Contents

Early Adopter Software .......................................................... v
Using This Document ............................................................ vii

PART 1  Introduction to SAS Viya: SAS/ACCESS Interface to PC Files  1

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

PART 2  Import and Export Procedures  5

Chapter 2 • Supported Data Sources and Environments ................................................. 7
   Reading and Writing Data between SAS Data Sets and PC Files ..................... 7
   Supported Data Sources and Environments ......................................... 7

Chapter 3 • IMPORT Procedure ........................................................................... 9
   Overview: PROC IMPORT ......................................................... 9
   Syntax: IMPORT Procedure .................................................... 10

Chapter 4 • EXPORT Procedure ........................................................................ 15
   Overview: PROC EXPORT ..................................................... 15
   Syntax: EXPORT Procedure .................................................... 16

Chapter 5 • File Format-Specific Reference for the IMPORT and EXPORT Procedures .... 21
   Delimited Files ........................................................................ 22
   Microsoft Excel Workbook Files ................................................ 28
   dBase DBF Files ..................................................................... 39
   dBase DBF MEMO Files .......................................................... 43
   JMP Files ................................................................................ 44
   Paradox DB File Formats .......................................................... 47
   SPSS SAV Files ...................................................................... 47
   Stata DTA Files ...................................................................... 50

PART 3  SAS LIBNAME Statement for the XLSX and JMP Engines on Linux  53

Chapter 6 • SAS LIBNAME Statement for the XLSX Engine for Microsoft Excel on Linux .... 55
   Dictionary ................................................................................ 55
# Contents

**Chapter 7 • SAS LIBNAME Statement for the JMP Engine on Linux** .......................... 61
  Dictionary ................................................................. 61

**PART 4  Appendixes** 63

**Appendix 1 • DBF Procedure** ................................................................. 65
  Overview: PROC DBF .......................................................... 65
  Syntax: DBF Procedure ....................................................... 66
  Converting DBF Fields to SAS Variables .................................. 67
  Converting SAS Variables to DBF Fields ................................ 67
  Transferring Other Software Files to DBF Files ....................... 67
  Example: Converting a dBase II File to a SAS Data Set on UNIX ... 68

**Recommended Reading** .............................................................. 69

**Index** .................................................................................. 71
Using This Document

Audience

This document is intended for applications programmers and users who know how to use their operating environment, PC files, and basic SAS commands and statements. This document provides a general reference, as well as specific details, and SAS code examples that show how to access and use data in PC files from a SAS session running on Linux.
Part 1

Introduction to SAS Viya: SAS/ACCESS Interface to PC Files

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

Methods for Accessing PC Files Data ................................. 3
Using This Document ......................................................... 4
Sample Data in This Document ............................................ 4

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 XLS and XLSX files as well as in other PC file formats.

The IMPORT and EXPORT procedures are part of SAS software. You can use these procedures without a license for SAS/ACCESS Interface to PC files. For SAS, access is available only to JMP 7 or later files or to CSV, TXT, and other delimited files. You can also use other SAS methods to import delimited data, such as the SAS \texttt{INFILE} statement.

The IMPORT and EXPORT procedures facilitate data transfer between SAS data sets and several PC file formats. You can access data from the following formats:

- Delimited files
- Microsoft
- DBF
- DBF MEMO
- JMP
- SPSS
- Stata
- Paradox

Not every PC file format is available under the Linux environment.

You can use the DBF procedure to convert formatted data between dBase (DBF) files and SAS data sets. For more information, see Appendix 1, “DBF Procedure,” on page 65.
Using This Document

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

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

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

Sample Data in This Document

Examples in this document show how you can use SAS/ACCESS Interface to PC Files to read and write PC file data directly from SAS programs. They are not meant as examples for you to follow in designing files for any purpose. PCFILES Samples are being installed in location SASHOME\version\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 Procedures

Chapter 2
Supported Data Sources and Environments ........................................ 7

Chapter 3
IMPORT Procedure ............................................................................ 9

Chapter 4
EXPORT Procedure ........................................................................... 15

Chapter 5
File Format-Specific Reference for the IMPORT and EXPORT Procedures .................................................. 21
Chapter 2
Supported Data Sources and Environments

Reading and Writing Data between SAS Data Sets and PC Files

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 3, “IMPORT Procedure,” on page 9 and Chapter 4, “EXPORT Procedure,” on page 15.

Supported Data Sources and Environments

Supported Data Sources and Environments

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

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

Table 2.1  Data Source Summary for Linux Environment

<table>
<thead>
<tr>
<th>Data Source</th>
<th>Identifier</th>
<th>PROC IMPORT</th>
<th>PROC EXPORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Excel using XLSX file formats</td>
<td>XLSX</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Excel using XLS file formats</td>
<td>XLS</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>dBase using DBF file formats</td>
<td>DBF</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Data Source</td>
<td>Identifier</td>
<td>PROC IMPORT</td>
<td>PROC EXPORT</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>------------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>dBase file format with MEMO support</td>
<td>DBFMEMO</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>JMP using Version 7 or later JMP file formats</td>
<td>JMP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Paradox using DB file formats</td>
<td>DB</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SPSS using SAV file formats</td>
<td>SAV</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Stata using DTA file formats</td>
<td>DTA</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Comma-separated file</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Tab-separated file</td>
<td>TAB</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Delimiter-separated file</td>
<td>DLM</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
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 Excel workbooks
- Paradox files
- SPSS files
- Stata files
- dBase (DBF and DBF MEMO files)
- JMP files
- delimited files

Delimited files contain columns of data values that are separated by a delimiter, such as a blank, a comma, or a tab. By default, PROC IMPORT reads delimited files as varying record-length files.

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

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

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

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 23.
Syntax: IMPORT Procedure

PROC IMPORT
DATAFILE="filename" | DATATABLE="tablename"
<option(s)>
;

PROC IMPORT Statement

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

See: "IMPORT" in SAS Viya Utility Procedures Guide

Syntax

PROC IMPORT
DATAFILE="filename" | DATATABLE="tablename" (Not used for Microsoft Excel files)
<DBMS=data-source-identifier>
;

Required Arguments

DATAFILE="filename" | "fileref"

specifies the complete path and filename or fileref for the input file. A fileref is a SAS name that is associated with the physical location of the output file. To assign a fileref, use the FILENAME= statement.

You can omit the quotation marks if the filename does not include certain special characters, such as a backslash or spaces.

Alias

FILE

Default

character

Restrictions

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

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

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 data type (character) 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.) 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 Viya 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.

Alias
TABLE

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

Optional Arguments

SAS data-set-option(s)
Specify SAS data set options. For example, to assign a password to the resulting SAS data set, you can use the ALTER= , PW= , READ= , or WRITE= data set options. To import only data that meets a specified condition, you can use the WHERE data set option. For information about all SAS data set options, see SAS Viya 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 3.1 on page 11. DBMS= specifications refer to local access, except where noted in this table.

Note: Transcoding is not supported for DBMS=XLS. The output yields unpredictable results. Use DBMS=XLSX for transcoding.

Table 3.1  DBMS Specifications

<table>
<thead>
<tr>
<th>Data Source Identifier</th>
<th>Output Data Source</th>
<th>File Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSV</td>
<td>Delimited file with comma-separated values</td>
<td>.csv</td>
</tr>
<tr>
<td>DBF</td>
<td>dBASE 5.0, IV, III+, and III files</td>
<td>.dbf</td>
</tr>
</tbody>
</table>
**Table 3.2 Microsoft Excel Workbook Specifications**

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Excel 2007 and later</th>
<th>Excel 97, 2000, 2002, 2003</th>
<th>Excel 5.0, 95</th>
<th>Excel 4.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>XLS</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>XLSX</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

The following table lists the DBMS, the software that it uses, any required software, and the operating platform that can use the DBMS.

**Note:** All DBMS= specifications refer to local access.

**Microsoft Excel**

When you specify DBMS=XLS or DBMS=XLSX for an Excel file, you can read and write to Excel workbooks under Linux directly. The following example demonstrates the use of DBMS=XLSX specifying a range of cells.

```plaintext
proc import datafile="fieldtypes.xlsx"
  out=small dbms=xlsx;
  range=colsb_d;
run;
```
**Table 3.3  DBMS Specifications for Excel**

<table>
<thead>
<tr>
<th>DBMS</th>
<th>Uses</th>
<th>Requires</th>
<th>Operating Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>XLS</td>
<td>File formats technology</td>
<td></td>
<td>Microsoft Windows, Linux, UNIX</td>
</tr>
<tr>
<td>XLSX</td>
<td>File formats technology</td>
<td></td>
<td>Microsoft Windows, Linux, UNIX</td>
</tr>
</tbody>
</table>

**Table 3.4  DBMS Data Source Identifiers**

<table>
<thead>
<tr>
<th>DBMS=</th>
<th>Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>XLS</td>
<td>yes</td>
</tr>
<tr>
<td>XLSX</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Note**  Transcoding is not supported for DBMS=XLS. The output yields unpredictable results. Use DBMS=XLSX for transcoding.

**Tip**  DBMS=XLS does not support multi-byte characters. Save the spreadsheets as XLSX if you have multi-byte characters.
Overview: PROC EXPORT

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

- Microsoft Excel workbook files
- Paradox files
- SPSS files
- Stata files
- dBase files (DBF and DBF MEMO)
- JMP files
- Delimited files

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

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

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

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

PROC EXPORT DATA=libref.SAS data set
OUTFILE="filename"
<option(s)>;

PROC EXPORT Statement

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

See: “EXPORT” in SAS Viya Utility Procedures Guide

Syntax

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

Required Arguments

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

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

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

OUTFILE="filename" | "fileref"

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

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

Optional Arguments

<\textit{SAS data-set-option(s)}> specifies SAS data set options. For example, if the data set that you are exporting has an assigned password, you can use the ALTER= option, the PW= option, the READ= option, or the WRITE= option. To export only data that meets a specified condition, you can use the WHERE= data set option. 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 \textit{SAS Viya Data Set Options: Reference}.

\textbf{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.

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

\begin{table}
\centering
\begin{tabular}{|l|l|l|}
\hline
\textbf{Data Source Identifier} & \textbf{Output Data Source} & \textbf{File Extension} \\
\hline
CSV & delimited file (comma-separated values) & .csv \\
\hline
DBF & dBASE 5.0, IV, III+, and III files & .dbf \\
\hline
DBFMEMO & dBASE 5.0, IV, III+, and III files with memos \\
& FoxPro and VisualPro with memos & .dbf, .fpt \\
\hline
DLM & delimited file (default delimiter is a blank) & . \\
\hline
DTA & Stata file & .dta \\
\hline
JMP & JMP files, Version 7, and later format & .jmp \\
\hline
PARADOX & Paradox DB files & .db \\
\hline
SAV & SPSS files, compressed and uncompressed binary files & .sav \\
\hline
TAB & delimited file (tab-delimited values) & .txt \\
\hline
XLS & Excel 97, 2000, 2002, or 2003 spreadsheet (using file formats) & .xls \\
\hline
\end{tabular}
\caption{Data Source Identifier Summary}
\end{table}

Note: Transcoding is not supported for DBMS=XLS. The output yields unpredictable results. Use DBMS=EXCEL as an alternative.
When you specify a value for DBMS=, consider the following for specific data sources:

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

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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>.xlsx</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>.xls</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

- 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.

<table>
<thead>
<tr>
<th>DBMS=</th>
<th>Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>XLS</td>
<td>yes</td>
</tr>
<tr>
<td>XLSX</td>
<td>yes</td>
</tr>
</tbody>
</table>

Restriction

Only Excel 2007 and later can use the .xlsx format.

Notes

Transcoding is not supported for DBMS=XLS. The output yields unpredictable results. Use DBMS=XLSX for transcoding.

Tip

DBMS=XLS does not support multi-byte characters. Save the spreadsheets as XLSX if you have multi-byte characters.

See

Chapter 5, “File Format-Specific Reference for the IMPORT and EXPORT Procedures,” on page 21
**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 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: Export SAS Data Sets to Excel 2010 Workbook and Replace Sheets” on page 38.

<file-format-specific-statements>

See Chapter 5, “File Format-Specific Reference for the IMPORT and EXPORT Procedures,” on page 21 for the supported syntax for your DBMS.
Chapter 5
File Format-Specific Reference for the IMPORT and EXPORT Procedures

Delimited Files ................................................................. 22
  Overview .................................................................. 22
  CSV Files .................................................................. 23
  Tab-Delimited Files .................................................... 23
  Other Delimiters ......................................................... 23
  Sharing Delimited Files across Hosts ............................ 23
  Processing Delimited Files in SAS ................................. 23
  IMPORT and EXPORT Procedure Statements for Delimited Files .................................................................. 24
  Example 1: Import a Tab-Delimited File into SAS ............ 26
  Example 2: Import a Space-Delimited File into SAS ......... 27
  Example 3: Import a CSV File That Has a Single Quotation Mark in a Name .............................................. 27
  Example 4: Export a SAS Data Set to a CSV File ............. 27
  Example 5: Import a Subset of a CSV File into SAS ......... 27
  Example 6: Export a SAS Data Set That Has a Single Quotation-Mark Name ........................................... 28

Microsoft Excel Workbook Files .................................... 28
  Microsoft Excel Files Essentials ................................... 28
  Excel Data Types .......................................................... 30
  Excel Numeric Data and Time Values ............................ 30
  Excel File Formats ....................................................... 30
  Processing XLSX and XLS Files in SAS ........................... 31
  Import and Export Microsoft Excel Files Using XLS and XLSX Drivers ......................................................... 33
  Example 1: Export SAS Data Sets to Excel 2010 Workbook and Replace Sheets ........................................... 38
  Example 2: Export SAS Data Sets to Excel 2010 Workbook and Add a New Sheet ........................................ 38
  Example 3: Import Data Using a Range Name .................. 38
  Example 4: Import Data Using an Absolute Range Address .. 39

dBase DBF Files ................................................................. 39
  dBase DBF Files Essentials ........................................... 39
  DBF Data Types ............................................................ 39
  Setting Environment Variables and System Options ........... 41
  Supported SAS IMPORT and EXPORT Procedure Statements ................................................................. 41
  Example 1: Export Data to a DBF File from a SAS Data Set .................................................................. 42
  Example 2: Import Data from a DBF File into a SAS Data Set ............................................................... 43
  Example 3: Export Data to a DBF File from a SAS Data Set Using Encoding ............................................ 43
  Example 4: Import and Translate Data from a DBF File ............................................................................ 43

dBase DBF MEMO Files ......................................................... 43
  Overview .................................................................. 43
  Import Data from a DBF File with Memo Field into a SAS Data Set ............................................................. 44
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.

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

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

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

Tab-Delimited Files

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

Other Delimiters

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

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. The default end-of-row delimiter for Linux 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 CSV, tab, or other character-delimited file, the procedure does the following 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.

The following 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 24 to do the following:

• 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 file types are CSV (comma-separated values), Tab (tab-separated values), and DLM (delimiter-separated values).

See: “Example 1: Import a Tab-Delimited File into SAS” on page 26 and “Example 4: Export a SAS Data Set to a CSV File” on page 27.

**Table 5.1 IMPORT and EXPORT Procedure Statements**

<table>
<thead>
<tr>
<th>Delimited File Type</th>
<th>Statement Options</th>
<th>PROC IMPORT</th>
<th>PROC EXPORT</th>
<th>Valid Value</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSV and TAB</td>
<td>DATAROW</td>
<td>Yes</td>
<td>No</td>
<td>1 to 2147483647</td>
<td>Depends on GETNAMES= option value</td>
</tr>
<tr>
<td></td>
<td>GETNAMES</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>GUESSINGROWS</td>
<td>Yes</td>
<td>No</td>
<td>1 to 2147483647</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>PUTNAMES</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
**Delimited Files**

<table>
<thead>
<tr>
<th>Delimited File Type</th>
<th>Statement Options</th>
<th>PROC IMPORT</th>
<th>PROC EXPORT</th>
<th>Valid Value</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DLM</td>
<td>DATAROW</td>
<td>Yes</td>
<td>No</td>
<td>1 to 2147483647</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>DELIMITER</td>
<td>Yes</td>
<td>Yes</td>
<td>'char'</td>
<td>'nn'x</td>
</tr>
<tr>
<td></td>
<td>GETNAMES</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>GUESSINGROWS</td>
<td>Yes</td>
<td>No</td>
<td>1 to 2147483647</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>PUTNAMES</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

**DATAROW=n**
specifies the row number where the IMPORT procedure starts reading data.

- **Default**
  - When GETNAMES=NO: 1; when GETNAMES=YES: 2
- **Range**
  - 1 to 2147483647
- **Restrictions**
  - If GETNAMES=YES, then DATAROW must be greater than or equal to 2.
  - If GETNAMES=NO, then DATAROW must be greater than or equal to 1.

**DELIMITER='char' | 'nn'x**
specifies the delimiter (either a single character or hexadecimal value) that separates the columns of data for the IMPORT and EXPORT procedures. 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.

**Example**
“Exporting to a Delimited External Data Source” in *SAS Viya Utility 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.

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.)

The default row number can be changed in the SAS registry using the REGISTRY procedure.

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 SAS Viya Utility 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 tab character (‘09’x) as the delimiter.

```
PROC IMPORT OUT=WORK.TEST
FILE="&dlmdir.\invoice.txt"
DBMS=TAB REPLACE;
GETNAMES=YES;
DATAROW=2;
GUESSINGROWS=10;
RUN;
```
Example 2: Import a Space-Delimited File into SAS

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

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

Example 3: Import a CSV File That Has a Single Quotation Mark in a Name

You can import a CSV file into SAS, even if the name contains a single quotation mark and 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 SAS data set name.

```
OPTIONS VALIDMEMNAME=EXTEND;
PROC IMPORT OUT=WORK."It's the Night Before Christmas Game"
  DATAFILE="c:\temp\game_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="c:\temp\CustFile.csv"
      DBMS=DLM REPLACE;
      DELIMITER=',';
      GETNAMES=NO;
      DATAROW=6;
      OBS=10;
RUN;
```
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.

```sas
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 Excel 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 collectively in this document as XLS or XLSX files.

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 two periods. For example, the range B2:E8 is the range address for a rectangular block of 12 cells, where the top left cell is B2 and the bottom right cell is E8 (shown as shaded, below). A range name identifies a range. A range is equivalent to a SAS table or SAS data set.
A worksheet in a .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 extension.

Figure 5.1  A Range of Data in an Excel Worksheet

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 the following 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 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 and 2010 have several different file extensions, but this release supports only files with the .xlsx file extension.
**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 characters.

- Numeric data can be numbers, formulas, or error values. Numeric data can include numbers (0 through 9), formulas, or error values (such as #NULL!, #N/A, #VALUE!).

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

Use a CURRENCY data type to prevent rounding during calculations. A CURRENCY type stores 8-byte numbers with a precision up to four decimal digits. Hence, when you read an Excel file into SAS, currency values have a precision up to four decimal places.

**Excel Numeric Data and Time Values**

Numeric data can also include date and time values. The conversion of date and time values between SAS data sets and Microsoft Excel spreadsheets is transparent to users. However, you are encouraged to understand the differences between them.

In Microsoft Excel software, a date value is the integer portion of a number that can range from 01 January 1900 (saved as integer value: 1) to 31 December 9999 (saved as integer value: 2,958,465). A Microsoft Excel software time value is the decimal portion of a number that represents time as a proportion of a day. For example, 0.0 is midnight, 0.5 is noon, and 0.999988 is 23:59:59 (on a 24-hour clock). 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 the following Excel formats:

Overview
SAS Import and Export Utilities provide only the XLSX engine to access Microsoft Excel files.

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

Be aware that the character set (“encoding”) that you use must match the set for your DBMS, 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 DBMS like Microsoft Excel 2010. 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.

Import and Export Procedure Statements for XLS and XLSX Files

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.

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.

The following 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 32.

Restriction Supported for only the IMPORT procedure.

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 log indicates that the data is deleted but that the worksheet name still exists.

The *sheet-name* can contain up to 31 characters. The following 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'"; .

**TEXTSIZE=1 to 32767**

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

**Alias** DBMAX_TEXT

**USEDATE=YES | NO**

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

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

NO does not specify the DATE9. format for the corresponding date column in the Microsoft Excel table. If your data includes time values, specify USEDATE=NO.

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

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

```sas
LIBNAME SDF "&sasdir";
PROC EXPORT DATA=SDF.ORDERS
    FILE='c:\temp\demo.xlsx'
    DBMS=XLSX REPLACE;
    SHEET='Orders';
RUN;
```

**Import and Export Microsoft Excel Files Using XLS and XLSX Drivers**

The IMPORT and EXPORT procedures use the XLS driver to read and write XLS file formats directly. The driver supports Excel versions 5/95, 97, 2000, 2002, and 2003. It is available on Linux.

The IMPORT and EXPORT procedures use the XLSX driver to read and write XLSX file formats directly. The driver supports only XLSX. XLSX is the recommended driver to use (rather than XLS). The XLSX driver is available on Linux.
Note: When you open a workbook in Excel (XLSX file), 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 that uses XLS and XLSX drivers.

Table 5.2 Available Statements for Importing and Exporting Excel Files Using the Translation Driver

<table>
<thead>
<tr>
<th>Data Source</th>
<th>Syntax</th>
<th>Valid Value</th>
<th>Default Value</th>
<th>PROC IMPORT</th>
<th>PROC EXPORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>XLS</td>
<td>ENDCOL</td>
<td>Last column for data</td>
<td>Last column that contains data</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>ENDNAMEROW</td>
<td>Last row for variable names</td>
<td>Same as NAMEROW</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>ENDROW</td>
<td>Last row for data</td>
<td>Last row that contains data</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>GETNAMES</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>NAMEROW</td>
<td>First row for variable names</td>
<td>First row that contains variable names</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>PUTNAMES</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>RANGE</td>
<td>NAME</td>
<td>SHEET$UL-LR</td>
<td>First row</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>SHEET</td>
<td>Sheet name</td>
<td>First sheet</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>STARTCOL</td>
<td>First column for data</td>
<td>Last column that contains data</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>STARTROW</td>
<td>First row for data</td>
<td>First row that contains data</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>XLSX</td>
<td>GETNAMES</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>RANGE</td>
<td>NAME</td>
<td>SHEET$UL-LR</td>
<td>First row</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>SHEET</td>
<td>Sheet name</td>
<td>First sheet</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**DATAROW=n**

specifies the row number where the IMPORT procedure starts reading data. For XLS and XLSX files, it is recommended that you use the RANGE= statement rather than DATAROW=. For more information, see the RANGE= statement on page 36.

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.

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 column of the imported file. If data in the first column 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**.

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.

### GETNAMES=YES  |  XLSX Result  |  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 column 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= was commented out.

**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.

**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 31.

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 37.

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 log indicates that the data is deleted but that the worksheet name still exists.

The sheet-name can contain up to 31 characters. The following 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 the following: ~, !, %, ^, &, (, ), +, {, }. 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, .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.

STARTROW= start-row
specifies the first row for data.
Default     The first row that contains data.

Restriction Available only for DBMS=XLS for backward compatibility.

Example 1: Export SAS Data Sets to Excel 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 2: 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 3: Import Data Using a Range Name

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

PROC IMPORT OUT=WORK.INVOICE
   FILE="&demodir.demo.xls"
   DBMS=XLS REPLACE;
Example 4: Import Data Using an Absolute Range Address

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

```sas
PROC IMPORT OUT=WORK.INVOICE
  FILE="&demodir.demo.xls"
  DBMS=XLS REPLACE;
  RANGE="Invoice$B4:D10";
  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 43.

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

A DBF file contains data that is organized in a tabular format of database fields and records. Each database field can contain one type of data, and each record can hold one data value for each field. This display shows four database fields from Customer.DBF and highlights a database field and a record.

**Figure 5.2 Database Field and Record**

<table>
<thead>
<tr>
<th>database field</th>
</tr>
</thead>
<tbody>
<tr>
<td>CITY</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>San Jose</td>
</tr>
<tr>
<td>Memphis</td>
</tr>
<tr>
<td>Rockville</td>
</tr>
<tr>
<td>La Rochelle</td>
</tr>
<tr>
<td>Buenos Aires</td>
</tr>
<tr>
<td>Singapore</td>
</tr>
</tbody>
</table>

**DBF Data Types**

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

*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 254 characters. Characters can be letters, digits, spaces, or special characters.

ALIAS: CHAR

NUMERIC (N, n)

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

Table 5.3  dBASE Maximum Numeric Values

<table>
<thead>
<tr>
<th>dBASE Version</th>
<th>Maximum Numeric (N, n) Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>dBASE II</td>
<td>16, 14</td>
</tr>
<tr>
<td>dBASE III</td>
<td>19, 15</td>
</tr>
<tr>
<td>dBASE III PLUS</td>
<td>19, 15</td>
</tr>
<tr>
<td>dBASE IV</td>
<td>20, 18</td>
</tr>
<tr>
<td>dBASE 5.0</td>
<td>20, 18</td>
</tr>
</tbody>
</table>

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, use the following code:

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

Supported SAS IMPORT and EXPORT Procedure Statements

The IMPORT | EXPORT method uses DBF file formats to access data in DBF Files on Linux.

The method imports data from DBF files in versions 3, 4, and 5 formats. It 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 DBF File from a SAS Data Set” on page 42 for additional information.

Table 5.4  IMPORT and EXPORT Procedure Statements for DBF Files

<table>
<thead>
<tr>
<th>Data Source</th>
<th>Syntax</th>
<th>Valid Values</th>
<th>Default Value</th>
<th>PROC IMPORT</th>
<th>PROC EXPORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBF</td>
<td>DBENCODING</td>
<td>Encoding-value</td>
<td>Current SAS session encoding</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>GETDELETED</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

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


GETDELETED=YES | NO

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

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

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

Alias   GETDEL

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

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

```sas
LIBNAME SDF "&sasdir";
PROC EXPORT DATA=SDF.EMPLOYEE (WHERE=(HIREDATE is not missing))
   OUTFILE="&tmpdir.test.dbf"
   DBMS=DBF REPLACE;
RUN;
```
Example 2: Import Data from a DBF File into a SAS Data Set

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

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

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

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

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

Example 4: Import and Translate Data from a DBF File

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

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

dBase DBF MEMO Files

Overview

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

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

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

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

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

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

---

**JMP Files**

**JMP File Essentials**

A JMP file is a file format that is created using JMP software. 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/](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.

SAS supports access to JMP files. Therefore, you can access JMP files without a license for SAS/ACCESS Interface to PC Files. JMP files can be used by the following SAS language elements: a LIBNAME statement, the IMPORT and EXPORT procedures, and the Import and Export Wizard.

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. Characters can be letters, digits, spaces, or special characters.
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 *SAS Viya Utility Procedures Guide*.

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.

TIME

specifies the time format. When you are reading data, the time values are mapped to a SAS number and scaled to the base time. The JMP time display format maps to the appropriate SAS time display format. When you are writing data, the SAS output format for the numeric variable is checked to determine whether it is a time format.
If so, the SAS numeric value is scaled to a JMP time value with the appropriate time display format.

**Importing and Exporting JMP Files Data**

SAS 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.

SAS IMPORT | EXPORT utilities provide the DBMS=JMP method for accessing JMP files.

**JMP File Formats (DBMS= JMP)**

This IMPORT | EXPORT method uses JMP file formats (JMP 7 and later) to access data in JMP files on Linux, UNIX, and Microsoft Windows operating platforms.

**IMPORT Procedure and EXPORT Procedure Supported Syntax**

\[ \text{FMTLIB=} \text{libref.format-catalog} \]

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.

\[ \text{META=} \text{libref.member-data-set; } \]

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: Export a SAS Data Set to a JMP File**

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

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

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

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

```
PROC IMPORT OUT=WORK.CUSTOMER
   FILE="&jmpdir.customer.jmp"
   DBMS=JMP REPLACE;
RUN;
```
Paradox DB File Formats

Paradox File Essentials

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

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

Export a SAS Data Set to a PARADOX DB File

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

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

Import a SAS Data Set from a Paradox DB File

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

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

SPSS SAV Files

SAV File Essentials

SAS/ACCESS supports SPSS files created with version 18 and earlier under Microsoft Windows. SPSS files have a .sav file extension. SPSS files that have short variable names are exported. See “Example 1: Export a SAS Data Set to an SPSS SAV File” on page 49 for additional information.

SPSS Data Types

MISSING VALUES

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

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

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

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

When you are exporting to SPSS, SAS variable names that are longer than eight characters are truncated to eight characters. If the new name is truncated and results in an existing name, the last character changes to a single digit (1, 2, 3...) until the variable name becomes unique.

VALUE LABELS

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

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.

VARIABLE LABELS

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

DATA TYPES

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

Date, Jdate, Wkday, Qyr, Wkyc: Date, Jdate, Wkday, Qyr, Wkyc
Datetime, Dtime: Converts to a SAS datetime value and SAS datetime format.
Time: Converts to a SAS datatime value and SAS datatime format.
Adate: Converts to a SAS date value in the mmddyy format.
Moyr: Converts to a SAS date value in the mmyy format.

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

When exporting data, character fields have a maximum length of 256.
Numeric fields are 8-byte floating-point numbers, with these format conversions:
COMMA
   Converts to SPSS format type comma.

DOLLAR
   Converts to SPSS format type dollar.

DATE
   Converts to SPSS format type date.

MMDDYY
   Converts to SPSS format Adate.

MMMYY
   Converts to SPSS format Moyr.

DATETIME
   Converts to SPSS format Dtime.

TIME
   Converts to SPSS format Time.

Importing and Exporting Data in SPSS Files

SPSS Files (DBMS=SPSS)
   This IMPORT | EXPORT method uses SPSS file formats to access data in SPSS files on Linux.

Import Procedure and the Export Procedure Supported Syntax

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

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

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

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

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

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

PROC IMPORT OUT=WORK.CUSTOMER
   FILE="&tmpdir.customer.sav"
   DBMS=SPSS REPLACE;
RUN;
Example 3: Import Data from an SPSS File and Apply FMTLIB= 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;

Stata DTA Files

DTA Files Essentials

SAS/ACCESS supports Stata 12 and earlier versions under Microsoft Windows. Stata files have a .dta file extension.

See “Example 1: Export a SAS Data Set to a Stata File on a Local System ” on page 52 for additional information.

DTA Data Types

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

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 characters in length. The first character in a variable name can be any lowercase letter or uppercase letter or an underscore. Subsequent characters can be any of these characters, plus numerals. No other characters are permitted. Stata reserves the 19 words shown in the table below, which are not allowed to stand alone as variable names:

Table 5.5  Stata Reserved Words

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>_all</td>
<td>_n</td>
</tr>
<tr>
<td>in</td>
<td>using</td>
</tr>
<tr>
<td>_pred</td>
<td>double</td>
</tr>
</tbody>
</table>
If the program encounters any of these reserved words as variable names, it appends an underscore to the variable name to distinguish it from the reserved word. For example, \_N becomes \_N\_.

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

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

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

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

Note: Numeric formats only.

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

Stata date variables become numerics with a date format.

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

Importing and Exporting Stata Data Files
Stata DTA Files (DBMS=STATA)
This IMPORT | EXPORT method uses Stata DTA file formats to access data in Stata DTA files on Linux.
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.

**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: Export a SAS Data Set to a Stata File on a Local System**

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

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

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

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

```
PROC IMPORT OUT=WORK.CUSTOMER
   FILE="&tmpdir.customer.dta"
   DBMS=STATA REPLACE;
RUN;
```
Part 3

SAS LIBNAME Statement for the XLSX and JMP Engines on Linux

Chapter 6
SAS LIBNAME Statement for the XLSX Engine for Microsoft Excel on Linux .................................................. 55

Chapter 7
SAS LIBNAME Statement for the JMP Engine on Linux ................. 61
Chapter 6
SAS LIBNAME Statement for the XLSX Engine for Microsoft Excel on Linux

Dictionary

SAS LIBNAME Statement Syntax for the XLSX Engine

Associates a SAS libref with a Microsoft Excel workbook.

Valid in: Anywhere

Syntax

LIBNAME <libref> XLSX '<physical-path and filename.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 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 an XLSX file format. The LIBNAME statement associates a libref with an XLSX engine that supports connections to Microsoft Excel 2007, 2010, and later files.

This engine name XLSX is required.

Note: The XLSX engine enables you to read and write XLSX data residing on your local Linux machine.

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 reference (that is, libref) to a SAS library. The XLSX engine specifies an engine that supports the connection to the Microsoft file type `.XLSX`.

```sas
LIBNAME wsinv XLSX 'c:\WestSector\Q1_Invoices.xlsx';
```

`'physical-path- filename.xlsx'` is the physical-path and filename of a Microsoft Excel 2007 or later workbook that resides on the Linux machine. Using the extension (xlsx) is required. Enclose the path and workbook name in quotation marks.

See “Details” on page 56

**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 files created with any version of Microsoft Excel 2007 or later, it has no engine-specific LIBNAME options, such as `VERSION=`.

**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 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:

```sas
LIBNAME LIBREF LIST;
```

List the attributes of all the libraries:

```sas
LIBNAME _ALL_ LIST;
```

**Details**

**LIBNAME Statement Advantages**

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

The SAS/ACCESS LIBNAME statement enables you to reference spreadsheets directly in a DATA step or SAS procedure.

The XLSX LIBNAME engine enables you to read XLSX data from files in a Linux system.
For example, you can access an XLSX file stored in a Linux file system, and then once you have done so, you can write a DATA step or use PROC SQL to rename columns or apply labels, handle null values, apply formats, and so on, to the data:

```
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.

In SAS Studio, go to the Navigation pane and click Libraries. The first icon on the left under Libraries is the New Folder icon.

**Figure 6.1  New Library Window**

Type your libref, path, and any options that you need. Click OK.

**Using the LIBNAME Statement**

When using data from an XLSX file, you can use a SAS LIBNAME statement 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 a particular 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.

Assigning a Libref with a SAS/ACCESS LIBNAME Statement: This statement assigns the libref, `myxlsx`, to a Microsoft Excel XLSX file residing on a Linux machine 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 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 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;
```
**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. When you sort PC files data, the results might vary. Depending on whether the external spreadsheet or database places data has NULL values. If the sort encounters NULL values, are they listed at the beginning or end of the result set. NULL values are translated in SAS to missing values.

**Using SAS Functions with PC Files Data**

Librefs that refer to PC files with SAS functions might return a different value than the value returned when you use the functions with normal SAS data sets. The PATHNAME function might return a Microsoft Excel filename assigned for the libref. For a normal SAS libref, it returns the pathname for the assigned libref.

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

**System Options**

When using the output data table viewer to display table names that include special characters, you need to specify the SAS system option VALIDMEMNAME=EXTEND.
Chapter 7
SAS LIBNAME Statement for the JMP Engine on Linux

Dictionary

LIBNAME Statement, JMP Engine

Associates a libref with a JMP data table and enables you to read and write JMP data tables.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>Anywhere</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category:</td>
<td>Data Access</td>
</tr>
<tr>
<td>See:</td>
<td>&quot;LIBNAME Statement&quot; in SAS Viya Statements: Reference</td>
</tr>
</tbody>
</table>

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 with DBMS=JMP specified 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 using DBMS=JMP. 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.

• the LIBNAME statement for the JMP engine

Note: The JMP LIBNAME engine does not support extended attributes. If you want extended attributes, use the IMPORT procedure or 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 4

Appendixes

Appendix 1
DBF Procedure .................................................. 65
Appendix 1

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=filename | 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=filename | fileref
  <DATA=<libref.>SAS data set>
  <OUT=<libref.>SAS data set>;

Required Argument

DB2 | DB3 | DB4 | DB5=filename | fileref

specifies the version of the dBase file and the filename or fileref of a DBF file. It is recommended that you always use a FILENAME statement to define the physical filename with a logical name to avoid certain DOS naming limitations.

The DBn 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 filename, specify the filename without the .dbf extension. The file must be in the current directory. The filename must be in uppercase. The following PROC DBF statement creates the EMP.DBF file from the SAS data set MYLIB.EMPLOYEE:

PROC DBF DB5=EMP data=mylib.employee;
RUN;

You cannot specify the file extension .dbf or a full pathname:

PROC DBF DB5='/my_unix_directory/emp.dbf';

If you specify a fileref instead of a filename, the FILENAME statement must specify the filename 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.

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. 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 yyyyymmdd.

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.
Example: 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;
Recommended Reading

Here is the recommended reading list for this title:

- *Cody's Collection of Popular SAS Programming Tasks and How to Tackle Them*
- *Learning SAS by Example: A Programmer's Guide*
- *The Little SAS Book: A Primer, Fifth Edition*
- *SAS Viya Component Objects: Reference*
- *SAS Viya Data Set Options: Reference*
- *SAS Viya Formats and Informats: Reference*
- *SAS Viya Functions and CALL Routines: Reference*
- *SAS Viya Macro Language: Reference*
- *SAS Viya Statements: Reference*
- *SAS Viya Utility Procedures Guide*

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

SAS Books  
SAS Campus Drive  
Cary, NC 27513-2414  
Phone: 1-800-727-0025  
Fax: 1-919-677-4444  
Email: sasbook@sas.com  
Web address: [sas.com/store/books](http://sas.com/store/books)
# Index

<table>
<thead>
<tr>
<th>A</th>
<th>access methods 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>cells 28</td>
</tr>
<tr>
<td></td>
<td>character data 30</td>
</tr>
</tbody>
</table>
|        | comma-separated values files
|        | See CSV files    |
|        | conversion       |
|        | DBF fields to SAS variables 67|
|        | DBF fields 68    |
|        | CSV files 23     |
|        | CSV files       |
|        | exporting data sets to 27|
|        | exporting from SAS 28|
|        | importing into SAS 26, 27|
|        | importing subsets into SAS 27|
| D      | data access methods 3 |
|        | data sets        |
|        | exporting to CSV files 27|
|        | exporting to Excel files 33|
|        | data sources
|        | supported 7      |
|        | data sources, supported 7 |
|        | data type        |
|        | DTA 50           |
|        | data types       |
|        | Excel 30         |
|        | JMP 44           |
|        | SAV 47           |
|        | SPSS 47          |
|        | DATA= argument   |
|        | PROC EXPORT statement 16|
|        | database fields 39|
|        | DATAFILE= argument|
|        | PROC IMPORT statement 10|
|        | DATAROW= statement|
|        | IMPORT and EXPORT procedures 25|
|        | IMPORT procedure 34|
|        | DATASETS procedure|
|        | JMP files 44     |
|        | DATATABLE= argument|
|        | PROC IMPORT statement 11|
|        | date values 30   |
|        | datetime values 30|
|        | DB files, Paradox 47|
|        | dBase DBF files 39|
|        | examples of importing and exporting 42|
|        | setting environment variables 41|
|        | setting system options 41|
|        | supported IMPORT/EXPORT procedure statements 41|
|        | dBase DBFMEMO files 43|
|        | DBENCODING = option|
|        | statement for DBF files 42|
|        | DBF procedure 66  |
|        | DBF fields       |
|        | converting to SAS variables 67|
|        | DBF files        |
|        | See also dBase DBF files|
|        | converting to SAS data sets 65|
|        | transferring data sets with 67|
|        | DBF procedure 65  |
|        | DATA= option 66  |
|        | OUT= option 67   |
|        | syntax 66        |
|        | DBMEMO files 43  |
|        | DBMS specifications|
|        | EXPORT procedure 16|
|        | IMPORT procedure 10|
|        | DBMS= option     |
|        | PROC EXPORT statement 17|
|        | PROC IMPORT statement 11|
|        | DBSASLABEL= option|
|        | statement for Excel 31|
|        | delimited files  |
|        | IMPORT and EXPORT procedure statements for 24|
|        | sharing across hosts 23|
|        | text files 22     |
|        | delimiter 22      |
|        | delimiter-separated values (DSV) files 22|
DELIMITER= statement
IMPORT and EXPORT procedures 25
DSV files
DLM files 22
DTA data types 50
DTA files 50

E
ENDCOL= option
statement for Excel 35
ENDNAMEROW= option
statement for Excel 35
ENDROW= option
statement for Excel 35
environment variables
dBase DBF files 41
Excel 2007
XLS files 28
Excel files 28
Excel workbooks 28
EXPORT procedure 15
Access table specification compatibility
10, 16
DBMS specifications 16
Excel spreadsheet specification
compatibility 16
features of 7
recalculating formulas in Excel 34
statements for delimited files 24
supported data sources and platforms 7
syntax 16
export utilities 31
Export Wizard
features of 7
supported data sources and platforms 7
exporting
data sets to Excel files 33
DB files 47
DBF files 41, 42
DTA files 52
JMP files 46
SPSS files 49
Stata data files 51
extended attributes
EXPORT procedure 46
IMPORT procedure 46

F
file formats
CSV 23
DBF 39
DBFMEMO 43
DTA 50
JMP 44
Paradox DB 47
SAV 47
TAB 23
XLS 28
XLSB 28
XLSM 28
XLSX 28
FMTLIB= statement
statement for exporting JMP files 46
statement for exporting SPSS files 49
statement for importing JMP files 46
statement for importing SPSS files 49
formulas
recalculating in Excel XLSX files 34
functions 59

G
GETDELETED= option
statement for DBF files 42
GETNAMES= option
statement for Excel 35
GETNAMES= statement
IMPORT and EXPORT procedures 25
GUESSINGROWS= statement
IMPORT and EXPORT procedures 26

I
IMPORT procedure 9, 10
DBMS specifications 10
Excel spreadsheet specification
compatibility 10
Excel spreadsheet specifications 10
features of 7
INFILE statement 11, 22
statements for delimited files 24
supported data sources and platforms 7
import utilities 31
Import Wizard
supported data sources and platforms 7
importing
DB files 47
DBF files 41, 42
DTA files 52
JMP files 46
SPSS files 49
Stata data files 51
INFIL statement
delimited data 11, 22
importing delimited data 3

J
JMP data types 44
JMP engine LIBNAME statement 61
JMP files 44
   DATASETS procedure 44
examples of importing and exporting 46
   importing and exporting data in 46
missing values 44

L
   LABEL option
      PROC EXPORT statement 19
LIBNAME options
   XLSX files 56
LIBNAME statement, JMP Engine 61
LIBNAME statement, PC files
   assigning librefs 57, 58
   functions with PC files data 59
   sorting data 59
LIBNAME statement, PC Files
   XLSX engine 55
LIBNAME statement, XLSX engine 55
   syntax 55
   libraries
   writing attributes to log 58
librefs
   assigning 57, 58
   disassociating 58
log
   writing library attributes to 58

M
   META data type
      JMP files 45
META statement
   for exporting JMP files 46
   for importing JMP files 46
Microsoft Access
   table specification compatibility 10, 16
Microsoft Excel
   data types 30
   DBMS spreadsheet specifications 10
   import and export components 31
   spreadsheet specification compatibility 10, 16
   workbooks 28
Microsoft Excel 2007 28
Microsoft Excel files 28
   exporting data sets to 33
Microsoft Excel workbook files 28
   missing values
      JMP files 44

N
   NAMEROW= option
   OUTFILE= argument
      PROC EXPORT statement 16

P
   Paradox DB files 47
   examples of importing and exporting 47
PC files
   assigning librefs 57, 58
   data access methods 3
   functions with PC files data 59
   sorting data 59
platforms, supported 7
PROC DBF statement
   DBF procedure 66, 0
PROC EXPORT statement 16
   EXPORT procedure 16, 0
PROC IMPORT statement 9
   IMPORT procedure 10
   PUTNAMES= option
      statement for Excel 36
      IMPORT and EXPORT procedures 26

R
   RANGE= option
      statement for Excel 31, 36
   ranges 28
RECALL command
   IMPORT procedure, Delimited files 24
   REPLACE option
      PROC EXPORT statement 19

S
   sample data 4
SAS data sets
   converting to DBF files 67
   transferring with DBF files 67
SAV data types 47
SAV files 47
SHEET= option
   statement for Excel 32, 37
   sorting PC files data 59
   space-separated data values
importing into SAS 27
SPSS SAV data types 47
SPSS SAV files 47
SPSS SAVE files
  Versions supported 47
STARTCOL= option
  statement for Excel 38
STARTROW= option
  statement for Excel 38
Stata DTA data types 50
  MISSING VALUES 50
Stata DTA files 50
  examples of importing and exporting 52
  Versions supported 50
sub-setting
  importing CSV file subsets 27
supported data sources 7
supported platforms 7
system options
  dBase DBF files 41

T
TAB files 23
  tab-delimited files 23
  TEXTSIZE= option

statement for Excel 33
time values 30

U
USEDATE= option
  statement for Excel 33
utilities
  import and export 31

V
VALIDMEMNAME= system option 27, 28

W
worksheets 28

X
XLSB files 28
XLSM files 28
XLSX files 28
XLSX LIBNAME engine 56
  syntax 55