SAS Libraries

Definition

A *SAS library* contains one or more files that are defined, recognized, and accessible by SAS, and that are referenced and stored as a unit. One special type of file is called a *catalog*. In SAS libraries, catalogs function much like subfolders for grouping other members.

Predefined SAS Libraries

By default, SAS defines several libraries for you:

Sashelp
: a permanent library that contains sample data and other files that control how SAS works at your site. This is a Read-Only library.

Sasuser
: a permanent library that contains SAS files in the Profile catalog and that stores your personal settings. This is also a convenient place to store your own files.

Work
: a temporary library for files that do not need to be saved from session to session.

You can also define additional libraries. When you define a library, you indicate the location of your SAS files to SAS. After you define a library, you can manage SAS files within it.

Note: If you are using SAS Studio, you might encounter the Webwork library. Webwork is the default output library in interactive mode. For more information about the Webwork library, see *SAS Studio: User's Guide*.

Defining Libraries

To define a library, you assign a library name to it and specify the location of the files, such as a directory path.

You can also specify an engine, which is a set of internal instructions that SAS uses for writing to and reading from files in a library.

You can define SAS libraries using programming statements. For information about how to write LIBNAME statements to define SAS libraries, see Assigning Librefs on page 23.

TIP Depending on your operating environment and the SAS/ACCESS products that you license, you can create libraries with various engines. Each engine enables you to read a different file format, including file formats from other software vendors.

When you delete a SAS library, the pointer to the library is deleted, and SAS no longer has access to the library. However, the contents of the library still exist in your operating environment.

How SAS Files Are Stored

A SAS library is the highest level of organization for information within SAS.

For example, in the Windows and UNIX environments, a library is typically a group of SAS files in the same folder or directory.

The table below summarizes the implementation of SAS libraries in various operating environments.

Table 2.1 *Environments and SAS Libraries*

Environment	Library Definition
Windows, UNIX	a group of SAS files that are stored in the same directory. Other files can be stored in the directory, but only the files that have SAS file extensions are recognized as part of the SAS library.
z/OS	a specially formatted host data set in which only SAS files are stored.

Storing Files Temporarily or Permanently

Depending on the library name that you use when you create a file, you can store SAS files temporarily or permanently.

Table 2.2 *Temporary and Permanent SAS Libraries*

Temporary SAS libraries last only for the current SAS session.	If you do not specify a library name when you create a file, the file is stored in the temporary SAS library, Work. If you specify the library name Work, then the file is stored in the temporary SAS library. When you end the session, the temporary library and all of its files are deleted.
Permanent SAS libraries are available to you during subsequent SAS sessions.	To store files permanently in a SAS library, specify a library name other than the default library name Work. In the example, when you specify the library name Cert when you create a file, you are specifying that the file is to be stored in a permanent SAS library.

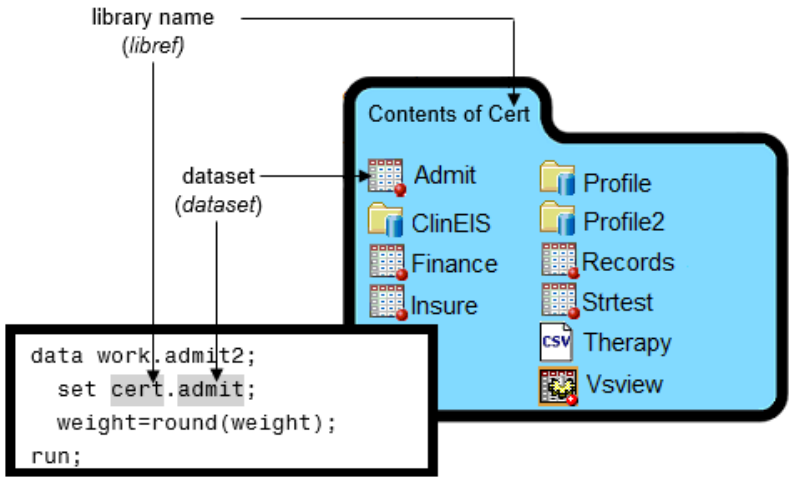
Referencing SAS Files

Referencing Permanent SAS Data Sets

To reference a permanent SAS data set in your SAS programs, use a two-level name consisting of the library name and the data set name:

libref.dataset
: In the two-level name, *libref* is the name of the SAS library that contains the data set, and *data set* is the name of the SAS data set. A period separates the libref and data set name.

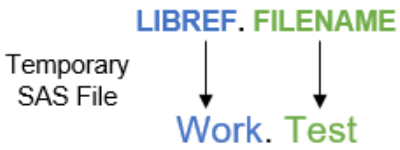
Figure 2.3 *Two-Level Permanent SAS Name*



Referencing Temporary SAS Files

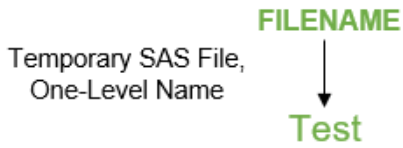
To reference temporary SAS files, you can specify the default libref Work, a period, and the data set name. For example, the two-level name, Work.Test, references the SAS data set named Test that is stored in the temporary SAS library Work.

Figure 2.4 *Two-Level Temporary SAS Library Name*



Alternatively, you can use a one-level name (the data set name only) to reference a file in a temporary SAS library. When you specify a one-level name, the default libref Work is assumed. For example, the one-level name Test references the SAS data set named Test that is stored in the temporary SAS library Work.

Figure 2.5 *One-Level Temporary SAS Library Name*



Rules for SAS Names

By default, the following rules apply to the names of SAS data sets, variables, and libraries:

- They must begin with a letter (A-Z, either uppercase or lowercase) or an underscore (_).
- They can continue with any combination of numbers, letters, or underscores.
- They can be 1 to 32 characters long.
- SAS library names (librefs) can be 1 to 8 characters long.

These are examples of valid data set names and variable names:

- Payroll
- LABDATA2015_2018
- _EstimatedTaxPayments3

VALIDVARNAME=System Option

SAS has various rules for variable names. You set these rules using the VALIDVARNAME= system option. VALIDVARNAME specifies the rules for valid SAS variable names that can be created and processed during a SAS session.

Syntax, VALIDVARNAME=

VALIDVARNAME= *V7|UPCASE|ANY*

V7 specifies that variable names must follow these rules:

- SAS variable names can be up to 32 characters long.
- The first character must begin with a letter of the Latin alphabet (A - Z, either uppercase or lowercase) or an underscore (_). Subsequent characters can be letters of the Latin alphabet, numerals, or underscores.
- Trailing blanks are ignored. The variable name alignment is left-justified.
- A variable name cannot contain blanks or special characters except for an underscore.
- A variable name can contain mixed-case letters. SAS stores and writes the variable name in the same case that is used in the first reference to the variable. However, when SAS processes a variable name, SAS internally converts it to uppercase. Therefore, you cannot use the same variable name with a different combination of uppercase and lowercase letters to represent different variables. For example, **`cat`**, **`Cat`**, and **`CAT`** all represent the same variable.
- Do not assign variables the names of special SAS automatic variables (such as _N_ and _ERROR_) or variable list names (such as _NUMERIC_, _CHARACTER_, and _ALL_) to variables.

UPCASE specifies that the variable name follows the same rules as V7, except that the variable name is uppercase, as in earlier versions of SAS.

ANY specifies that SAS variable names must follow these rules:

- The name can begin with or contain any characters, including blanks, national characters, special characters, and multi-byte characters.
- The name can be up to 32 bytes long.
- The name cannot contain any null bytes.
- Leading blanks are preserved, but trailing blanks are ignored.
- The name must contain at least one character. A name with all blanks is not permitted.
- A variable name can contain mixed-case letters. SAS stores and writes the variable name in the same case that is used in the first reference to the variable. However, when SAS processes a variable name, SAS internally converts it to uppercase. Therefore, you cannot use the same variable name with a different combination of uppercase and lowercase letters to represent different variables. For example, **`cat`**, **`Cat`**, and **`CAT`** all represent the same variable.

Note: If you use characters other than the ones that are valid when VALIDVARNAME=V7, then you must express the variable name as a name literal and set VALIDVARNAME=ANY. If the name includes either a percent sign (%) or an ampersand (&), then use single quotation marks in the name literal to avoid interaction with the SAS macro facility.

CAUTION:

Throughout SAS, using the name literal syntax with SAS member names that exceed the 32-byte limit or have excessive embedded quotation marks might cause unexpected results. The VALIDVARNAME=ANY system option enables compatibility with other DBMS variable (column) naming conventions, such as allowing embedded blanks and national characters.

VALIDMEMNAME=System Option

You can use the VALIDMEMNAME= system option to specify rules for naming SAS data sets.

Syntax, VALIDMEMNAME=

VALIDMEMNAME= COMPATIBLE | EXTEND

Important: COMPATIBLE is the default system option for VALIDMEMNAME=.

COMPATIBLE specifies that a SAS data set name must follow these rules:

- The length of the names can be up to 32 characters long.
- Names must begin with a letter of the Latin alphabet (A- Z, a - z) or an underscore. Subsequent characters can be letters of the Latin alphabet, numerals, or underscores.
- Names cannot contain blanks or special characters except for an underscore
- Names can contain mixed-case letters. SAS internally converts the member name to uppercase. Therefore, you cannot use the same member name with a different combination of uppercase and lowercase letters to represent different variables. For example, **`customer`**, **`Customer`**, and **`CUSTOMER`** all represent the same member name. How the name is saved on disk is determined by the operating environment.

EXTEND specifies that the data set name must follow these rules:

- Names can include national characters.
- The name can include special characters, except for the / \ * ? " < > |: - characters.
- The name must contain at least one character.
- The length of the name can be up to 32 bytes.
- Null bytes are not allowed.
- Names cannot begin with a blank or a '.' (period).
- Leading and trailing blanks are deleted when the member is created.
- Names can contain mixed-case letters. SAS internally converts the member name to uppercase. Therefore, you cannot use the same member name with a different combination of uppercase and lowercase letters to represent different variables. For example, **`customer`**, **`Customer`**, and **`CUSTOMER`** all represent the same member name. How the name appears is determined by the operating environment.

Note: If VALIDMEMNAME=EXTEND, SAS data set names must be written as a SAS name literal. If you use either a percent sign (%) or an ampersand (&), then you must use single quotation marks in the name literal in order to avoid interaction with the SAS macro facility.

CAUTION:

Throughout SAS, using the name literal syntax with SAS member names that exceed the 32-byte limit or that have excessive embedded quotation marks might cause unexpected results. The intent of the VALIDMEMNAME=EXTEND system option is to enable compatibility with other DBMS member naming conventions, such as allowing embedded blanks and national characters.

When to Use VALIDMEMNAME=System Option

Use VALIDMEMNAME= EXTEND system option when the characters in a SAS data set name contains one of the following:

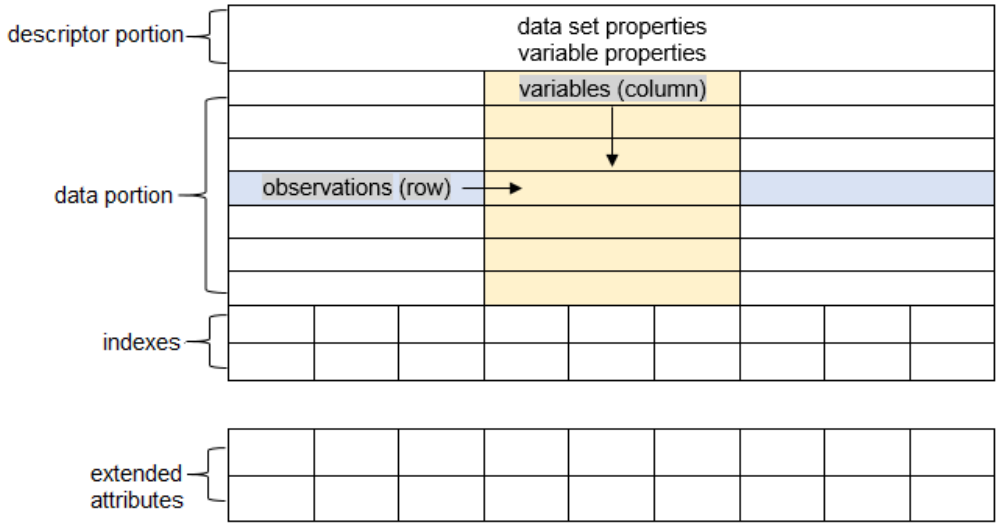
- international characters
- characters supported by third-party databases
- characters that are commonly used in a filename

SAS Data Sets

Overview of Data Sets

A *SAS data set* is a file that consists of two parts: a descriptor portion and a data portion. Sometimes a SAS data set also points to one or more indexes, which enable SAS to locate rows in the data set more efficiently. (The data sets that are shown in this Lesson do not contain indexes.) Extended attributes are user-defined attributes that further define a SAS data set.

Figure 2.6 *Parts of a SAS Data Set*



Descriptor Portion

The descriptor portion of a SAS data set contains information about the data set, including the following:

- the name of the data set
- the date and time that the data set was created
- the number of observations
- the number of variables

The table below lists part of the descriptor portion of the data set Cert.Insure, which contains insurance information for patients who are admitted to a wellness clinic.

Table 2.3 *Descriptor Portion of Attributes in a SAS Data Set*

Data Set Name:	CERT.INSURE
Member Type:	DATA
Engine:	V9
Created:	07/03/2018 10:53:05
Observations:	21
Variables:	7
Indexes:	0
Observation Length:	64

SAS Variable Attributes

The descriptor portion of a SAS data set contains information about the properties of each variable in the data set. The properties information includes the variable's name, type, length, format, informat, and label.

When you write SAS programs, it is important to understand the attributes of the variables that you use. For example, you might need to combine SAS data sets that contain same-named variables. In this case, the variables must be the same type (character or numeric). If the same-named variables are both character variables, you still need to check that the variable lengths are the same. Otherwise, some values might be truncated.

The following table uses Cert.Insure data and the VALIDVARNAME=ANY system option. The SAS variable has several attributes that are listed here:

Table 2.4 *Variable Attributes*

Variable Attribute	Definition	Example	Possible Values
Name	identifies a variable. A variable name must conform to SAS naming rules. See "Rules for SAS Names" for SAS names rules.	Policy Total Name	Any valid SAS name.
Type	identifies a variable as numeric or character. Character variables can contain any values. Numeric variables can contain only numeric values (the numerals 0 through 9, +, -, ., and E for scientific notation).	Char Num Char	Numeric and character

Variable Attribute	Definition	Example	Possible Values
Length	refers to the number of bytes used to store each of the variable's values in a SAS data set. Character variables can be up to 32,767 bytes long. All numeric variables have a default length of 8 bytes. Numeric values are stored as floating-point numbers in 8 bytes of storage.	5 8 14	2 to 8 bytes 1 to 32,767 bytes for character
Format	affects how data values are written. Formats do not change the stored value in any way; they merely control how that value is displayed. SAS offers a variety of character, numeric, and date and time formats.	$98.64	Any SAS format If no format is specified, the default format is **`BEST12.`** for a numeric variable, and **`$w.`** for a character variable.
Informat	reads data values in certain forms into standard SAS values. Informats determine how data values are read into a SAS data set. You must use informats to read numeric values that contain letters or other special characters.	99	Any SAS informat The default informat for numeric is **`w.d`** and for character is **`$w.`**
Label	refers to a descriptive label up to 256 characters long. A variable label, which can be printed by some SAS procedures, is useful in report writing.	Policy Number Total Balance Patient Name	Up to 256 characters

The following output is the descriptor portion of Cert.Insure.

Output 2.2 *Descriptor Portion of Cert.Insure*

Alphabetic List of Variables and Attributes						
#	Variable	Type	Len	Format	Informat	Label
7	BalanceDue	Num	8	6.2		
4	Company	Char	11			
1	ID	Char	4			
2	Name	Char	14			Patient Name
5	PctInsured	Num	8			
3	Policy	Char	5			Policy Number
6	Total	Num	8	DOLLAR8.2	COMMA10.	Total Balance

Data Portion

Data Portion Overview

The data portion of a SAS data set is a collection of data values that are arranged in a rectangular table. In the example below, the company **`MUTUALITY`** is a data value, Policy **`32668`** is a data value, and so on.

Figure 2.7 *Parts of a SAS Data Set: Data Portion*

ID	Name	Policy	Company	PctInsured	Total	BalanceDue
2458	Murray, W	32668	MUTUALITY	100	98.64	0.00
2462	Almers, C	95824	RELIABLE	80	780.23	156.05
2501	Bonaventure, T	87795	A&R	80	47.38	9.48
2523	Johnson, R	39022	ACME	50	122.07	61.04

Observations (Rows)

Observations (also called rows) in the data set are collections of data values that usually relate to a single object. The values **`2458`**, **`Murray W`**, **`32668`**, **`MUTALITY`**, **`100`**, **`98.64`**, and **`0.00`** are comprised in a single observation in the data set shown below.

Figure 2.8 *Parts of a SAS Data Set: Observations*

	ID	Name	Policy	Company	PctInsured	Total	BalanceDue
Observation	2458	Murray, W	32668	MUTUALITY	100	98.64	0.00
	2462	Almers, C	95824	RELIABLE	80	780.23	156.05
	2501	Bonaventure, T	87795	A&R	80	47.38	9.48
	2523	Johnson, R	39022	ACME	50	122.07	61.04

This data set has 21 observations, each containing information about an individual. To view the full descriptor portion of this data set, see Table 2.3 on page 16. A SAS data set can store any number of observations.

Variables (Columns)

Variables (also called columns) in the data set are collections of values that describe a particular characteristic. The values `2458`, `2462`, `2501`, and `2523` are comprised in the variable ID in the data set shown below.

Figure 2.9 *Parts of a SAS Data Set: Variables*

Variables

ID	Name	Policy	Company	PctInsured	Total	BalanceDue
2458	Murray, W	32668	MUTUALITY	100	98.64	0.00
2462	Almers, C	95824	RELIABLE	80	780.23	156.05
2501	Bonaventure, T	87795	A&R	80	47.38	9.48
2523	Johnson, R	39022	ACME	50	122.07	61.04

This data set contains seven variables: ID, Name, Policy, Company, PctInsured, Total, and BalanceDue. A SAS data set can store thousands of variables.

Missing Values

Every variable and observation in a SAS data set must have a value. If a data value is unknown for a particular observation, a missing value is recorded in the SAS data set. A period (.) is the default value for a missing numeric value, and a blank space is the default value for a missing character value.

Figure 2.10 *Parts of a SAS Data Set: Missing Data Values*

ID	Name	Policy	Company	PctInsured	Total	BalanceDue
2458	Murray, W	32668	MUTUALITY	100	98.64	0.00
2462	Almers, C	95824	RELIABLE	80	780.23	156.05
2501	Bonaventure, T	87795	A&R	.	47.38	9.48
2523	Johnson, R	39022	ACME	50	122.07	61.04

Missing Value

SAS Indexes

An index is a separate file that you can create for a SAS data file in order to provide direct access to a specific observation. The index file has the same name as its data file and a member type of INDEX. Indexes can provide faster access to specific observations, particularly when you have a large data set. The purpose of SAS indexes is to optimize WHERE expressions and to facilitate BY-group processing. For more information, see "Specifying WHERE Expressions" and see "Group Processing Using the BY Statement".

Extended Attributes

Extended attributes are user-defined metadata that is defined for a data set or for a variable (column). Extended attributes are represented as name-value pairs.

TIP You can use PROC CONTENTS to display data set and variable extended attributes.

SAS Libraries

A *SAS library* is a collection of one or more SAS files, including SAS data sets, that are referenced and stored as a unit. In a directory-based operating environment, a *SAS library* is a group of SAS files that are stored in the same directory. In z/OS, a *SAS library* is a group of SAS files that are stored in an operating environment file.

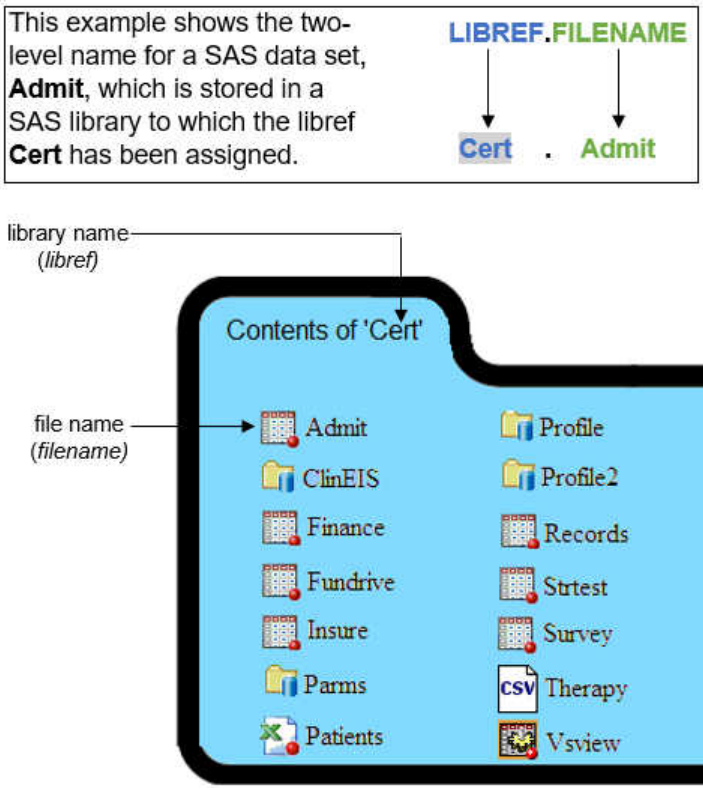
Assigning Librefs

Often the first step in setting up your SAS session is to define the libraries. You can use programming statements to assign library names.

To reference a permanent SAS file:

1. Assign a name (*libref*) to the SAS library in which the file is stored.
2. Use the libref as the first part of the two-level name (*libref.filename*) to reference the file within the library.

Figure 3.1 *Defining Libraries*



A logical name (libref) can be assigned to a SAS library using the LIBNAME statement. You can include the LIBNAME statement with any SAS program so that the SAS library is assigned each time the program is submitted. Using the user interface, you can set up LIBNAME statements to be automatically assigned when SAS starts.

Syntax, LIBNAME statement:

LIBNAME *libref engine 'SAS-data-library'*;

- *libref* is 1 to 8 characters long, begins with a letter or underscore, and contains only letters, numbers, or underscores.
- *engine* is the name of a library engine that is supported in your operating environment.

  Note: For SAS®9, the default engine is V9, which works in all operating environments.
- *SAS-data-library* is the name of a SAS library in which SAS data files are stored. The specification of the physical name of the library differs by operating environment.

The LIBNAME statement below assigns the libref Cert to the SAS library **`C:\Users\Student1\Cert`** in the Windows environment. When the default engine is used, you do not have to specify it in the LIBNAME statement.

```
libname cert 'C:\Users\Student1\Cert';
```

The table below gives examples of physical names for SAS libraries in various operating environments.

Table 3.1 *Sample Physical Names for SAS Libraries*

Environment	Sample Physical Name
Windows	c:\fitness\data
UNIX	/users/april/fitness/sasdata
z/OS)	april.fitness.sasdata

TIP You can use multiple LIBNAME statements to assign as many librefs as needed.

Verifying Librefs

After assigning a libref, it is a good idea to check the log to verify that the libref has been assigned successfully.

Log 3.1 *Output for Cert Libref*

```
1    libname cert 'C:\Users\Student1\Cert';
NOTE: Libref CERT was successfully assigned as follows:
      Engine:        V9
      Physical Name: C:\Users\Student1\Cert
```

How Long Librefs Remain in Effect

The LIBNAME statement is global, which means that the librefs remain in effect until changed or canceled, or until the SAS session ends.

By default, the LIBNAME statement assigns the libref for the current SAS session only. Each time you begin a SAS session, you must assign a libref to each permanent SAS library that contains files that you want to access in that session. (Remember that Work is the default libref for a temporary SAS library.)

Specifying Two-Level Names

After you assign a libref, you specify it as the first element in the two-level name for a SAS file.

In order for the PRINT procedure to read cert.admit, you specify the two-level name of the file as follows:

```
proc print data=cert.admit;
run;
```

Referencing Third-Party Data

You can use the LIBNAME statement to reference not only SAS files but also files that were created with other software products, such as database management systems.

A SAS engine is a set of internal instructions that SAS uses for writing to and reading from files in a SAS library or a third-party database. SAS can read or write these files by using the appropriate engine for that file type. For some file types, you need to tell SAS which engine to use. For others, SAS automatically chooses the appropriate engine.

An example of an engine that accesses third-party data is the XLSX engine, which processes Microsoft Excel workbooks.

Accessing Stored Data

If your site licenses SAS/ACCESS software, you can use the LIBNAME statement to access data that is stored in a database management system (DBMS) file. The types of data you can access depend on your operating environment and on which SAS/ACCESS products you have licensed. For more information about SAS/ACCESS engines, see the SAS documentation for your DBMS.

Viewing SAS Libraries

Viewing Libraries

Besides accessing library details with librefs, you can also see libraries in other environments. You can access a brief overview on the windows and menus for your environment at http://video.sas.com/. From **Categories** select **How To Tutorials** ⇨ **Programming**. Select the video for your SAS environment. Other tutorials are available from the SAS website.

Viewing Libraries Using PROC CONTENTS

You can use the CONTENTS procedure to create SAS output that describes either of the following:

- the contents of a library
- the descriptor information for an individual SAS data set

The default library is either Work or User depending on your SAS solution or environment.

Syntax, PROC CONTENTS step:

PROC CONTENTS DATA=*SAS-file-specification* **NODS;**

RUN;

- *SAS-file-specification* specifies an entire library or a specific SAS data set within a library. *SAS-file-specification* can take one of the following forms:
  - *<libref.>SAS-data-set* names one SAS data set to process.
  - *<libref.>*_ALL_ requests a listing of all files in the library. (Use a period (.) to append _ALL_ to the libref.)
- NODS suppresses the printing of detailed information about each file when you specify _ALL_. (You can specify NODS *only* when you specify _ALL_.)

Example: View the Contents of an Entire Library

To view the contents of an entire library, specify the _ALL_ and NODS options in the PROC CONTENTS step. The _ALL_ option lists all files in the Cert library, and the NODS option suppresses the printing of detailed information about each specific file.

```
proc contents data=cert._all_ nods;
run;
```

The following output displays a partial output of the contents of the Cert library. The _ALL_ option lists all files including indexes, views, and catalogs.

Output 3.1 *PROC CONTENTS Output: the SAS Library Cert (partial output)*

The CONTENTS Procedure

Directory	
Libref	CERT
Engine	V9
Physical Name	C:\Users\Student1\Cert
Filename	C:\Users\Student1\Cert
Owner Name	Users
File Size	96KB
File Size (bytes)	98304

#	Name	Member Type	File Size	Last Modified
1	ACITIES	DATA	128KB	07/03/2018 10:53:06
2	ADMIT	DATA	128KB	07/03/2018 10:53:05
3	ADMIT2	DATA	128KB	08/07/2017 13:46:50
4	ADMITFEE	DATA	128KB	08/15/2017 14:25:18
5	ADMITJUNE	DATA	128KB	07/03/2018 10:53:05
6	AIRPORTS	DATA	128KB	07/03/2018 10:53:06
7	ALL	VIEW	9KB	07/03/2018 10:53:08
8	ALLEMPS	DATA	128KB	07/03/2018 10:53:06
9	AMOUNTS	DATA	128KB	11/27/2017 11:04:23
10	APRBILLS	DATA	128KB	11/02/2017 15:11:30

Example: View Descriptor Information

To view the descriptor information for only a specific data set, use the PROC CONTENTS step. The following example lists the descriptor information for Cert.Amounts including an alphabetic list of the variables in the data set.

```
proc contents data=cert.amounts;
run;
```

The following output is the result from submitting the PROC CONTENTS step.

Output 3.2 *PROC CONTENTS Output*

Data Set Name	CERT.AMOUNTS	**Observations**	6
Member Type	DATA	**Variables**	4
Engine	V9	**Indexes**	0
Created	11/27/2017 11:04:23	**Observation Length**	40
Last Modified	11/27/2017 11:04:23	**Deleted Observations**	0
Protection		**Compressed**	NO
Data Set Type		**Sorted**	NO
Label			
Data Representation	WINDOWS_64		
Encoding	wlatin1 Western (Windows)		

Engine/Host Dependent Information	
Data Set Page Size	65536
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	1632
Obs in First Data Page	6
Number of Data Set Repairs	0
ExtendObsCounter	YES
Filename	C:\Users\Student1\Cert\amounts.sas7bdat
Release Created	9.0401M4
Host Created	X64_10PRO
Owner Name	Users
File Size	128KB
File Size (bytes)	131072

Alphabetic List of Variables and Attributes			
#	**Variable**	**Type**	**Len**
4	Amount	Num	8
3	Date	Num	8
2	EmpID	Num	8
1	Name	Char	13

Example: View Descriptor Information Using the Varnum Option

By default, PROC CONTENTS lists variables alphabetically. To list variable names in the order of their logical position (or creation order) in the data set, specify the VARNUM option in PROC CONTENTS.

```
proc contents data=cert.amounts varnum;
run;
```

Output 3.3 *View Descriptor Information for Cert.Amounts Using the VARNUM Option*

Variables in Creation Order			
#	Variable	Type	Len
1	Name	Char	13
2	EmpID	Num	8
3	Date	Num	8
4	Amount	Num	8

The IMPORT Procedure

The Basics of PROC IMPORT

The IMPORT procedure reads data from an external data source and writes it to a SAS data set. You can import structured and unstructured data using PROC IMPORT. You can import delimited files (blank, comma, or tab) along with Microsoft Excel files. If you are using SAS 9.4, then you can import JMP 7 or later files as well.

When you run the IMPORT procedure, it reads the input file and writes the data to the specified SAS data set. By default, the IMPORT procedure expects the variable names to appear in the first row. The procedure scans the first 20 rows to count the variables, and it attempts to determine the correct informat and format for each variable. You can use the IMPORT procedure statements to do the following:

- indicate how many rows SAS scans for variables to determine the type and length (GUESSINGROWS=)
- modify whether SAS extracts the variable names from the first row of the data set (GETNAMES=)
- indicate at which row SAS begins to read the data (DATAROW=)

When the IMPORT procedure reads a delimited file, it generates a DATA step to import the data. You control the results with options and statements that are specific to the input data source.

The IMPORT procedure generates the specified output SAS data set and writes information about the import to the SAS log. The log displays the DATA step code that is generated by the IMPORT procedure.

PROC IMPORT Syntax

The IMPORT procedure imports an external data file to a SAS data set.

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

DATAFILE= "*filename*" | "*fileref*"
specifies the complete path and filename or fileref for the input PC file, spreadsheet, or delimited external 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.

If you specify a fileref, complete path, and filename does not include special characters, then you can omit the quotation marks.

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 only if SAS supports the data type. 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 be able to import it correctly. In many cases, the procedure attempts to convert the data to the best of its ability. However, conversion is not possible for some types.

Interactions By default, the IMPORT procedure reads delimited files as varying record-length files. If your external file has a fixed-length format, use the FILENAME statement prior to PROC IMPORT to specify the input filename using the RECFM=F and LRECL= options.

When you use a fileref to specify a delimited file to import, the logical record length (LRECL) defaults to 256, unless you specify the LRECL= option in the FILENAME statement. The maximum LRECL value that the IMPORT procedure supports is 32,767.

For delimited files, the first 20 rows are scanned to determine the variable attributes. You can increase the number of rows that are scanned by using the GUESSINGROWS= statement. All values are read in as character strings. If a Date and Time format or a numeric informat can be applied to the data value, the type is declared as numeric. Otherwise, the type remains character.

OUT= *<libref.> SAS-data-set*
identifies the output SAS data set with either a one or two-level SAS name (library and member name). If the specified SAS data set does not exist, the IMPORT procedure creates it. If you specify a one-level name, by default the IMPORT procedure uses either the USER library (if assigned) or the WORK library (if USER is not assigned).

A SAS data set name can contain a single quotation mark when the VALIDMEMNAME=EXTEND system option is also specified. Using VALIDMEMNAME= expands the rules for the names of certain SAS members, such as a SAS data set name.

If a SAS data set name contains national characters or special characters, use VALIDMEMNAME=EXTEND system option. The exceptions for special characters are: / \ * ? " < > | : —. Using VALIDMEMNAME= expands the rules for the name of certain SAS members, such as a SAS data set name. For more information, see "VALIDMEMNAME=System Option" on page 14.

TABLE= "*tablename*"
specifies the 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. Note that the DBMS table name might be case sensitive.

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

<DBMS=*identifier*>
specifies the type of data to import.

Here are the common DBMS identifiers that are included with Base SAS:

- CSV — comma-separated values. For a comma-separated file with a .CSV extension, DBMS= is optional.
- JMP — JMP files. Use JMP 7 or later. Use DBMS=JMP to specify importing JMP files. JMP variable names can be up to 255 characters long. SAS supports importing JMP files that have more than 32,767 variables.
- TAB — tab-delimited values. Specify DBMS=DLM to import any other delimited file that does not end in .CSV.

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

Instead, use a SAS DATA step with the REPLACE= data set option to replace a permanent SAS data set.

<SAS-data-set-options>
specifies 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.

Restriction You cannot specify data set options when importing delimited, comma-separated, or tab-delimited external files.

Example: Importing an Excel File with an XLSX Extension

This example imports an Excel file and creates a temporary SAS data set, Work.BootSales.

```
options validvarname=v7;                                    /* #1 */
proc import datafile='C:\Users\Student1\cert\boots.xlsx'    /* #2 */
  dbms=xlsx
  out=work.bootsales
  replace;
  sheet=boot;                                               /* #3 */
  getnames=yes;                                             /* #4 */
run;
proc contents data=bootsales;                               /* #5 */
run;
proc print data=bootsales;
run;
```

1 The VALIDVARNAME=V7 statement forces SAS to convert spaces to underscores when it converts column names to variable names. In SAS Studio, the _ (underscore) in Total_Sale would not be added without the VALIDVARNAME=V7 statement.

2 Specify the input file. DATAFILE= specifies the path for the input file. The DBMS= option specifies the type of data to import. When importing an Excel workbook, specify DBMS=XLSX. The REPLACE option overwrites an existing SAS data set. The OUT= option identifies the output SAS data set.

3 Use the SHEET option to import specific worksheets from an Excel workbook.

4 Set the GETNAMES= statement to YES to generate variable names from the first row of data.

5 Use the CONTENTS procedure to display the descriptor portion of the Work.BootSales data set.

The following is printed to the SAS log. The SAS log notes that the import was successful. It also notes that there is a variable name change from Total Sale (with a space between the two words) to Total_Sale. SAS converted the space to an underscore (_).

Log 4.1 *SAS Log*

```
75   options validvarname=v7;
76   proc import datafile='C:\Users\Student1\cert\boots.xlsx'
77      dbms=xlsx
78      out=work.bootsales replace;
79      sheet=boot;
80      getnames=yes;
81   run;

NOTE:    Variable Name Change.  Total Sale -> Total_Sale
NOTE: The import data set has 10 observations and 3 variables.
NOTE: WORK.BOOTSALES data set was successfully created.
```

Output 4.1 *PROC CONTENTS Descriptor Portion (partial output)*

Alphabetic List of Variables and Attributes						
#	Variable	Type	Len	Format	Informat	Label
2	City	Char	11	$11.	$11.	City
1	Region	Char	25	$25.	$25.	Region
3	Total_Sale	Num	8	DOLLAR15.2		Total Sale

Output 4.2 *PROC PRINT Output of the Work.BootSales Data Set*

Obs	Region	City	Total_Sale
1	Africa	Addis Ababa	$191,821.00
2	Asia	Bangkok	$9,576.00
3	Canada	Calgary	$63,280.00
4	Central America/Carribean	Kingston	$393,376.00
5	Eastern Europe	Budapest	$317,515.00
6	Middle East	Al-Khobar	$44,658.00
7	Pacific	Auckland	$97,919.00
8	South America	Bogota	$35,805.00
9	United States	Chicago	$305,061.00
10	Western Europe	Copenhagen	$4,657.00

For an alternate method of reading Microsoft Excel files in SAS, see "Reading Microsoft Excel Data with the XLSX Engine" on page 45.

Example: Importing a Delimited File with a TXT Extension

This example imports a delimited external file and creates a temporary SAS data set, Work.MyData. The delimiter is an ampersand (&).

```
options validvarname=v7;
proc import datafile='C:\Users\Student1\cert\delimiter.txt'    /*#1*/
  dbms=dlm                                                     /*#2*/
  out=mydata
  replace;
  delimiter='&';                                               /*#3*/
```

```
  getnames=yes;
run;
proc print data=mydata;
run;
```

1 Specify the input file. DATAFILE= specifies the path for the input file. The DBMS= option specifies the type of data to import. If the delimiter is a character other than TAB or CSV, then the DBMS= option is DLM. The REPLACE option overwrites an existing SAS data set. The OUT= option identifies the output SAS data set.

2 Specify an ampersand (&) for the DELIMITER statement.

3 Set the GETNAMES= statement to YES to generate variable names from the first row of data.

Output 4.3 *PROC PRINT Output: Work.MyData Data Set*

Obs	Region	State	Month	Expenses	Revenue
1	Southern	GA	JAN2001	2000	8000
2	Southern	GA	FEB2001	1200	6000
3	Southern	FL	FEB2001	8500	11000
4	Northern	NY	FEB2001	3000	4000
5	Northern	NY	MAR2001	6000	5000
6	Southern	FL	MAR2001	9800	13500
7	Northern	MA	MAR2001	1500	1000

Example: Importing a Space-Delimited File with a TXT Extension

This example imports a space-delimited file and creates a temporary SAS data set named Work.States.

The following input data illustrates enclosing values in quotation marks when you want to avoid separating their values by the space between the words.

```
Region State Capital Bird
South Georgia Atlanta 'Brown Thrasher'
South 'North Carolina' Raleigh Cardinal
North Connecticut Hartford Robin
West Washington Olympia 'American Goldfinch'
Midwest Illinois Springfield Cardinal
```

You can submit the following code to import the file.

```
options validvarname=v7;
filename stdata 'C:\Users\Student1\cert\state_data.txt' lrecl=100;  /*#1*/
proc import datafile=stdata                                        /*#2*/
  dbms=dlm
  out=states
  replace;
  delimiter=' ';                                                   /*#3*/
  getnames=yes;
run;
proc print data=states;
run;
```

1 Specify the fileref and the location of the file. Specify the LRECL= system option if the file has a fixed-length format. The LRECL= system option specifies the default logical record length to use when reading external files.

2 Specify the input file and specify that it is a delimited file. The DBMS= option specifies the type of data to import. If the delimiter type is a character other than TAB or CSV, then the DBMS= option is DLM. The REPLACE option overwrites an existing SAS data set. The OUT= option identifies the output SAS data set.

3 Specify a blank value for the DELIMITER statement. Set the GETNAMES= statement to YES to generate variable names from the first row of data.

Output 4.4 *PROC PRINT Output: Work.States Data Set*

Obs	Region	State	Capital	Bird
1	South	Georgia	Atlanta	Brown Thrasher
2	South	North Carolina	Raleigh	Cardinal
3	North	Connecticut	Hartford	Robin
4	West	Washington	Olympia	American Goldfinch
5	Midwest	Illinois	Springfield	Cardinal

Example: Importing a Comma-Delimited File with a CSV Extension

This example imports a comma-delimited file and creates a temporary SAS data set Work.Shoes. Boot.csv is a comma-separated value file that is a delimited-text file and that uses a comma to separate values.

```
options validvarname=v7;
proc import datafile='C:\Users\Student1\cert\boot.csv'  /*#1*/
  dbms=csv
  out=shoes
  replace;
  getnames=no;                                          /*#2*/
run;
proc print data=work.shoes;
run;
```

1 Specify the input file. DATAFILE= specifies the input data file, and OUT= specifies the output data set. The DBMS= specifies the type of data to import. If the file type is CSV, then the DBMS= option is CSV. The REPLACE option overwrites an existing SAS data set.

2 Set the GETNAMES= statement to NO to not use the first row of data as variable names.

Output 4.5 *PROC PRINT Output: Work.Shoes Data Set*

Obs	VAR1	VAR2	VAR3	VAR4	VAR5	VAR6	VAR7
1	Africa	Boot	Addis Ababa	12	29761	191821	769
2	Asia	Boot	Bangkok	1	1996	9576	80
3	Canada	Boot	Calgary	8	17720	63280	472
4	Central America/Caribbean	Boot	Kingston	33	102372	393376	4454
5	Eastern Europe	Boot	Budapest	22	74102	317515	3341
6	Middle East	Boot	Al-Khobar	10	15062	44658	765
7	Pacific	Boot	Auckland	12	20141	97919	962
8	South America	Boot	Bogota	19	15312	35805	1229
9	United States	Boot	Chicago	16	82483	305061	3735
10	Western Europe	Boot	Copenhagen	2	1663	4657	129

Example: Importing a Tab-Delimited File

This example imports a tab-delimited file and creates a temporary SAS data set Work.Class.

```
proc import datafile='C:\Users\Student1\cert\class.txt'    /*#1*/
  dbms=tab
  out=class
  replace;
  delimiter='09'x;                                         /*#2*/
run;
proc print data=class;
run;
```

1 Specify the input file. DATAFILE= specifies the input data file, and OUT= specifies the output data set. DBMS= specifies the type of data to import. If the file type is TXT, then the DBMS= option is TAB. The REPLACE option overwrites an existing SAS data set. GETNAMES= statement defaults to YES.

2 Specify the delimiter. On an ASCII platform, the hexadecimal representation of a tab is '09'x. On an EBCDIC platform, the hexadecimal representation of a tab is a '05'x.

Output 4.6 *PROC PRINT Output of Work.Class*

Obs	Name	Gender	Age
1	Louise	F	12
2	James	M	12
3	John	M	12
4	Robert	M	12
5	Alice	F	13
6	Barbara	F	13
7	Jeffery	M	13
8	Carol	F	14
9	Judy	F	14
10	Alfred	M	14
11	Henry	M	14
12	Jenet	F	15
13	Mary	F	15
14	Ronald	M	15
15	William	M	15
16	Philip	M	16

Reading Microsoft Excel Data with the XLSX Engine

Running SAS with Microsoft Excel

The examples in this section are based on SAS 9.4 64-bit running with Microsoft Office 2016 64-bit on Microsoft Windows 64-bit.

This configuration does not require the SAS/ACCESS PC Files Server. If SAS runs in a UNIX environment and needs to access Excel files on Microsoft Windows, you must license the SAS/ACCESS PC Files Server.

Steps for Reading Excel Data

To read the Excel workbook file, SAS must receive the following information in the DATA step:

- a libref to reference the Excel workbook to be read
- the name of the Excel worksheet that is to be read

The table below outlines the basic statements that are used in a program that reads Excel data and creates a SAS data set from an Excel worksheet. The PROC CONTENTS and PROC PRINT statements are not requirements for reading Excel data and creating a SAS data set. However, these statements are useful for confirming that your Excel data has successfully been read into SAS.

Table 4.1 *Basic Steps for Reading Excel Data into a SAS Data Set*

Task	Statement	Example
Reference an Excel workbook file	SAS/ACCESS LIBNAME statement	LIBNAME cert `libname cert xlsx 'C:\Users\Student1\cert\exercise.xlsx';`
Write out the contents of the SAS Library	PROC CONTENTS	`proc contents data=cert._all_;`
Execute the PROC CONTENTS statement	RUN statement	`run;`
Name and create a new SAS data set	DATA statement	`data work.stress;`
Read in an Excel worksheet (as the input data for the new SAS data set)	SET statement	`set cert.ActLevel;`
Execute the DATA step	RUN statement	`run;`
View the contents of a particular data set	PROC PRINT	`proc print data=stress;`
Execute the PROC PRINT statement	RUN statement	`run;`

Here is the syntax for assigning a libref to an Excel workbook.

The LIBNAME Statement

To assign a libref to a database, use the LIBNAME statement. The SAS/ACCESS LIBNAME statement associates a SAS libref with a database, schema, server, or a group of tables and views.

Syntax, SAS/ACCESS LIBNAME statement:

LIBNAME <*libref*>**XLSX** <*'physical-path-and-filename.xlsx'*><*options*>**;**

- *libref* is a name that you associate with an Excel workbook.
- *XLSX* is the SAS LIBNAME engine name for an XLSX file format. The SAS/ACCESS LIBNAME statement associates a libref with an XLSX engine that supports the connections to Microsoft Excel 2007, 2010, and later files.

  Important: The engine name XLSX is required.

  When reading XLSX data, the XLSX engine reads mixed data (columns containing numeric and character values) and converts it to character data values.

  The XLSX engine allows sequential reading of data only. It does not support random access. Therefore, 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.
- *'physical-path-and-filename.xlsx'* is the physical location of the Excel workbook.

Example:

libname results XLSX **`'C:\Users\Student1\cert\exercise.xlsx';`**

Note: The XLSX engine requires quotation marks for *physical-path-and-filename.xlsx*.

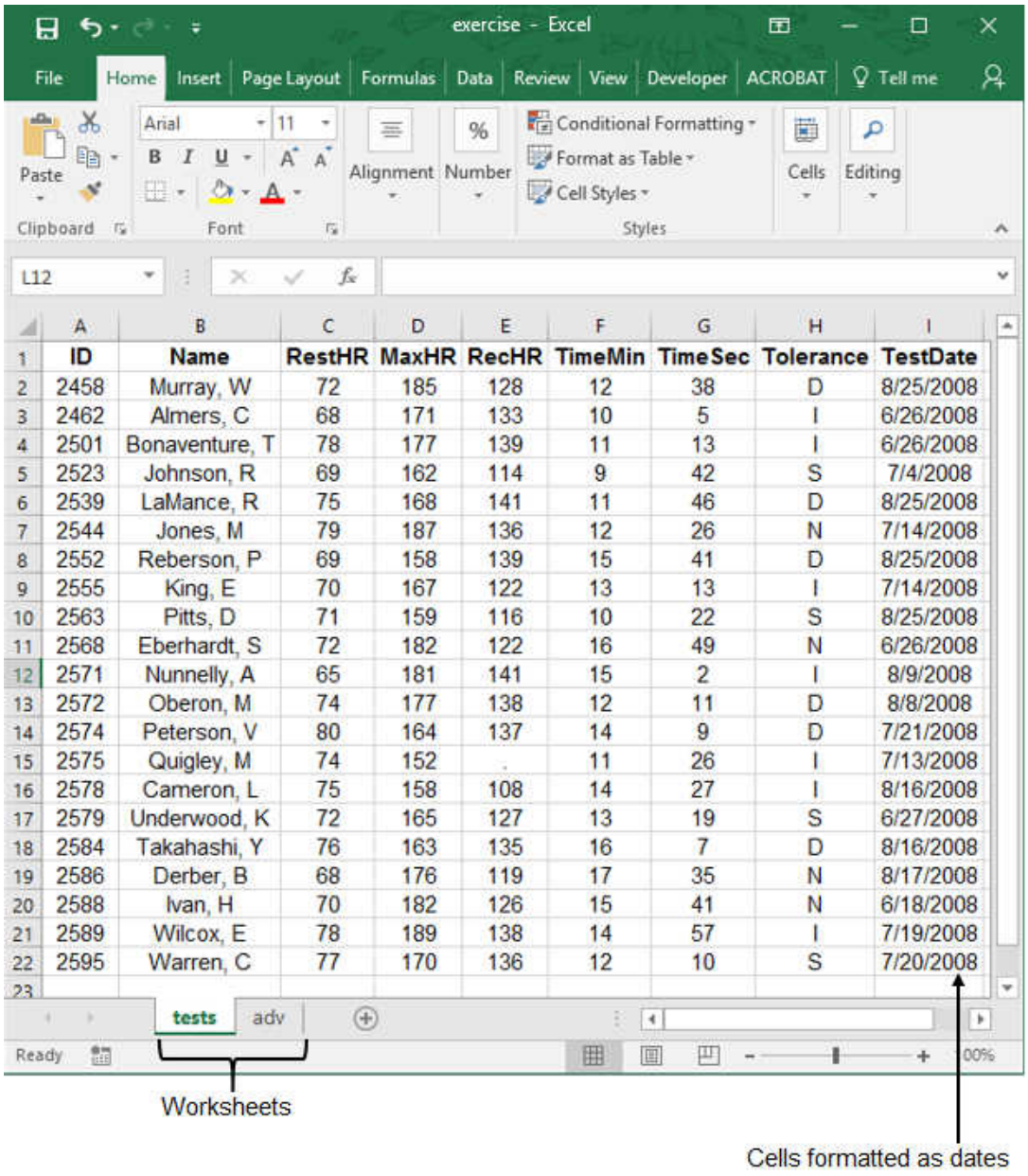
Referencing an Excel Workbook

Overview

This example uses data similar to the scenario used for the raw data in the previous section. The data shows the readings from exercise stress tests that have been performed on patients at a health clinic.

The stress test data is located in an Excel workbook named exercise.xlsx (shown below), which is stored in the location `C:\Users\Student1\cert\`.

Figure 4.3 *Excel Workbook*



	A	B	C	D	E	F	G	H	I
1	ID	Name	RestHR	MaxHR	RecHR	TimeMin	TimeSec	Tolerance	TestDate
2	2458	Murray, W	72	185	128	12	38	D	8/25/2008
3	2462	Almers, C	68	171	133	10	5	I	6/26/2008
4	2501	Bonaventure, T	78	177	139	11	13	I	6/26/2008
5	2523	Johnson, R	69	162	114	9	42	S	7/4/2008
6	2539	LaMance, R	75	168	141	11	46	D	8/25/2008
7	2544	Jones, M	79	187	136	12	26	N	7/14/2008
8	2552	Reberson, P	69	158	139	15	41	D	8/25/2008
9	2555	King, E	70	167	122	13	13	I	7/14/2008
10	2563	Pitts, D	71	159	116	10	22	S	8/25/2008
11	2568	Eberhardt, S	72	182	122	16	49	N	6/26/2008
12	2571	Nunnelly, A	65	181	141	15	2	I	8/9/2008
13	2572	Oberon, M	74	177	138	12	11	D	8/8/2008
14	2574	Peterson, V	80	164	137	14	9	D	7/21/2008
15	2575	Quigley, M	74	152	.	11	26	I	7/13/2008
16	2578	Cameron, L	75	158	108	14	27	I	8/16/2008
17	2579	Underwood, K	72	165	127	13	19	S	6/27/2008
18	2584	Takahashi, Y	76	163	135	16	7	D	8/16/2008
19	2586	Derber, B	68	176	119	17	35	N	8/17/2008
20	2588	Ivan, H	70	182	126	15	41	N	6/18/2008
21	2589	Wilcox, E	78	189	138	14	57	I	7/19/2008
22	2595	Warren, C	77	170	136	12	10	S	7/20/2008
23									

In the sample worksheet above, the date column is defined in Excel as dates. If you right-click the cells and select **Format Cells**, the cells have a category of Date. SAS reads this data just as it is stored in Excel. If the date had been stored as text in Excel, then SAS would have read it as a character string.

To read in this workbook, create a libref to point to the workbook's location:

```
libname certxl XLSX 'C:\Users\Student1\cert\exercise.xlsx';
```

The SAS/ACCESS LIBNAME statement creates the libref Certxl, which points to the Excel workbook exercise.xlsx. The workbook contains two worksheets, Tests and Adv, which are now available in the new SAS library (Results) as data sets.

Referencing an Excel Workbook in a DATA Step

SET Statement

Use the SET statement to indicate which worksheet in the Excel file you want to read.

```
data work.stress;
   set certxl.ActivityLevels;
run;
```

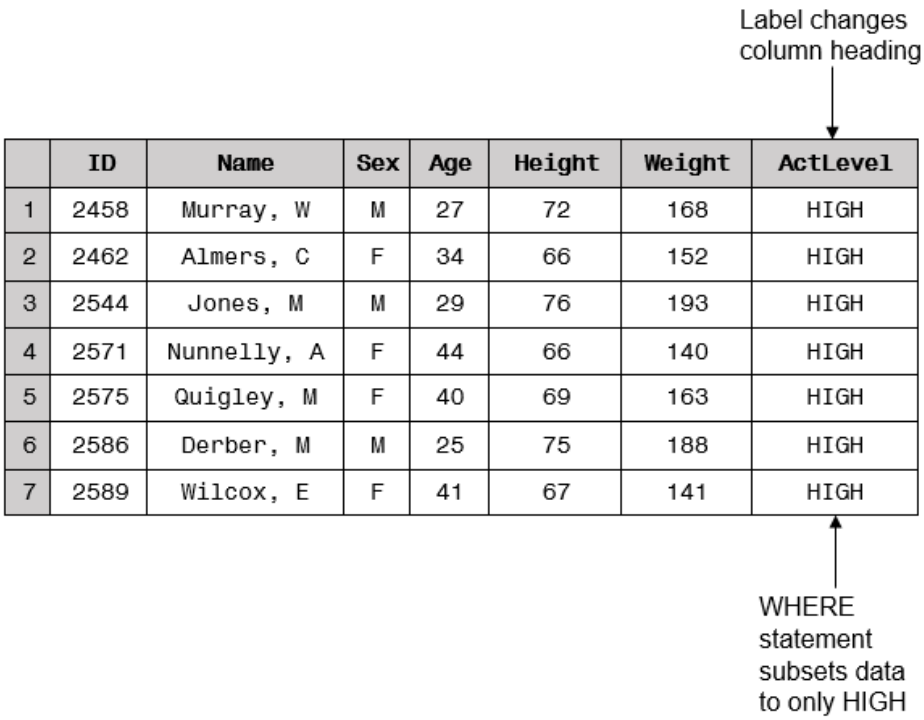
In this example, the DATA statement tells SAS to name the new data set, Stress, and store it in the temporary library Work. The SET statement specifies the libref (the reference to the Excel file) and the worksheet name as the input data.

You can use several statements in the DATA step to subset your data as needed. Here, the WHERE statement is used with a variable to include only those participants whose activity level is HIGH.

```
data work.stress;
   set certxl.ActivityLevels;
   where ActLevel='HIGH';
run;
```

The figure below shows the output for this DATA step in table format.

Figure 4.4 *DATA Step Output*



	ID	Name	Sex	Age	Height	Weight	ActLevel
1	2458	Murray, W	M	27	72	168	HIGH
2	2462	Almers, C	F	34	66	152	HIGH
3	2544	Jones, M	M	29	76	193	HIGH
4	2571	Nunnelly, A	F	44	66	140	HIGH
5	2575	Quigley, M	F	40	69	163	HIGH
6	2586	Derber, M	M	25	75	188	HIGH
7	2589	Wilcox, E	F	41	67	141	HIGH

Name Literals

The SAS/ACCESS LIBNAME statement created a permanent library, Certxl, which is the libref for the workbook file and its location. The new library contains two SAS data sets, which access the data from the Excel worksheets.

Name literals are required with the XLSX engine only when the worksheet name contains a special character or spaces. By default, SAS does not allow special characters in SAS data set names. A SAS *name literal* is a name token that is expressed as a string within quotation marks, followed by the uppercase or lowercase letter *n*. The name literal tells SAS to allow the special character ($) in the data set name.

The following example illustrates reading an Excel worksheet using a name literal. Specify the name of the worksheet in quotation marks with an n following the name. This syntax tells SAS that there are special characters or spaces in the data set name.

```
libname certxl xlsx 'C:\Users\Student1\cert\stock.xlsx';
data work.bstock;
  set certxl.'boots stock'n;
run;
```

Printing an Excel Worksheet as a SAS Data Set

After using the DATA step to read in the Excel data and create a SAS data set, you can use PROC PRINT to produce a report that displays the data set values. In the following example, the PROC PRINT statement displays all the data values for the new data set, Work.Bstock.

```
libname certxl xlsx 'C:\Users\Student1\cert\stock.xlsx';
data work.bstock;
  set certxl.'boots stock'n;
run;
proc print data=work.bstock;
run;
```

Output 4.10 *PROC PRINT Output of Work.Bstock*

Obs	Region	Item	City	Stock
1	Eastern Europe	Boot	Budapest	22
2	Middle East	Boot	Al-Khobar	10
3	Pacific	Boot	Auckland	12

In the following example, the PROC PRINT statement refers to the worksheet Boot Sales and prints the contents of the Excel worksheet that was referenced by the SAS/ACCESS LIBNAME statement.

```
libname certxl xlsx 'C:\Users\Student1\cert\stock.xlsx';
proc print data=cerxl.'boots stock'n;
run;
```

Output 4.11 *PROC PRINT Output Using Name Literals*

Obs	Region	Item	City	Stock
1	Eastern Europe	Boot	Budapest	22
2	Middle East	Boot	Al-Khobar	10
3	Pacific	Boot	Auckland	12