data set is translated into a dBase file, any underscores in SAS variable names are changed to colons in dBase field names.

Converting SAS Variables to DBF Fields

Numeric field values in a .dbf file are stored in character form. SAS decimal values must be stored in a decimal format to be converted to a .dbf decimal value. (You can associate the SAS numeric variable with an appropriate SAS decimal format when you create the data set or by using the DATASETS procedure.) The corresponding .dbf field does not have any value to the right of the decimal point.

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, up to 255 characters. 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 a Microsoft Excel .xls file to a .dbf file by selecting **File** ⇨ **Save as** from within an Excel spreadsheet. Select the (uppercase) .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: DBF Procedure

Example 1: Converting a dBase II File to a SAS Data Set on UNIX

The dBase II file named EMPLOYEES_START_Q4.DBF is converted to a SAS data set, SAVE.EMPLOYEESQ4. You must specify the FILENAME statement before the PROC DBF statement.

```
LIBNAME save '/hr/employees_thisyear';
FILENAME empQ4 '/hr/personnel/employees_start_Q4.dbf';
PROC DBF DB2=empQ4 OUT=save.employeesq4;
RUN;
```

Example 2: Converting a SAS Data Set to a DBF File in a Microsoft Windows Environment

In this example, the SAS data set MYLIB.EMPLOYEE is converted to a dBase 5 file. The FILENAME statement specifies the name of the .dbf file.

```
LIBNAME mylib 'c:\my\directory';
FILENAME employee 'c:\sasdemo\EMPLOYEE.DBF';
PROC DBF DB5=employee DATA=mylib.employee;
RUN;
```

Example 3: Converting a SAS Data Set to a DBF File in a z/OS Environment

This example is the same as the previous example except it is created 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;
```

22

DIF Procedure

Overview: DIF Procedure ***397***
Overview: PROC DIF 397
Syntax: DIF Procedure ***398***
PROC DIF Statement 398
Usage: DIF Procedure ***400***
Converting DIF Variables to SAS Variables 400
Transferring SAS Data Sets to and from Other Software Products Using DIF 400
Missing Values 401
Examples: DIF Procedure ***401***
Example 1: Converting a DIF File to a SAS Data Set 401
Example 2: Converting a SAS Data Set to a DIF File 401
Example 3: Converting a SAS Data Set to a DIF File in the UNIX Environment . . . 402

Overview: DIF Procedure

Overview: PROC DIF

The DIF procedure converts a DIF file to a SAS data set, or it converts a SAS data set to a DIF file. You can use the DIF procedure under the UNIX and Microsoft Windows operating environments.

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. The procedure also converts SAS data sets to DIF files.

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

Software Arts, Inc. developed the data interchange format to be used as a common language for data. Originally, DIF was made popular by products such as Lotus 1-2-3 and VisiCalc. Although DIF is not as popular today, 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 license for SAS/ACCESS Interface to PC Files.

Syntax: DIF Procedure

PROC DIF
DIF=*file name | fileref*
<DATA=*<libref.>SAS data set*>
<OUT=*<libref.>SAS data set*>
<LABELS>
<PREFIX=*name*>
<SKIP=*n*>;

PROC DIF Statement

Converts a DIF file to a 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 DIF=*file name | fileref*
<DATA=*<libref.>SAS data set*>
<OUT=*<libref.>SAS data set*>
<LABELS>
<PREFIX=*name*>
<SKIP=*n*>;

Required Argument

DIF=*file name | fileref*
specifies the file name or fileref of a DIF file. You use the FILENAME statement to define a fileref. It is recommended that you always use a FILENAME statement to define the physical file name with a logical name to avoid certain DOS naming limitations.

If you specify a file name, 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 SAS data set MYLIB.EMPLOYEE:

```
PROC DIF dif=emp data=mylib.employee;
RUN;
```

You cannot specify EMP.DIF or a full pathname:

```
PROC DIF dif='/my/unix_directory/emp.dif';
RUN;
```

If you specify a fileref instead of a file name, the FILENAME statement must specify the file name plus a .dif extension:

```
FILENAME myref '/my_dir/myfile.dif';
```

Optional Arguments

DATA=*<libref.>SAS data set*
specifies the name of the SAS data set used to create a DIF file. Use this option to convert a SAS data set to a DIF file. If you use this option, do not use the OUT= option.

OUT=*<libref.>SAS data set*
specifies the name of the SAS data set you are creating from a DIF file. Use this option to convert a DIF file to a SAS data set. If you use this option, do not use the DATA= option.

If OUT= is omitted, SAS creates a temporary data set in the WORK library. The temporary data set is named Data1, Data2, and so on.

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, and so on.

Default If you omit the PREFIX= option, PROC DIF assigns the names Col1, Col2, and so on.

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.

Usage: DIF Procedure

Converting DIF Variables to SAS Variables

Character variables (sometimes referred to as *string-values*) in a DIF file 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 into 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, and then 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: DIF Procedure

Example 1: Converting a DIF File to a SAS Data Set

A DIF file named EMPLOYEES_START_Q4.DIF is converted to a SAS data set, SAVE.EMPLOYEESQ4. Specify the FILENAME statement before the PROC DIF statement.

```
LIBNAME save '/hr/employees_thisyear';
FILENAME empQ4 '/hr/personnel/employees_start_Q4.dif';
PROC DIF DIF=empQ4 OUT=save.employeesq4;
RUN;
```

Example 2: Converting a SAS Data Set to a DIF File

A SAS data set named SAVE.EMPLOYEE is converted to a DIF file. A FILENAME statement is used to specify a fileref that names the DIF file.

```
FILENAME employee 'c:\sasdemo\employee.dif';
PROC DIF DIF=emp DATA=save.employee;
RUN;
```

Example 3: Converting a SAS Data Set to a DIF File in the UNIX Environment

This example is like the previous one except that it is coded for the UNIX environment. A SAS data set named SAVE.EMPLOYEE is converted to a DIF file. A FILENAME statement is used to specify a fileref that names the DIF file.

```
FILENAME emp '/sasdemo/emp_UNX.dif';
PROC DIF DIF=emp DATA=save.employee;
RUN;
```